Computational Physics WS 2025

Stefan.Goedecker@unibas.ch

Contents

1 Organization

- Lecture with script: http://comphys.unibas.ch/teaching.htm
- Exercises and projects
  - 'Small' exercises accompanying each lecture: Traditional analytic problems and simple numerical problems (on your own laptop?)
  - Solve 2 'large' projects out of 6 being offered
    Development of a short program to solve a simple physical problem. Can be done at any time by the student either at home or in the physics institute. Programming help can be given to Fortran and Python Programmers.

Credit points:

- 4 credit points for doing the small exercises and passing an oral examination on the course
- 2 additional credit points for the successful solution of the 2 projects.

Outline of the course

- Linux or Mac operating system
- Python programming (tutorial?)
- Computer arithmetic
- Numerical differentiation
- Minimization algorithms
- Atomistic simulations and Molecular dynamics (MD)
  - Inter-atomic potentials
  - Structure determination
  - Vibrational properties
  - Basic Molecular Dynamics algorithms
  - Boundary conditions

- Electrostatic and gravitational forces
  - Barnes Hut algorithm
  - Fast multipole algorithm
  - Fourier methods
  - Ewald and particle mesh methods
  - Multigrid methods
- Statistical mechanics
  - Random numbers
  - Calculations of thermodynamic properties from MD simulations
  - Monte Carlo simulations
  - Global minimization and structure prediction
- Quantum mechanics
  - Numerical solution of the time-independent single particle Schrödinger equation
  - Density functional methods

- Numerical integration methods
  - One-dimensional integrals
  - High-dimensional integrals

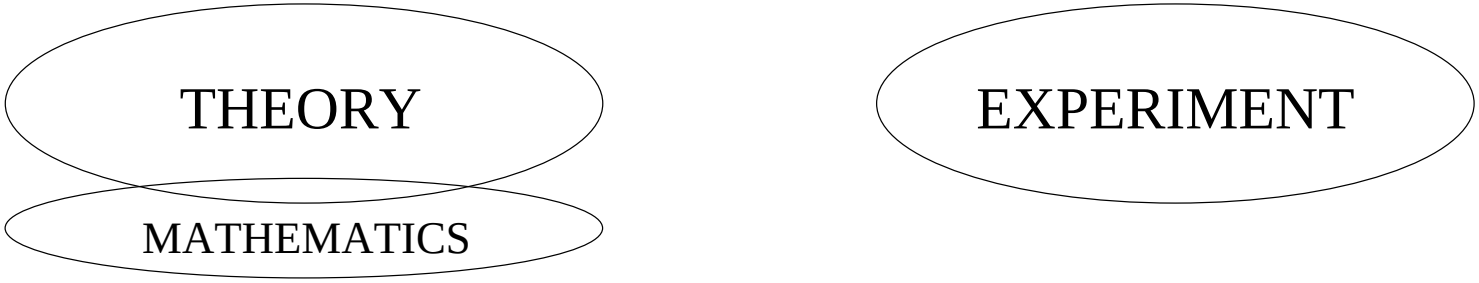
2 Motivating the use of computers in physics

2.1 Computers are changing our daily life

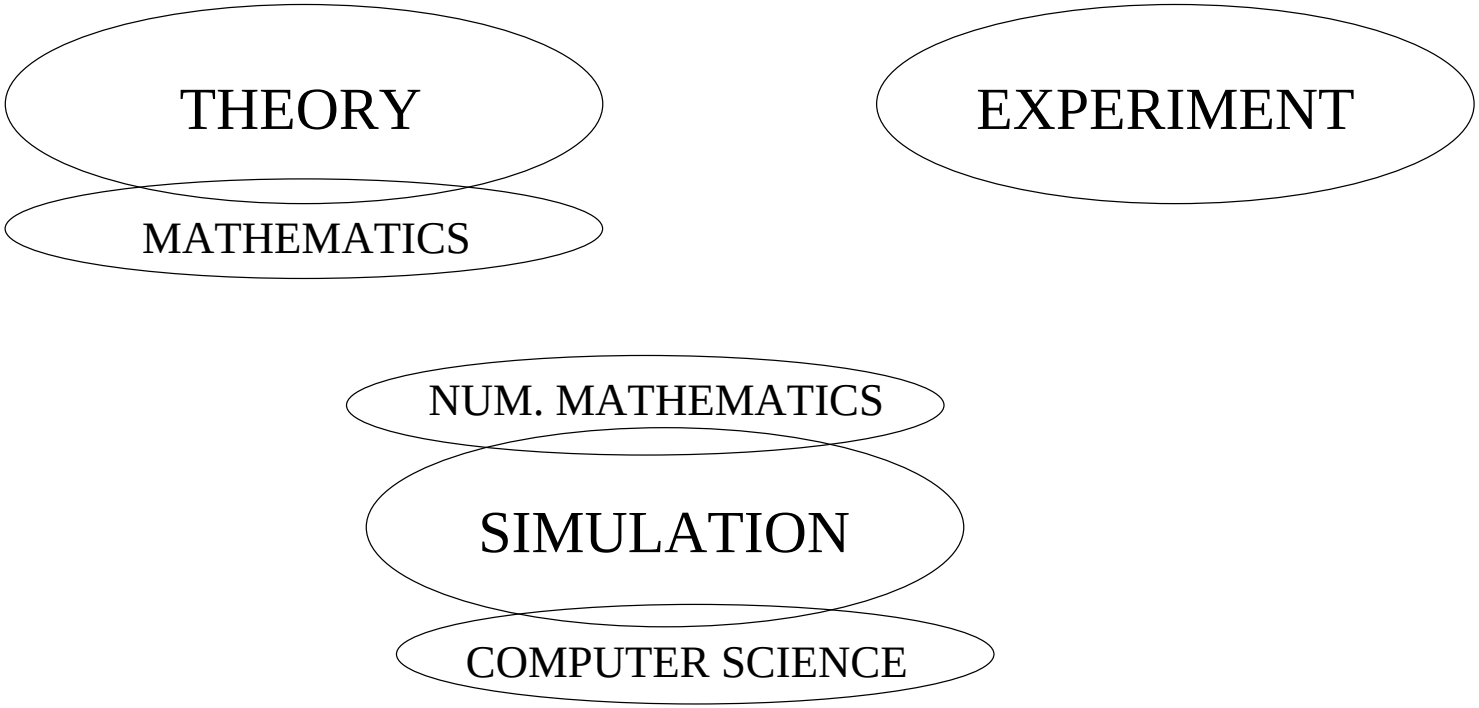
- email
- Internet
- (on-line) data banks
- tele-working
- electronic control systems in cars: ABS, ESP

2.2 Computers are changing the way science is done

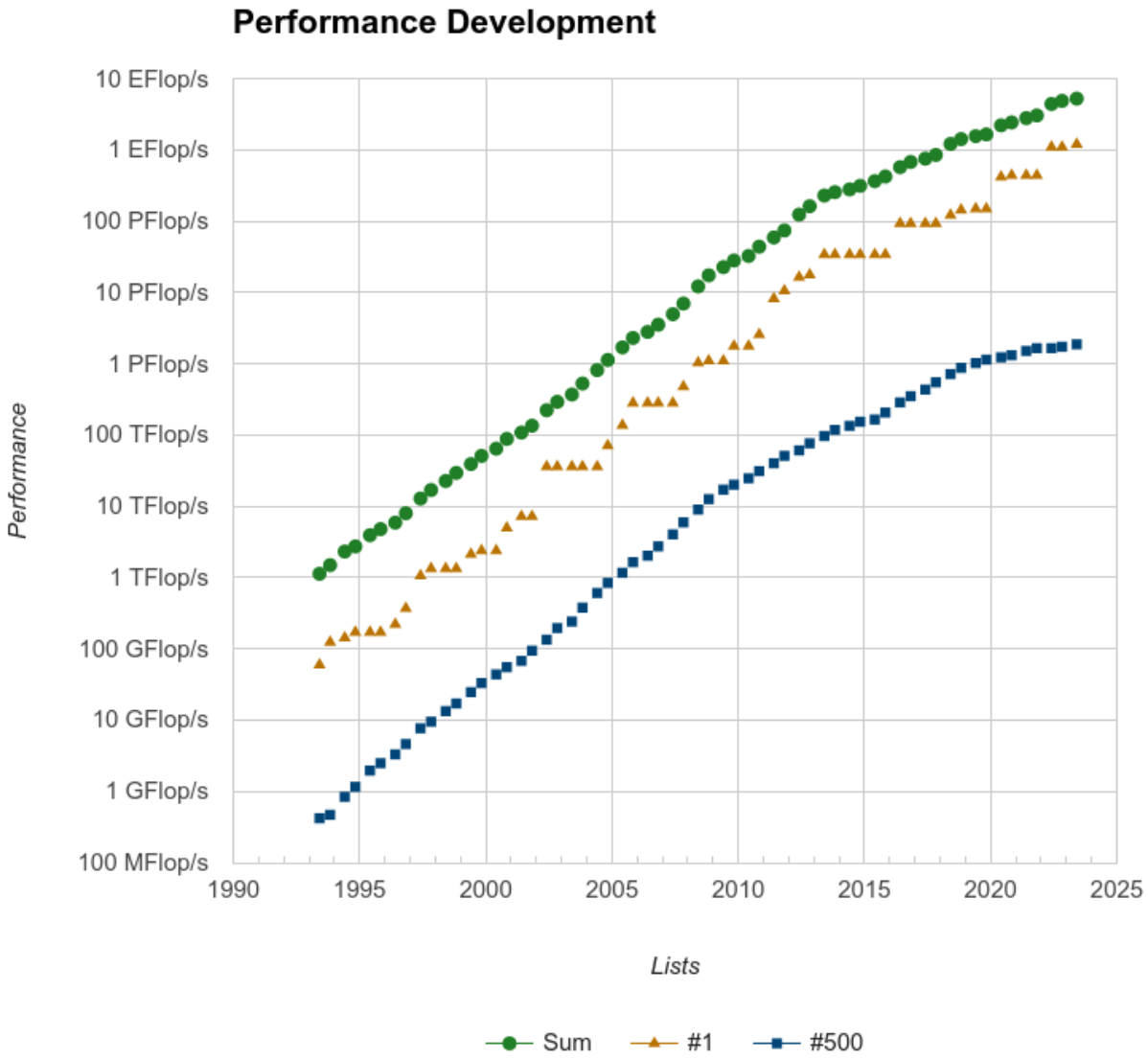
”Old” science



”New” science



2.3 100-fache Zunahme der CPU hours für wissenschaftliche Simulationen pro Decade



2.4 Ingredients of the virtual chemistry/physics laboratory

- Models of the physical reality, e.g. density functional theory for the description of interacting electronic systems or elasticity theory for the description of macroscopic bodies
- Algorithms that allow us to solve the fundamental equations of these models numerically
- Fast computers
- Efficient implementations of the algorithms on modern computers

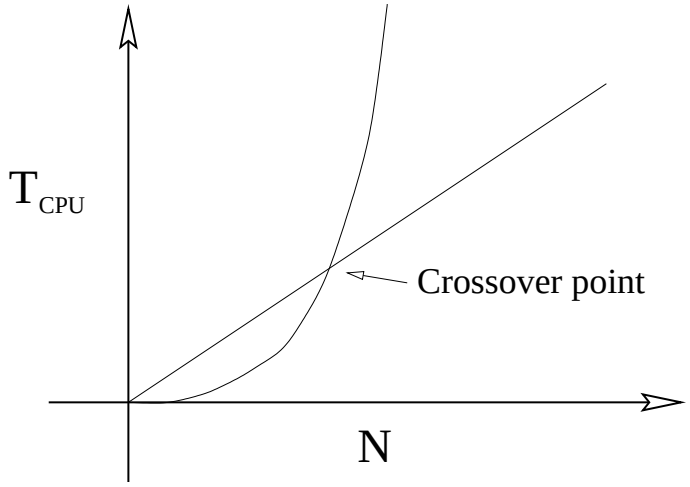
2.5 Scaling behavior of algorithms

$$T_{CPU} = cN^{\gamma} \tag{1}$$

N is some measure of the size of the system to be simulated

- γ large: High complexity algorithm, bad scaling
- γ small: Low complexity algorithm, favorable scaling

Ultimate goal: linear scaling, $T_{CPU} = cN$



In the presence of very powerful computers, bad scaling is the most serious limitation in numerical simulations

2.6 Fast algorithms are among the main achievements of modern mathematics

- Fast Fourier Transform: Frequency analysis with O(N log(N)) operations;
  N = number of data points
- Multi-grid method: Solution of elliptic PDE's with O(N) operations;
  N = the number of grid points
- Fast Wavelet transformation: Multi-resolution analysis with O(N) operations ;
  N = number of data points
- Merge Sort: Sorting N data items with O(N log(N)) operations

2.7 Two sorting algorithms

Ordinary Sorting :
The subroutine below sorts an array A into ascending order:
$A(1) \leq A(2) \leq ... \leq A(N\text{-}1) \leq A(N)$

```
      SUBROUTINE slowsort(N,ARR)
      DIMENSION ARR(N)

      DO J=2,N
        A=ARR(J)
        DO I=J-1,1,-1
          IF(ARR(I) <= A) GOTO 10
          ARR(I+1)=ARR(I)
        ENDDO
        I=0
10      ARR(I+1)=A
      ENDDO

      END
```

The scaling is quadratic: $T_{CPU} \propto N^2$

Merge Sort: $T_{CPU} \propto N \log_2(N)$

1. Any consecutive pair of entries is ordered
2. All consecutive pairs of ordered segments of length 2 are merged into ordered segments of length 4
3. All consecutive pairs of ordered segments of length 4 are merged into ordered segments of length 8
4. The remaining two ordered segments of length 8 are merged into the final ordered result

Exercise [5pt]: *Write a subroutine that implements the merge sort algorithm*

PROJECT: Sorting continously growing data sets

Background

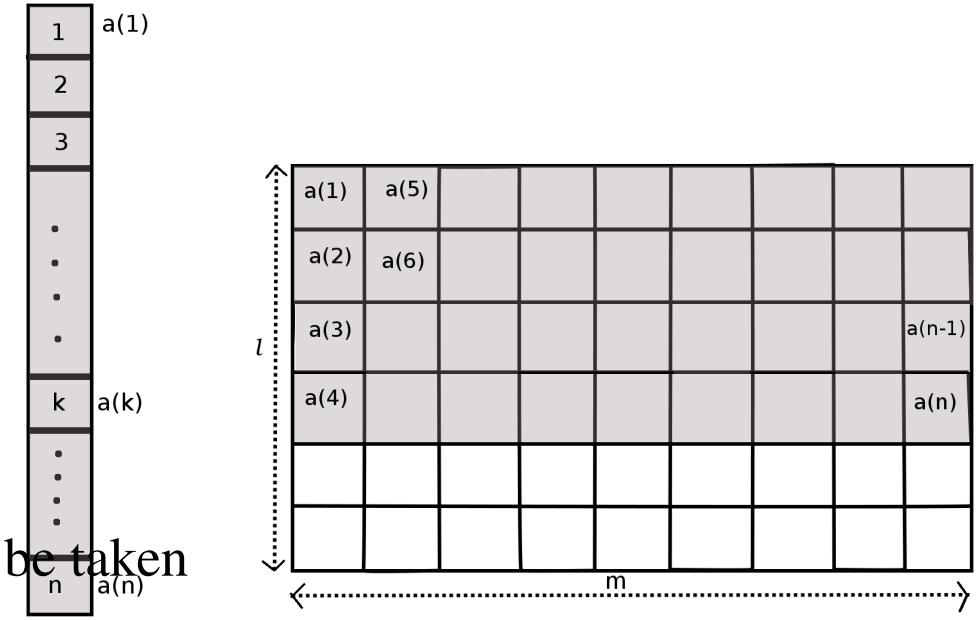
The merge sort algorithm allows to sort a given data set in an efficient way. Frequently the data set is however not constant in time, but increasing continously. For instance new participants have to be added continously in an alphabetically ordered telephon book. Using the merge sort repeatedly after adding one or a few new items to the data set would be inefficient. In this project a method will be introduced, which allows us to maintain continously growing data sets with moderate effort.

We consider a large one dimensional array (a) of numbers which are stored in increasing order, that is $a(n) > a(n-1) > ...a(3) > a(2) > a(1)$. Here n is the length of the array and is in the order of 100k. For simplicity we can assume that the numbers are in the interval [0:1] which means that they can be generated by a simple call to a random number generator (call random_number(r) in Fortran; import random ; num = random.random() in Python). Now we want to insert an element (which is a random number) into this array. According to the value of this new element, it has to inserted in the k^{th} position. This position can be found efficiently with at most $O(\log(n))$ operations using bisection. Then we have to increase the length of our final array to $n+1$ and shift all elements of the old array having the positions $\geq k$. This operation becomes very expensive when n is very large because memory access is typically more expensive on modern computers than numerical operations. We will try to find a different way to do this operation more

efficiently.

Tasks

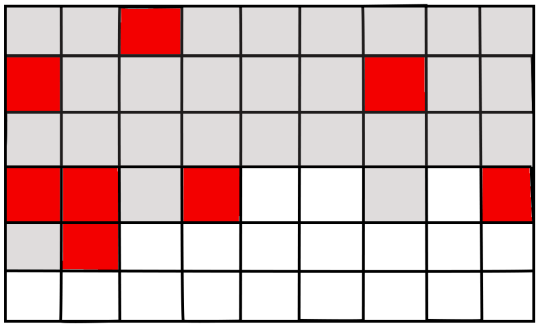
- Write a bisection routine that will find in an ordered array the correct position for inserting a new element. Take as the first search interval the whole array. In each consecutive step the length of interval which contains the correct position is cut into half until the insert position is found.
- We will now convert the large one dimensional data array $a(n)$ into which data have to be inserted to a two dimensional array $b(m,l)$ as shown in Figure below, where the shaded region represents non zero values of the array The value of m and l has to be chosen such that $m \times l > n$, that is, we will have space for more elements in this two dimensional array than in our original one dimensional array.



Here are the steps to be taken

1. Choose the number of columns(m) and rows(l) of the two dimensional array.
2. Store the elements in two dimensional array such that each column contains same number of elements. While storing the elements also store the number of elements in column i in an array $ncolelements(i)$ and also the maximum value of the element in $valindex(i)$.
3. Now if we want to insert an element in our original array we will first find out in which column it belongs to using the value stored in $valindex(:)$.
4. Once we know to which column it belongs we will insert it into that column at the right array position so that the increasing order in preserved. In addition we also update the array $ncolelements(:)$ containing the number of elements

in each column and possibly also $valindex(i)$. In the figure below, the shaded region represents old values of the array and red are the new inserted values



5. If one column gets filled up in this process, then go to the first step and redistribute again.

- Test the method carefully. Write a routine which reconverts the two dimensional array into an one dimensional array and check whether the one dimensional data are correctly ordered at any stage of the continous insertion process.

- Compare the performance of this more sophisticated algorithm with the performance of the trivial method where the insert position is found in a one-dimensional array by bisection and then all the elements beyond the insertion point are shifted. Examine the influence of the size of the data set and of the choice of the side lengths of the two dimensional array on the performance.

2.8 Computational Physics and theory

The basic equations of physics are in general too complicated to be solved by analytical methods. In many cases they can however be solved by numerical methods. Examples:

- Newton's equations of motion can be solved analytically for a system of at most two spherical bodies. No analytic solution is available if one wants to describe for instance the entire solar system with the sun and all its planets. Numerically this can however easily be done.

- The only system found in nature for which the Schrödinger equation can be solved is the hydrogen atom. If one wants to solve this equation even for the simplest molecule, one has to use numerical methods.

- The electrostatic potential can be calculated analytically for simple charge distributions such as point charges, charged lines etc as every student knows from numerous exercises of this type. For realistic and more complicated charge distributions, such as the charge distributions found in a semiconductor device one has again to resort to numerical methods.

- Simulation based on physical laws is also an essential ingredient in engineering. To obtain a stable and save body of a car, computer simulations are just as essential as for the design of the wing of an airplane.

2.9 Computational Physics and experiment

Experiments have several limitations

- Experimental results are usually not the findings of direct observations but of interpreted observations. These assumptions on which the interpretations were based may be wrong or inaccurate. In computer simulations on the contrary, most quantities are directly accessible. One can for instance exactly know the atomic positions of the atoms in a chemical reaction when it is studied with a molecular dynamics simulation.

- Simulations can be done for systems under extreme conditions (e.g. very high temperature or very high pressure) that can not be handled by any experimental equipment.

- If an answer to a physical problem can both be obtained by simulation and by experiment, simulation is frequently cheaper.

2.10 Modern Computers

- Single core: A few Gigaflops/sec (10^9 flops/sec) of peak performance
  Example: a single core has a peak of around 12 Gflops/sec : Clocked at 3 GHz, 4 floating point operations per cycle
- Multi core processor
- GPU: up to 100 Tflops (10^{12} flops/sec) with some 20'000 cores
- Parallel computers: Distributed memory computers: Up to a few thousand nodes (typically with GPU accelerator) connected by a fast network. Programming on such a machine is very different from the programming of a serial machine since one has to tell each processor individually what to do.

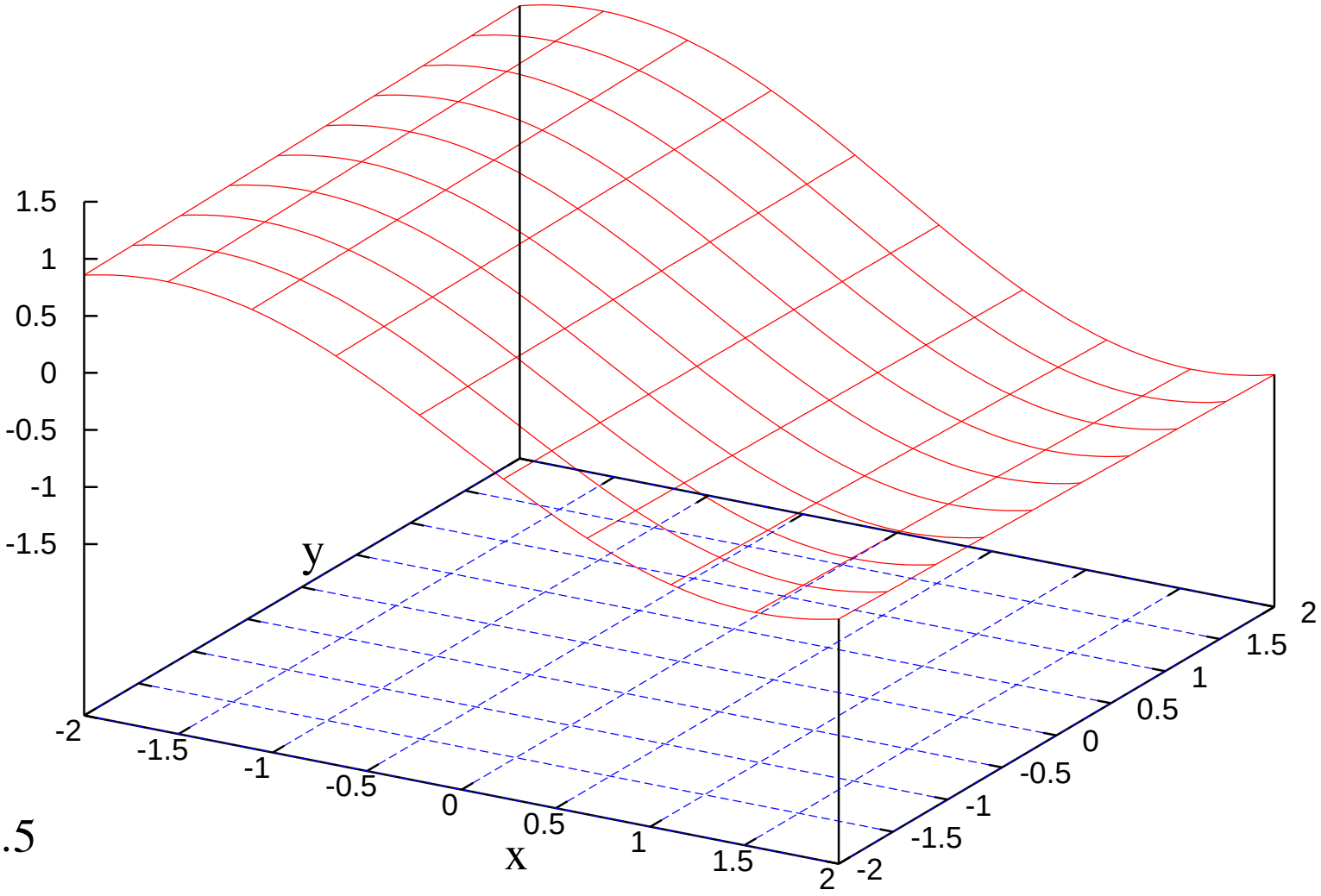
Increased parallelism is the driving force for further performance gains: Swiss National Supercomputer 2025: Exa flop = 10^{18} floating point operations per second

2.11 How are computers used in science

- Number crunching: Numerical calculations using floating point numbers
- Symbolic computer algebra (Mathematica, Maple)
- Data banks (scientific journals and data)
- Electronic communication (email, downloading of programs over the Internet, etc)

Numerical calculations are frequently done on grids

function $f(x,y)$ on a two-dimensional grid: functional values on grid points are stored in an array



grid spacing: $h = .5$

Is the performance of modern computers sufficient

Taking into account the impressive speed of modern computers, one might think that enough performance is available to solve any computational problem in little time. This is actually not true. There are numerous important problems that would require a computer speed that is several orders of magnitude larger than what we have nowadays. To see how the present computational power can easily be used up, let us consider some typical computational problem. We want to follow the time evolution of a three-dimensional system over a certain time period. The time dependent solution of the three-dimensional system will be specified on a numerical grid with let's say at least 100 grid points along each direction. There are thus 10^6 real space grid points. Let us assume that we need 1000 time steps. Hence 10^9 values need to be computed during the simulation. Assuming that the algorithm has linear scaling and that the calculation of each value requires 100 floating point operations we will need 10^{11} operations to perform the simulation. This would take roughly 10 seconds on a single core of a modern workstation. However our assumptions are highly optimistic. In most cases the scaling will not be linear and prefactors can be much larger. Hence there are numerous problems that can not even be solved on large parallel computers nowadays.

3 Computer arithmetic

Traditional mathematics distinguishes between integers, rational and irrational numbers. Computer arithmetic is very different. It does not have rational and irrational numbers. Both are approximated by floating point numbers. And even though computer arithmetic has integers, they have different properties than the integers known from traditional mathematics. These differences are due to the fact that all kinds of numbers have a binary representation on a computer, i.e they are represented by a finite length sequence of bits. A bit can take on the values of 0 and 1.

3.1 Binary representation of integers in Fortran and C

Typically integers are stored using a 32-bit word. The most obvious way would be to use one bit for the sign and the remaining 31 bits for the absolute value. The standard integer representation is however a different one. It is called signed integers via 2's complement. In this convention a nonnegative integer k in the range $0 \leq k \leq 2^{31} - 1$ is stored as the binary representation of k, but a negative integer $-l$, where $1 \leq l \leq 2^{31}$, is stored as the binary representation of the positive integer $2^{32} - l$.
Examples

$$(00000000\,00000000\,00000000\,00000000)_2 \quad = \quad 0$$

$$
\begin{aligned}
(00000000\,00000000\,00000000\,00000001)_2 &= 1\\
(00000000\,00000000\,00000000\,00000010)_2 &= 2\\
(00000000\,00000000\,00000000\,00000011)_2 &= 3\\
(00000000\,00000000\,00000000\,00000100)_2 &= 4\\
(00000000\,00000000\,00000000\,00000101)_2 &= 5\\
(00000000\,00000000\,00000000\,01000111)_2 &= 71\\
(11111111\,11111111\,11111111\,10111001)_2 &= -71\\
(01111111\,11111111\,11111111\,11111111)_2 &= 2^{31}-1\\
(10000000\,00000000\,00000000\,00000000)_2 &= -2^{31}
\end{aligned}
$$

It can easily be verified that the representations of 71 and -71 add up to zero

$$
\begin{aligned}
&(00000000\,00000000\,00000000\,01000111)_2\\
+&(11111111\,11111111\,11111111\,10111001)_2\\
=&\,(1\,00000000\,00000000\,00000000\,00000000)_2
\end{aligned}
$$

since the leftmost bit, called the overflow bit, is discarded.

The basic operations with integers are additions/subtractions and multiplications. Since the result is again an integer it does have a binary machine representation unless it is

too big in absolute value to be represented by the available 32 bits. Such an event is called an overflow. Modern computers, that respect the IEEE (Institute of Electric and Electronic Engineers) standard, do not print an error message upon overflow, unless the program is compiled with special flags. Instead they will assign a wrong integer result to the operation. The programmer has thus to ensure that no overflow will occur. Divisions are also allowed between integers. If the result is a representable integer, such a division will deliver the expected result. If the result is not an integer, the resulting rational number will be rounded to the integer part if the result is positive and to minus the integer part of the absolute value if the result is negative. This rounding is also called rounding towards zero. An useful intrinsic function that is acting on integers is the modulus function.

Declaration and use of integers in Fortran

Since the two types of numbers available on a computer, integers and floating point numbers, behave very differently, the programmer has to declare to which type each variable belongs. A sample program is shown below.

```
program integers
   implicit none
   integer(4) :: i1,i2,i3,k
   integer(4) :: ia(4),ib(4)

   i1=8-2 ; i2=3*5 ; i3=3**3
   write(*,*) 'i1,i2,i3=',i1,i2,i3

   i1=8/2 ; i2=12/3
   write(*,*) 'i1,i2=',i1,i2

   i1=modulo(10,3) ; i2=modulo(-10,3)
   write(*,*) 'i1,i2=',i1,i2

   do k=1,4
     ia(k)=2/k ; ib(k)=-2/k
   enddo
   write(*,'(a,4(x,i3))')'ia=',ia
   write(*,'(a,4(x,i3))')'ib=',ib

end  program
```

3.2 Binary representation of integers in Python

Integers in Python can be of any size but if the integers are in the range $[-2^{30} : 2^{30}]$ the calculations are faster. Variables are typically not declared explicitly as being integers or floating point numbers. The interpreter decides instead what type is reasonable.

```
n=1
for i in range(20000):
       n=n*2
       if i%1000==0:
           print(i,n)
```

The type of numerical constants is specified by the absence/presence of the decimal point.

```
n=1.0
for i in range(20000):
       n=n*2
       if i%1000==0:
           print(i,n)

n=1
for i in range(20000):
       n=n*2.
       if i%1000==0:
           print(i,n)
```

Integers beyond the 32 bit limit are however rarely useful since they can for instance not be an input to most other intrinsic functions.

```
import numpy as  np
n=1
for i in range(10000):
       n=n*2
       y=np.log10(float(n))
       if i%1000==0:
           print(i,n)
```

Also the address space is limited to $2^{31} - 1$ for the 32 bit case and $2^{63} - 1$for the 32 bit case.

```
import numpy as  np
n=33
a_array=np.zeros(2**n)
```

Variables (myint, myfloat) can be forced to be of integer or floating point type with the int(myint) and float(myfloat) function. The type(myvariab) function can return the type of myvariab. However output variables of an elementary operation can be of a different type than the input variables. The output of a standard division is for instance always a floating point number. There is however a second division operation // that behaves like the ordinary division in Fortran. A floating point number can be transformed back into a ratio, but the result is not necessarily identical to the ratio that was used to obtain the floating point number.

```
x=1/3
print(x)

y=x.as_integer_ratio()
print(y)

x=int(1)/int(3)
print(x)

x=1//3
print(x)
```

3.3 Floating point numbers

Floating point representation is based on scientific (exponential) notation. In base 10 a real number x is written as

$$x = \pm S \times 10^E \tag{2}$$

S is called the significand and the integer E the exponent. For computer arithmetic base 2 is used instead of base ten and hence

$$x = \pm S \times 2^E \text{ , where } 1 \leq S < 2 \tag{3}$$

The binary expansion of the significand is

$$S = (b_0.b_1b_2b_3...b_{p-1})_2 \text{ , with } b_0 = 1 \tag{4}$$

A floating point number for which $b_0 = 1$ is called a normalized number. For normalized numbers, it is not necessary to store the first bit b_0. For instance the number 5.5 is represented as

$$5.5 = \frac{11}{2} = (1.011)_2 \times 2^2 \tag{5}$$

A real number such as $1/3$ or $\sqrt{2}$ can not be represented exactly by an expansion of the significand of finite length p.

For scientific computations double precision numbers are a de facto standard, since ordinary single precision does not give sufficient precision. The IEEE double precision standard specifies 1 bit for the sign, 11 bits for the exponent and 52 bits for the significand. The total storage requirement is thus 64 bits which equals 8 bytes. Out of the 11 bits for the exponent, one is lost for the sign and the largest exponent is thus $2^{10} - 1 = 1023$. The largest normalized floating point number is thus approximately $2^{1024} \approx 1.8 \times 10^{308}$. For technical reasons the smallest possible exponent is not -1023, but -1022 which gives the smallest normalized floating point number of $2^{-1022} \approx 2.2 \times 10^{-308}$. By allowing non-normalized floating point numbers for the smallest possible exponent it is actually possible to represent even smaller floating point numbers down to $2^{-1022}2^{-52} = 2^{-1074} \approx 4.9 \times 10^{-324}$. If a result of a floating point operation produces a result outside the range of the smallest and largest floating point number a floating point exception occurs. For a result that becomes too large in magnitude, the IEEE standard has introduced the notation plus or minus 'Infinity'. If it gets too small in magnitude it is set to zero. A floating point exception also occurs if one executes a mathematically forbidden operation, such as diving by zero or taking the square root of a negative number. According to the IEEE standard such a result should be denoted by 'NaN', which stands for 'Not a Number'. Unfortunately many computer manufacturers do not observe the IEEE floating point exception standards. While a Fortran code, compiled without debugging flags, continues to run after encountering a floating point exception, producing in most cases meaningless results, the Python interpretoer will stop with an error message.

Floating point exceptions in Fortran

```
program floating_point
implicit none
integer :: i
real(8) :: x,xi,xr,y

x=1/5 ; y=1.d0/5.d0
write(*,*) x,y

x=2.d0
do i=1,5
  x=x-1.d0
  xi=1.d0/x ; xr=sqrt(x)
  write(*,*) 'i,x,xi,xr ',i,x,xi,xr
enddo

x=2.d0**1020
do i=1,20
x=x*2.d0 ; write(*,*) x
enddo

x=2.d0**(-1060)
do i=1,20
x=x/2.d0 ; write(*,*) x
enddo
end program
```

3.4 Rounding

Unless overflow occurs, the sum or the product of two machine integers is again am exact machine integer. This does not hold true for floating point numbers. Their sum or product will in general require more than the number of bits used to represent the two input numbers. Hence any floating point operation has to be followed by a rounding operation. For this reason floating point arithmetic is never exact arithmetic. Floating point numbers have the property that the distance between two neighboring numbers is approximately proportional to their magnitude. Hence rounding to the closest floating point number introduces a relative error that is roughly constant. The gap between a floating point number x and the next larger floating point number that is larger in magnitude is called $ulp(x)$, where ulp stands for Unit in the Last Place. $ulp(1)$ is called the machine epsilon. The machine epsilon thus gives the smallest number that will produce a result different from 1 when added to 1.

To visualize the properties of the floating point number system, let us introduce a toy floating point number system, where the significand has only 3 bits and where the exponent can take on only the values -1,0,1. We will assume normalized numbers such that $b_0 = 1$.

$$\pm(b_0.b_1b_2)_2 \times 2^E \tag{6}$$

For $E = 1$ the possible floating point numbers are

$$(10.0)_2 = 2 \quad ; \quad (10.1)_2 = 2.5 \quad ; \quad (11.0)_2 = 3 \quad ; \quad (11.1)_2 = 3.5$$

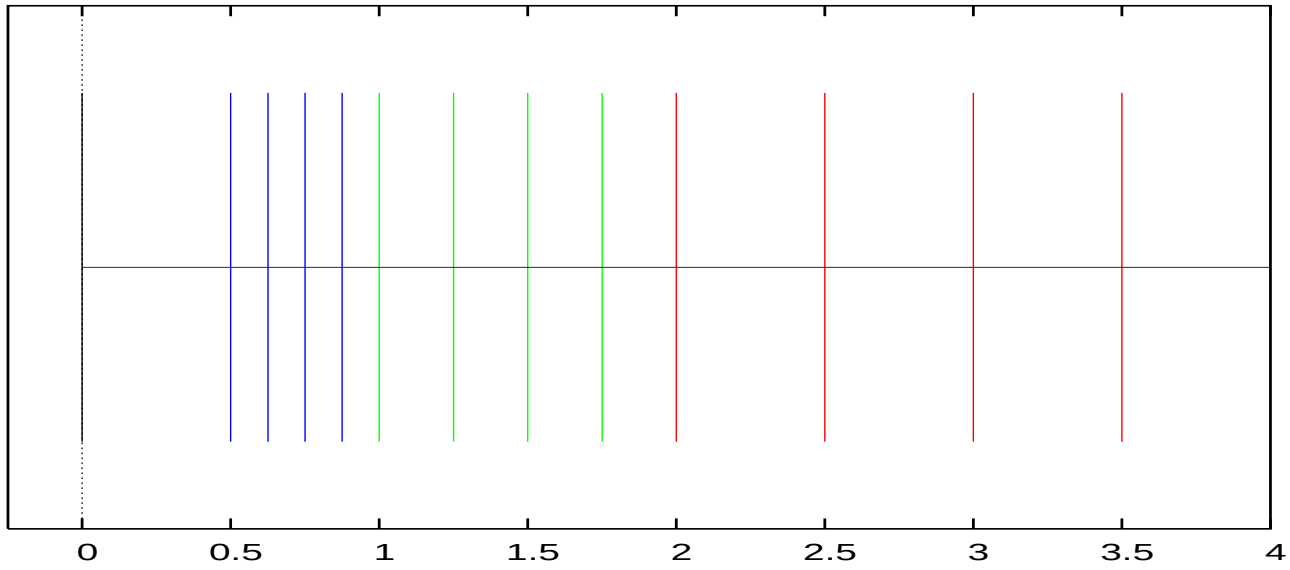
for $E = 0$ they are

$$(1.00)_2 = 1 \quad ; \quad (1.01)_2 = 1.25 \quad ; \quad (1.10)_2 = 1.5 \quad ; \quad (1.11)_2 = 1.75$$

and for $E = -1$ they are

$$(.100)_2 = .5 \quad ; \quad (.101)_2 = .625 \quad ; \quad (.110)_2 = .75 \quad ; \quad (.111)_2 = .875$$

In addition there is the unnormalized floating point number zero $(0.00)_2$



For the toy system it is easy to see that $ulp(1.) = 1/4$. For a general binary floating point number $ulp(1.) = \left(\frac{1}{2}\right)^{p-1}$ where p is defined in Eq. 4. The distance between the first floating point number below 1. and 1. is $1/8$. In the general case it is $\left(\frac{1}{2}\right)^{p}$. It follows that the fractional error $\frac{ulp(x)}{|x|}$ introduced by rounding is in the interval

$$\left(\frac{1}{2}\right)^{p} \leq \frac{ulp(x)}{|x|} \leq \left(\frac{1}{2}\right)^{p-1} \tag{7}$$

For the IEEE double precision floating point system where $p = 53$ we thus get

$$1.1 \times 10^{-16} \leq \frac{ulp(x)}{|x|} \leq 2.2 \times 10^{-16} \tag{8}$$

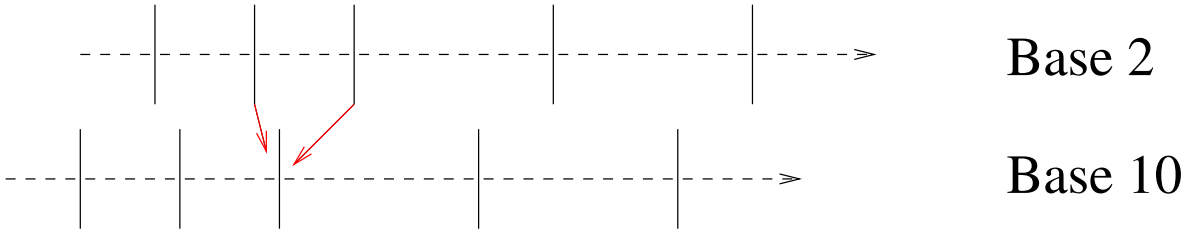
We can thus expect that due to rounding a single floating point operation will give a result with an accuracy of nearly 16 decimal places. Since one is doing typically millions of floating point operations the error between the exact arithmetic result and the floating point arithmetic result can grow larger, but for a stable problem it should not grow dramatically. Empirically it turns out that in stable large scale calculations one can expect an accuracy of 11 to 13 digits.

Because of the rounding error it does usually not make sense to print out the results of floating point operations with more than 15 digits. There is however one exception,

namely if one wants to restart a program with exactly the same values that the program used when it stopped. In this case we have to make sure that a conversion from binary to decimal and back returns the original binary floating point number. The requirement is that the decimal floating point system has to be at any point at least two times denser than the original binary floating point system. The best resolution we can expect from the IEEE double precision standard is $\frac{1}{2}^{53}$. Hence we have the condition that

$$\left(\frac{1}{10}\right)^q < \frac{1}{2}\left(\frac{1}{2}\right)^{53} \tag{9}$$

where q is the number of decimal places in the decimal system. The smallest integer value of q satisfying this condition is $q = 17$. What can go wrong if we do not double the resolution is shown in the figure below. So we have to write all numbers with 17 decimal places into a restart file if we want to be sure that we get back our initial binary floating point number.



3.5 Cancellation

Relative precision is lost in floating point arithmetic when two numbers are subtracted. This loss is worst if the two numbers to be subtracted are similar in magnitude. The phenomenon of cancellation is not only found in binary floating point numbers but also in decimal floating point numbers. Since we are more familiar with the decimal system, cancellation will be demonstrated for a decimal system with 8 decimal places. This means that we have an relative error of 10^{-9}. Let us now assume that we want to subtract the two floating point numbers x and y where

$$x = 0.12345678$$

$$y = 0.12345578$$

The result is obviously $z = x - y = 0.00000100 = .1e-5$. Even though z is again a floating number with a potential relative accuracy of 10^{-9} it has in reality a much lower accuracy, namely a relative accuracy of 10^{-4}. In order to get z with 8 significant decimal places, it would be necessary to have a representation of x and y with 13 decimal places. For instance with

$$x = 0.1234567800009$$

$$y = 0.1234557800000$$

we obtain $z = 0.0000010000009 = .10000009.e-5$

Catastrophic Cancellation

Various instances can be encountered where floating point arithmetic gives wrong results. An experienced programmer can usually detect such problems and eliminate them. As an example let us look at the expression

$$\sqrt{1.d0 - x} - 1.d0 \tag{10}$$

and let us assume that x is very small, i.e. $x = 1.d - 20$. Since x is smaller than the machine epsilon $1.d0 - x$ will be equal to $1.d0$ and $\sqrt{1.d0 - x} - 1.d0 = 0$. The true result is approximately $-.5d - 20$ which can easily be represented by a floating point number. We have thus a relative error in our result that is of the order of one. In this case the problem can be avoided by multiplying numerator and denominator by $\sqrt{1.d0 - x} + 1.d0$

$$\sqrt{1.d0 - x} - 1.d0 = \frac{(\sqrt{1.d0 - x} - 1.d0)(\sqrt{1.d0 - x} + 1.d0)}{\sqrt{1.d0 - x} + 1.d0} = \frac{-x}{\sqrt{1.d0 - x} + 1.d0} \tag{11}$$

Exercise [1pt]: *Calculate* $\sqrt{1.d0 - x} - 1.d0$ *for* $x = 1.d - 2, 1.d - 4, 1.d - 6, 1.d - 8, 1.d - 10, 1.d - 12, 1.d - 14, 1.d - 16, 1.d - 18$ *in both ways and compare the results. Which result is more reliable?*

Exercise [1pt]: *Can it be taken for granted that in floating point arithmetic*

- $$(x+y)-x=y$$

- $$1.d0/(1.d0/y)=y$$

  Hint: Consider the toy system with $x=.875$, *1/.875 = 1.14286, 1/1.25 = .8*

- $$(y/x)*x=y$$

  Hint: Consider the toy system with $x=1.5$, $y=1.75$, *1.75/1.5 = 1.166667, 1.25*1.5 = 1.875*

- $$x*(y+z)=x*y+x*z$$

In order to transfer the results of the toy system to real floating point arithmetic, write a computer program. To make sure that the results of any operation are rounded before they are used for the next operation, write intermediate results into variables and compile without any optimization.

4 Numerical differentiation

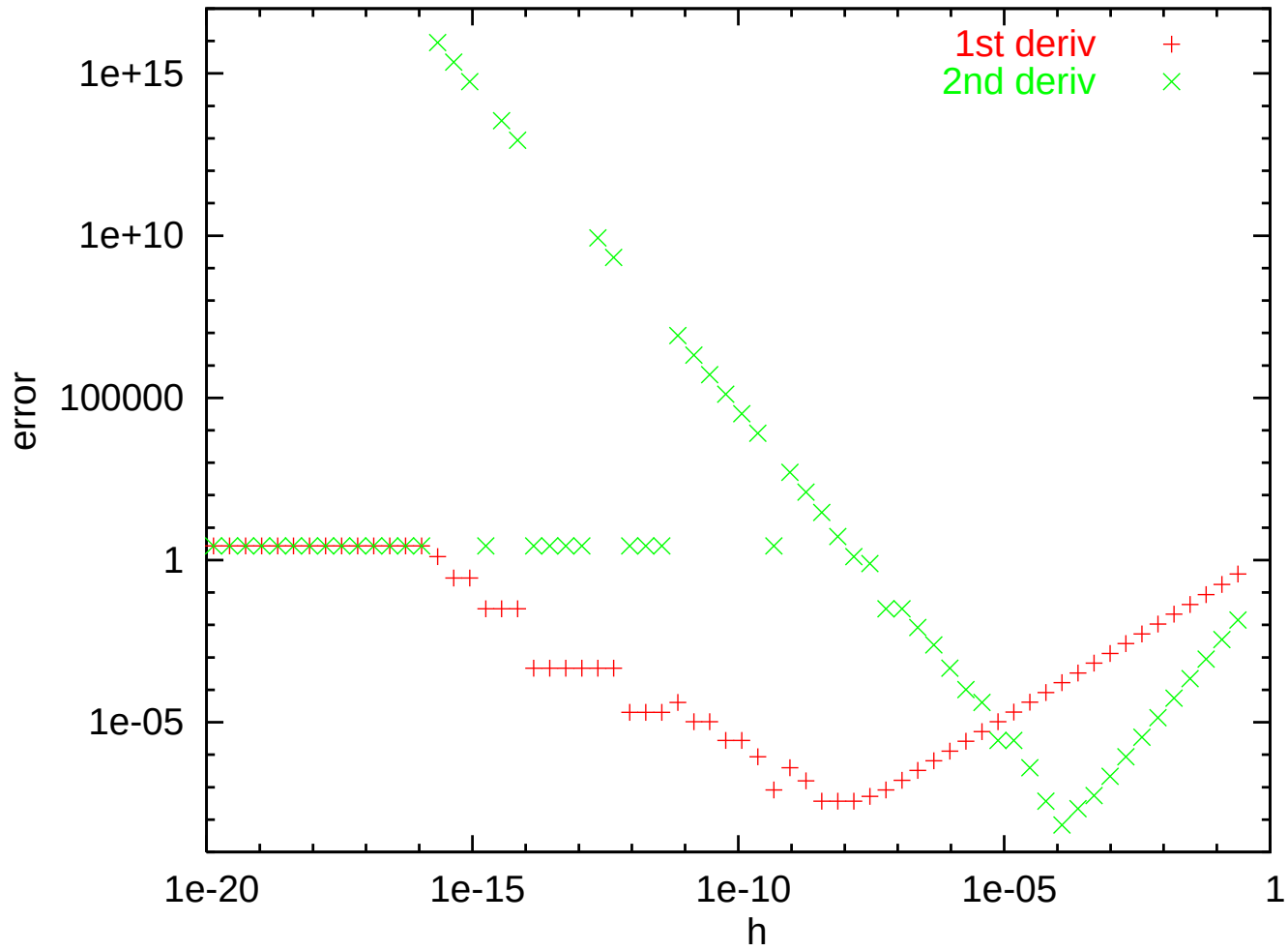
In mathematics the first derivative of a function $f(x)$ is defined as

$$f'(x) = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h} \approx \frac{f(x+h) - f(x)}{h} + O(h) \tag{12}$$

and the second derivative as

$$f''(x) = \lim_{h\to 0} \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2) \tag{13}$$

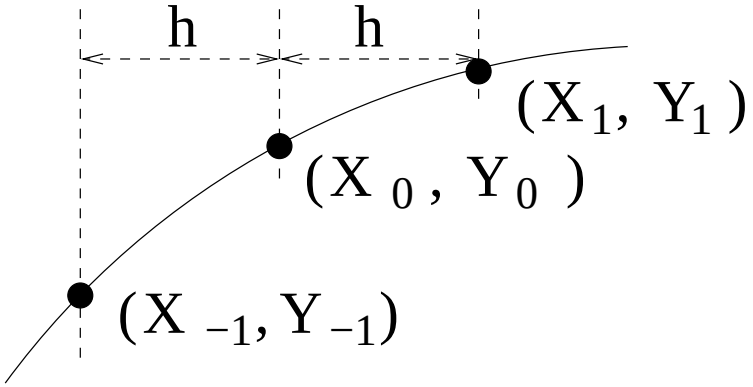
The formulas tell us that if we want to have an accurate result we have to choose h sufficiently small. For example if we want to calculate both derivatives with 10 decimal places h should be of order of 1.d-10 for the first derivative and of the order of 1.d-5 for the second derivative. Unfortunately this accuracy can not be obtained numerically because of catastrophic cancellation effects. The error of both derivatives of the exponential function evaluated at $x = 1.d0$ as a function of h is shown below



Using the above derivative formulas it is not possible to calculate derivatives with high accuracy. If higher accuracy is required one has to use derivative formulas with a much higher accuracy which is of the order of $O(h^{16})$. In this way one should obtain for a $h \approx 1.d-1$ an accuracy of 1.d-16, i.e. machine precision. Due to cancellation effects we will perhaps loose one digit and end up with a numerical accuracy of 1.d-15

Derivation of finite difference formulas for derivatives

Finite difference formulas that approximate derivatives of continuous and differentiable functions are based on the fact that such a function can locally be approximated with arbitrary accuracy by a polynomial. Let us first rederive Equations 12 and 13.



For Eqn. 12 we find a linear function that goes through the points (x_0, y_0) and $(x_1 = x_0 + h, y_1)$. It is easy to see that this polynomial is given by

$$p_1(x) = y_0 \frac{x_1 - x}{h} + y_1 \frac{x - x_0}{h} \tag{14}$$

The derivative of this polynomial is given by

$$p_1'(x) = -y_0 \frac{1}{h} + y_1 \frac{1}{h} = \frac{y_1 - y_0}{h} = y_0 \frac{-1}{h} + y_1 \frac{1}{h} \tag{15}$$

which is identical to Eq. 12. A polynomial that goes through the points $(x_{-1},y_{-1}),(x_0,y_0)$ and (x_1,y_1) is given by

$$p_2(x) = y_{-1}\frac{(x_0-x)(x_1-x)}{2h^2} + y_0\frac{(x_1-x)(x-x_{-1})}{h^2} + y_1\frac{(x-x_0)(x-x_{-1})}{2h^2} \tag{16}$$

The first and second derivative of the polynomial evaluated at $x = x_0$ is given by

$$p_2'(x_0) = \frac{y_1-y_{-1}}{2h} = y_{-1}\frac{-1}{2h} + y_1\frac{1}{2h} \tag{17}$$

$$p_2''(x_0) = y_{-1}\frac{1}{h^2} + y_0\frac{-2}{h^2} + y_1\frac{1}{h^2} \tag{18}$$

The above equation is identical to Eq. 13. The principle for constructing higher order finite difference formulas is simple. One has to fit higher order polynomials to the function f for which one wants to calculate the derivative and then calculate analytically the derivative for this polynomial. For even degree polynomials the resulting finite difference formula will be a weighted sum of m functional values to the right and m functional values to the left of the point x_0 at which one wants to calculate the l-th derivative.

$$f^{(l)}(x_0) \approx p_{2m}^{(l)}(x_0) = \sum_{i=-m}^{m} y_i\frac{c_i}{h^l} \tag{19}$$

where we have suppressed in our notation the fact that the set of coefficients c_i differs for different l.
The problem is that it is very cumbersome to derive by hand the coefficients c_i in Eq. 19 for large m or l. Here symbolic computation can come to our help. The following Mathematica program finds the 16-th degree polynomial $p_{16}(x)$ that goes through 17 points of a function, differentiates it and evaluates it at x_0:

```
f[x_]:=Evaluate[InterpolatingPolynomial[{{-8,ym8},{-7,ym7},{-6,ym6},
       {-5,ym5},{-4,ym4},{-3,ym3},{-2,ym2},{-1,ym1},{0,y0},{1,yp1},
       {2,yp2},{3,yp3},{4,yp4},{5,yp5},{6,yp6},{7,yp7},{8,yp8}},x]]
ff=Simplify[ReplaceAll[D[f[x],x],{x -> 0}]]
```

The output one obtains is the following:

```
Out[4]= (-640640 ym1 + 224224 ym2 - 81536 ym3 + 25480 ym4 - 6272 ym5 +

>       1120 ym6 - 128 ym7 + 7 ym8 + 640640 yp1 - 224224 yp2 + 81536 yp3 -

>       25480 yp4 + 6272 yp5 - 1120 yp6 + 128 yp7 - 7 yp8) / 720720
```

Lets now come back to our old problem of calculating numerically the derivative of the exponential function with very large precision at $x = 1.d0$. Using the finite difference

formula Eq. 19 with the above coefficients c_i calculated with Mathematica, one can indeed calculate the derivative with 15 correct decimals for $h \approx 1.d-1$.

Exercise [2pt]: *Write a short computer program to calculate the first derivative of the exponential function with high accuracy using the coefficients c_i given in Table 1.*

Using an analysis based on Taylor expansions we get the finally the following formulas for the error:

$$|f^{(l)}(x_0) - p_{2m}^{(l)}(x_0)| < \begin{cases} const\ h^{2m-l+1} & \text{if } l \text{ is odd} \\ const\ h^{2m-l+2} & \text{if } l \text{ is even} \end{cases} \tag{20}$$

The plot on the next page numerically demonstrates the above error formula for the case where $m = 2$.

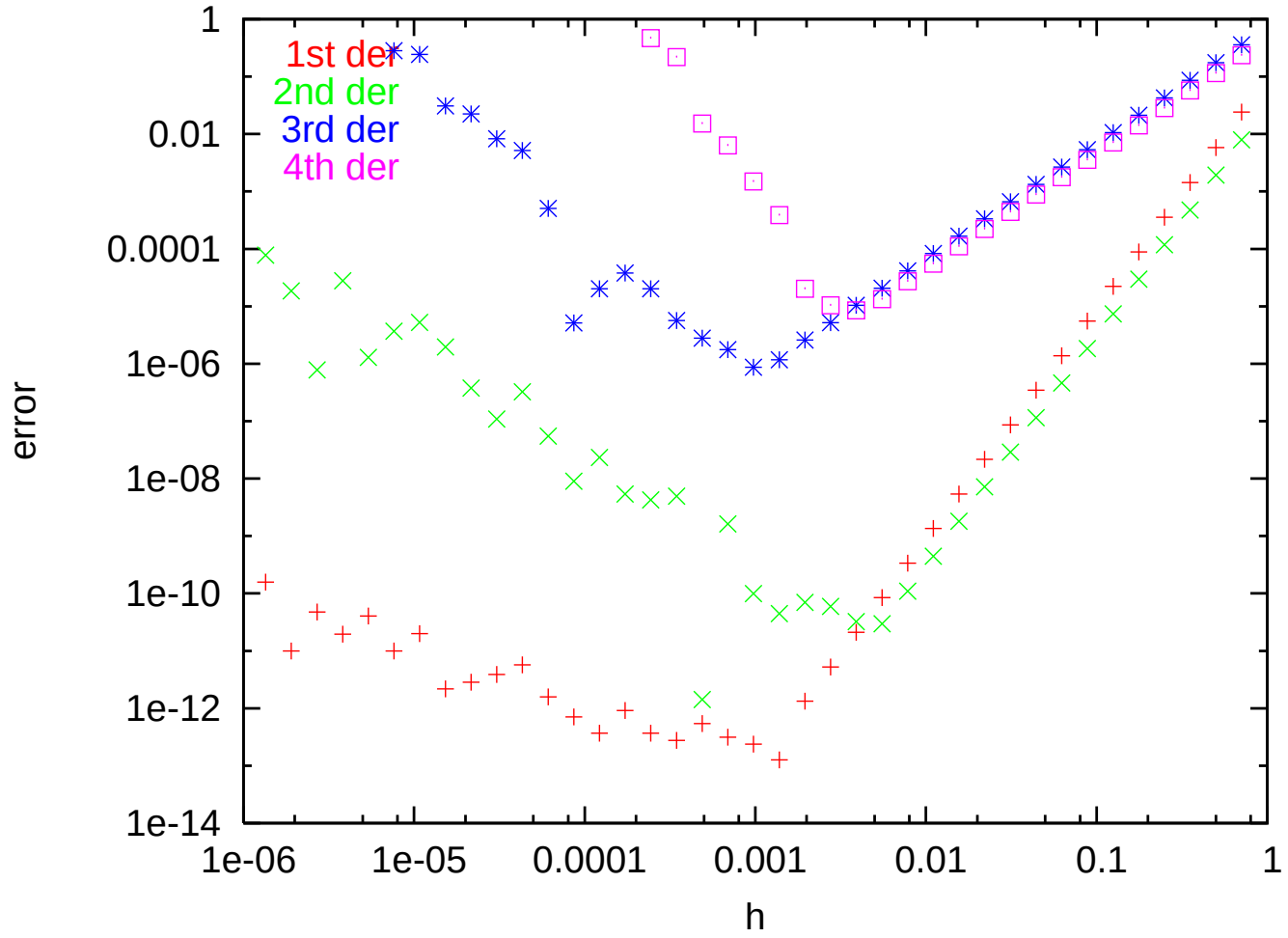
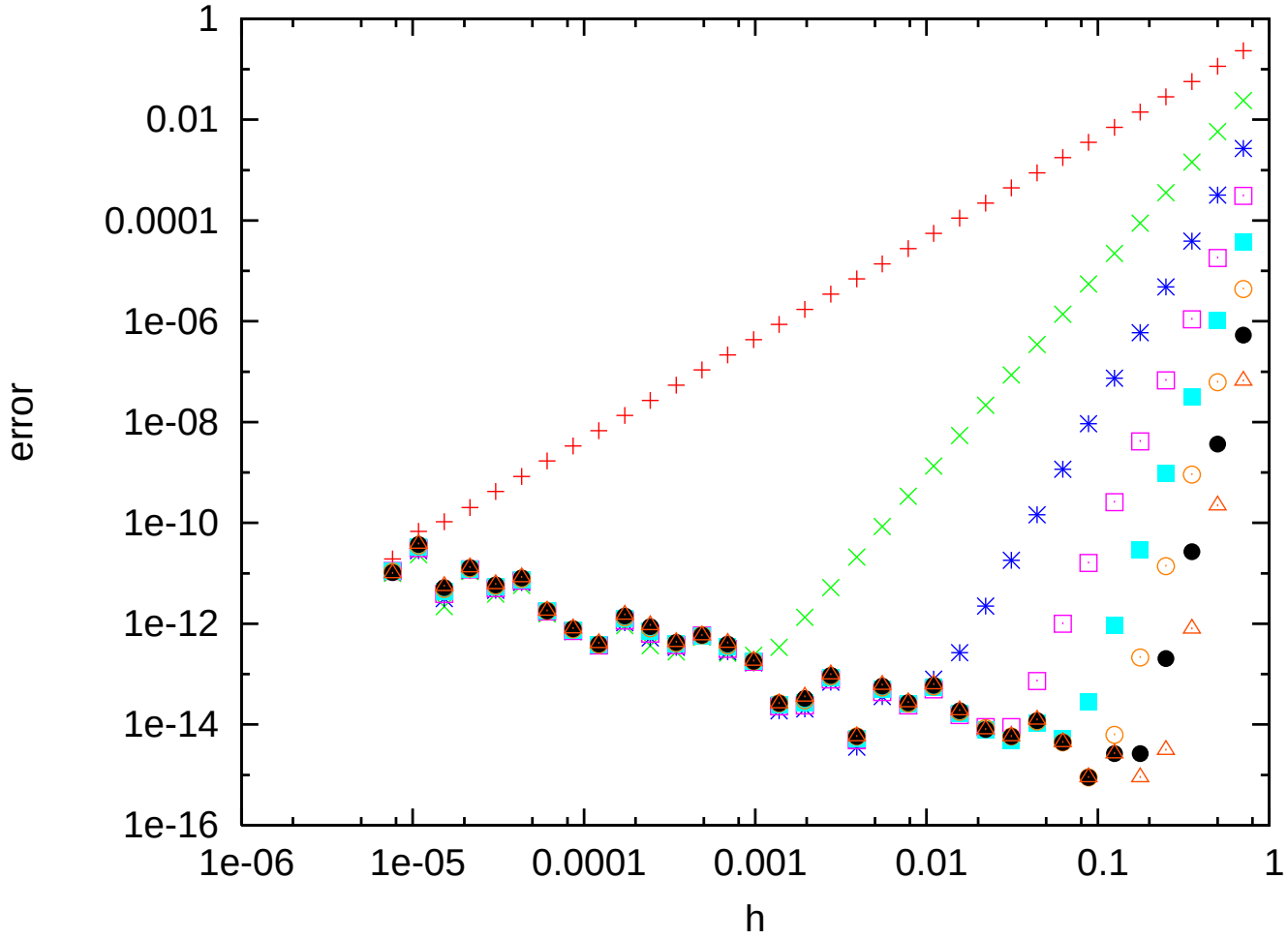

1st der
2nd der
3rd der
4th der
error
h
1
0.01
0.0001
1e-06
1e-08
1e-10
1e-12
1e-14
1e-06
1e-05
0.0001
0.001
0.01
0.1
1

Table 1: The coefficients c_i for calculating first and second derivatives for different values of m according to Eq. 19. The coefficients for negative i follow from the symmetry $c_{-i} = -c_i$ for the first and $c_{-i} = c_i$ for the second derivative.

m	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
1	0	1/2							
2	0	2/3	-1/12						
3	0	3/4	-3/20	1/60					
4	0	4/5	-1/5	4/105	-1/280				
5	0	5/6	-5/21	5/84	-5/504	1/1260			
6	0	6/7	-15/56	5/63	-1/56	1/385	-1/5544		
7	0	7/8	-7/24	7/72	-7/264	7/1320	-7/10296	1/24024	
8	0	8/9	-14/45	56/495	-7/198	56/6435	-2/1287	8/45045	-1/102960
1	-2	1							
2	-5/2	4/3	-1/12						
3	-49/18	3/2	-3/20	1/90					
4	-205/72	8/5	-1/5	8/315	-1/560				
5	-5269/1800	5/3	-5/21	5/126	-5/1008	1/3150			
6	-5369/1800	12/7	-15/56	10/189	-1/112	2/1925	-1/16632		
7	-266681/88200	7/4	-7/24	7/108	-7/528	7/3300	-7/30888	1/84084	
8	-1077749/352800	16/9	-14/45	112/1485	-7/396	112/32175	-2/3861	16/315315	-1/411840

This figure shows the error for all the 8 sets of differentiation coefficients for the first derivative in the table on the previous page. The lowest order formula (red plus sign) has the smallest slope, whereas the highest order formulas with m=7 (black points) m=8 (red triangels) have the largest slope and allow us to reach 1.e-15, which is close to machine precision.



Exercise [2pt]: *Prove the statement about the symmetry properties of the differentiation coefficients, i.e show that $c_{-i} = (-1)^l c_i$ for the lth derivative.*
Hint: Without restriction, we can consider the origin as the point x_0 where the derivative has to be calculated. Use the fact that

$$\frac{\partial^l}{\partial x^l} f(-x)\bigg|_{x=0} = (-1)^l \left.\frac{\partial^l}{\partial x^l} f(x)\right|_{x=0} \tag{21}$$

Consider f to be a polynomial of low enough degree, such that it can exactly be differentiated with the coefficients c_i.

Partial derivatives of functions depending on several variables

The coefficients of finite difference formulas for mixed partial derivatives can easily be obtained from the one-dimensional coefficients. Let us consider as an example the mixed derivative $\frac{\partial^2 f}{\partial x \partial y}$ of a function $f(x,y)$. We obtain

$$\begin{aligned}
\frac{\partial^2 f}{\partial x \partial y} &= \frac{\partial}{\partial y}\frac{\partial f}{\partial x} \\
&\approx \frac{\partial}{\partial y}\sum_i f(x+ih,y)\frac{c_i}{h} \\
&= \sum_i \frac{c_i}{h}\frac{\partial}{\partial y} f(x+ih,y) \\
&\approx \sum_i \frac{c_i}{h}\sum_j f(x+ih,y+jh)\frac{c_j}{h} \\
&= \sum_i\sum_j f(x+ih,y+jh)\frac{c_i c_j}{h^2}
\end{aligned}$$

where the coefficients c_i belong to some set of Table 1.

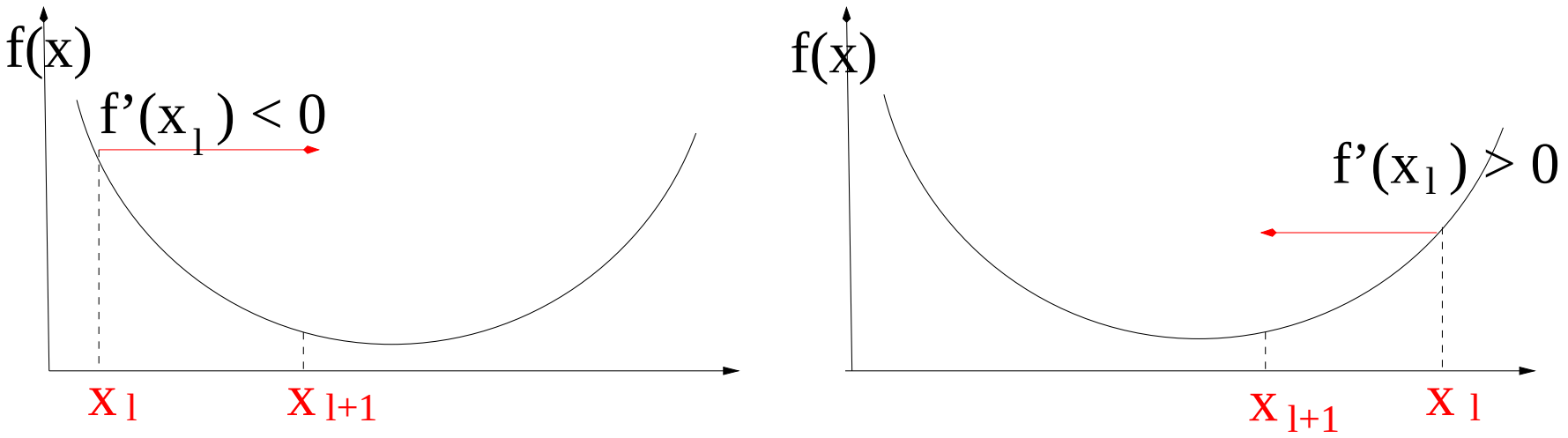
5 Minimization Methods

5.1 Minimizing a continuous 1-dim function

Minimizing a smooth function is considerably easier than minimizing a non-smooth or even discontinuous function. If the first derivative exists, its sign tells us whether we have to move to the right or to the left to come closer to the minimum. The so-called steepest descent iteration

$$x_{l+1} = x_l - \alpha f'(x_l) \tag{22}$$

will therefore converge to the minimum $f(x_M)$ of the function f if the step size α is sufficiently small. If α is too large the iteration will diverge.



l

Next, we will discuss the case where the second derivative exists as well. Using in a combined way the information on the first and second derivative gives the most efficient minimization algorithms. The information about even higher derivatives is typically not used since this would be too complicated. Consequently we can assume in our discussion of minimization algorithms that we have to minimize a quadratic function. Then we can do a Taylor expansion of the function f and its derivative around an arbitrary point $\tilde{x}$

$$f(x) = f(\tilde{x}) + (x-\tilde{x})\, f'(\tilde{x}) + \frac{1}{2}(x-\tilde{x})^2\, f'' \tag{23}$$

$$f'(x) = f'(\tilde{x}) + (x-\tilde{x})\, f'' \tag{24}$$

The stationary point $x = x_M$ where the derivative vanishes can easily be obtained by solving Eq. 24.

$$x_M = \tilde{x} - f'(\tilde{x})/f'' \tag{25}$$

We assume it is a minimum (i.e. $f'' > 0$) and not a maximum. Eq. 25 gives rise to the Newton iteration

$$x_{l+1} = x_l - f'(x_l)/f'' \tag{26}$$

The iteration of Eq. 26 will obviously converge in a single step for a quadratic function, but several iterations are needed for a general function. In the case of a quadratic function we did not have to worry where to evaluate the second derivative since it was a constant.

This is of course not any more true for a general function. As a matter of fact we see that for the one-dimensional case we are discussing, Eq. 22 and Eq. 26 are identical if we put $\alpha = 1/f''$. Therefore one best adopts for the one-dimensional case the point of view that we just do steepest descent iterations where α is of the order of $1/f''$, but small enough to ensure convergence. In this case we do not have to answer the question where to evaluate f''.

Exercise [1pt]: *Minimize the function* $-\exp(-x^2)$ *numerically using Eq. 22. For which starting values does the iteration of Eq. 26 diverge if we evaluate* f'' *at* x_l

5.2 Minimizing continuous many-dimensional functions

The basic concepts of the 1-dimensional case can be carried over into the many dimensional case. The first derivative has just to be replaced by the gradient which by definition points in the direction of the strongest increase of the function. The opposite direction consequently gives the strongest decrease of the function. Hence, the steepest descent iteration becomes

$$\vec{x}_{l+1} = \vec{x}_l - \alpha\, \vec{g}(\vec{x}_l) \tag{27}$$

where $\vec{g}(\vec{x}) = \nabla f(\vec{x})$ is the gradient of the function f. As in the 1-dim case this will converge to a minimum if α is sufficiently small.
For a function where the second derivatives exist we can again do a Taylor expansion

$$f(\vec{x}) = f(\tilde{\vec{x}}) + (\vec{x}-\tilde{\vec{x}})^T\, \vec{g}(\tilde{\vec{x}}) + \frac{1}{2}\,(\vec{x}-\tilde{\vec{x}})^T\, A\,(\vec{x}-\tilde{\vec{x}}) \tag{28}$$

$$\vec{g}(\vec{x}) = \vec{g}(\tilde{\vec{x}}) + A(\vec{x}-\tilde{\vec{x}}) \tag{29}$$

where A is the Hessian matrix

$$A(i,j) = \frac{\partial}{\partial x(i)}\frac{\partial}{\partial x(j)} f(\vec{x}) \tag{30}$$

For a quadratic form the Hessian matrix would not depend on the evaluation point, for a general function it of course does and the problem where to evaluate it will be postponed.

Requiring $\vec{g}(\vec{x}) = 0$ in Eq. 29 and solving for $\vec{x}$ while putting $\vec{x}_l = \tilde{\vec{x}}$ and $\vec{x}_{l+1} = \vec{x}$ leads to the Newton iteration

$$\vec{x}_{l+1} = \vec{x}_l - A^{-1}\,\vec{g}(\vec{x}_l) = \vec{x}_l - \vec{p}_l \tag{31}$$

Note that we have to solve in each iteration of the Newton method a linear system of equations for the preconditioned gradient vector p

$$A\vec{p} = \vec{g} \tag{32}$$

There are several basic problems with the Newton iteration:

- As mentioned before, it is not clear where to evaluate it for a non-quadratic form
- Realistic functions are not quadratic forms and so the theory is anyway only an approximation.
- The calculation of the exact Hessian matrix is numerically too expensive for complicated high-dimensional functions
- Solving Equation. 32 is too expensive for high-dimensional functions.

Let us therefore define a slightly more general iteration that we will call preconditioned steepest descent iteration

$$\vec{x}_{l+1} = \vec{x}_l - P\,\vec{g}(\vec{x}_l) \tag{33}$$

where P is a still unspecified preconditioning matrix. Evidently we get the steepest descent iteration of Eq. 27 if we put $P = \alpha I$ and we get the Newton iteration of Eq. 31 if we put $P = A^{-1}$

5.3 Convergence of the steepest descent iteration

For the convergence analysis we will again assume that we are already sufficiently close to the minimum, so that the function is a quadratic form. Because by definition the gradient vanishes at x_M the Taylor expansion of Eq. 28 becomes

$$f(\vec{x}) - f(\vec{x}_M) = \frac{1}{2}\,(\vec{x}-\vec{x}_M)^T\,A\,(\vec{x}-\vec{x}_M) \tag{34}$$

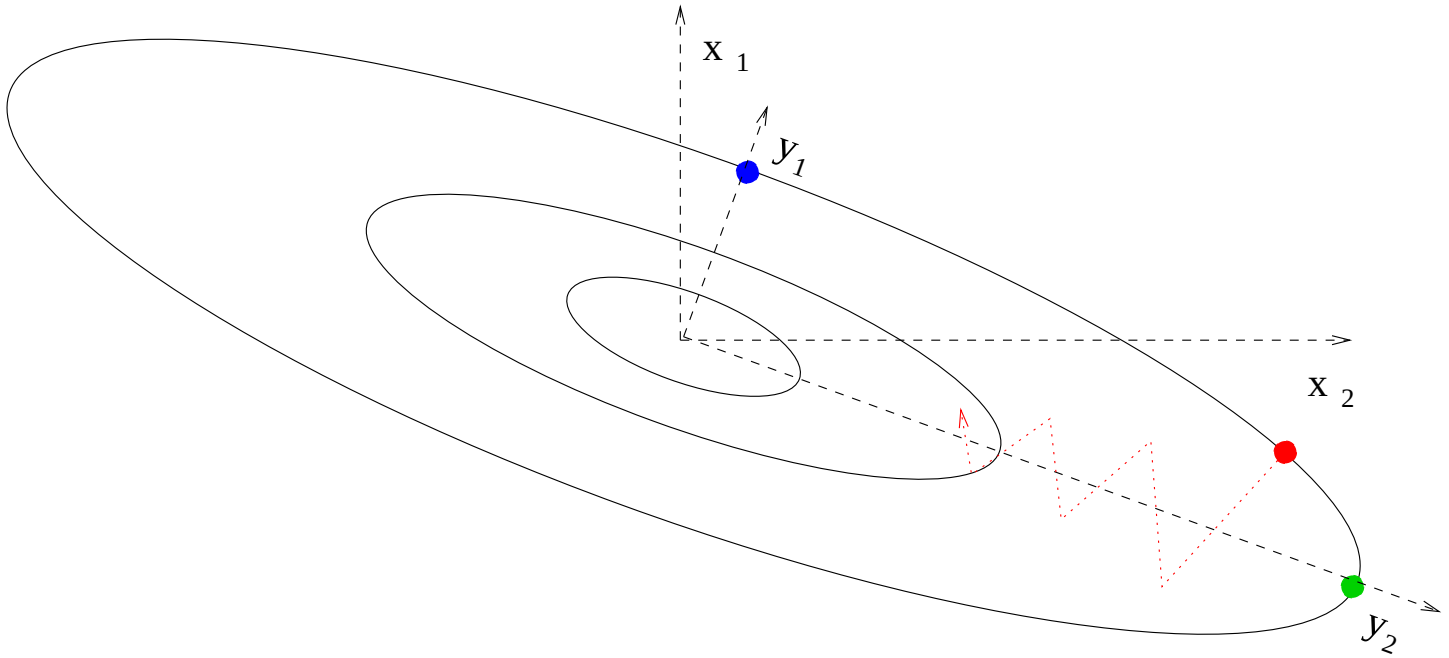
By shifting the origin (such that $\vec{x}_M = 0$) and the function (such that $f(\vec{x}_M) = 0$) we can without any restriction consider the simpler case

$$f(\vec{x}) = \frac{1}{2}\,\vec{x}^T\,A\,\vec{x} \tag{35}$$

Since the Hessian A is a positive definite symmetric matrix, we can go into an coordinate system $\vec{y}$, that is obtained by applying a unitary transformation U on the original coordinate system $\vec{x}$, where A becomes a positive real diagonal matrix $D = U^T AU$ with diagonal elements (eigenvalues) $d(k)$.

$$f(\vec{y}) = \frac{1}{2}\sum_k d(k)\,y(k)^2 \qquad ; \qquad g(k) = d(k)\,y(k) \tag{36}$$

Things are illustrated in the figure below. The ellipsoids represent the equipotential lines of the function f. The axis of the y coordinate system coincide with the principal axis of the ellipsoids.



Let us now assume that at a certain stage of a steepest descent iteration the current point $\vec{x}_l$ coincides with the blue dot. Since in this case the gradient points exactly in the direction of the minimum, we can find the minimum of this one-dimensional subproblem with a single steepest descent step if we chose $\alpha = 1/d(1)$. If we are at the green dot the same arguments apply except that now $\alpha = 1/d(2)$. In general our current iterations points are not located on any principle axis. The gradient of an arbitrary point such as the red dot has components of both principal axis. In order to guarantee convergence we have to be conservative and to choose $\alpha = 1/max[d(1),d(2)]$. Since the components of the gradient

that correspond to principal axes with small eigenvalues will be damped too strongly, a steepest descent iteration in more than two dimensions is approaching the minimum very slowly by a large number of zigzag moves.
The generalization to more than 2 dimensions is obvious. The α of a steepest descent iteration has to be taken to be the reciprocal of the largest eigenvalue of the Hessian. Let us now examine the convergence rate for the multi-dimensional case in a more mathematical way. Since the steepest descent iteration is invariant under unitary transformations of the coordinate system we can without restriction consider a diagonal Hessian.
Exercise [1pt]: *Prove the above statement*
The steepest descent iteration then becomes

$$y_{l+1}(k) = y_l(k) - \alpha d(k) y_l(k) \tag{37}$$

Hence

$$y_{l+1}(k) = y_1(k)(1 - \alpha d(k))^l \tag{38}$$

where $\vec{y}_1$ is the starting vector for the iteration. Convergence can only be obtained if $|1 - \alpha d(k)| < 1$ for all k. Hence α can be at most twice of the reciprocal of the largest eigenvalue. So let us put

$$\alpha = t/d_{max} \tag{39}$$

where t is in between 0 and 2. For $t = 1$, the component k that will converge most slowly is the one associated to the smallest eigenvalue. Requiring this component to be equal to

a certain precision p gives

$$(1 - t\frac{d_{min}}{d_{max}})^l = p \tag{40}$$

The number of iterations l necessary to obtain this precision p is then given by

$$l = \ln(p)/\ln(1 - t\frac{d_{min}}{d_{max}}) \tag{41}$$

If $\frac{d_{min}}{d_{max}}$ is small, this is asymptotically equal to

$$l = -\ln(p)\frac{d_{max}}{t\, d_{min}} = -\frac{\ln(p)}{t}\kappa \tag{42}$$

The ratio between the largest and the smallest eigenvalue of the Hessian matrix is called the condition number $\kappa = \frac{d_{max}}{d_{min}}$. We have thus the result that the number of iterations is proportional to the condition number κ in the steepest descent method. This is a big problem. As we will see the conditioning number is typically growing rapidly with respect to the size of the physical system represented by the matrix. Hence the number of iterations is growing substantially as well.

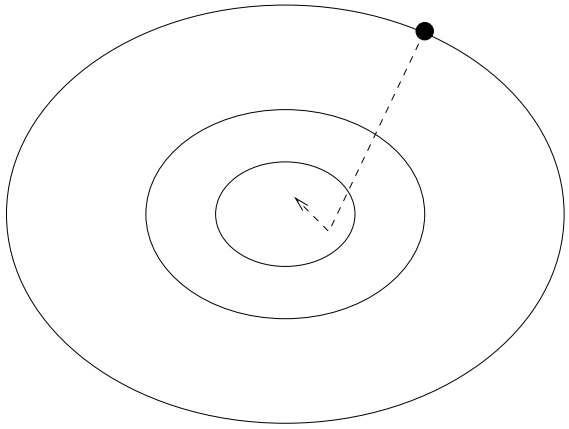
5.4 Convergence of the preconditioned steepest descent iteration

The convergence analysis of the preconditioned steepest descent iteration of Eq. 33 is analogous to the one for the simple steepest descent iteration. The only difference is that we perform the analysis in a coordinate system that diagonalizes PA instead of A. The number of iterations is consequently given by the same formula

$$l = -\ln(p)\frac{d_{max}}{t\, d_{min}} \tag{43}$$

the only difference being that d_{max} and d_{min} are now the largest and smallest eigenvalues of PA. If the conditioning number of PA is smaller than of A, the number of iterations of the preconditioned steepest descent method will be reduced compared to the simple steepest descent method. A good preconditioning matrix is a compromise between 2 requirements. On the one hand it should give a small condition number, on the other hand it should be easy to calculate and to apply to the gradient. A frequent choice for P is a diagonal or sparse matrix.

Topology of preconditioned problem:



5.5 Steepest descent with line minimization

The prescription for a steepest descent iteration with line minimization for a function f is formally identical to an ordinary steepest descent minimization

$$\vec{x}_{l+1} = \vec{x}_l - \alpha \vec{g} \tag{44}$$

The difference is that α is not fixed, but optimized such that

$$\frac{\partial}{\partial \alpha} f(\vec{x} + \alpha \vec{g}) = 0 \tag{45}$$

The line minimization ensures that the function will decrease at each iteration point. This does however not imply that one comes as close as possible to the minimum. As a matter of fact it turns out that with an optimal value of t (Eq.39) the convergence is as fast as with line minimization. In addition one iteration is much cheaper without the line minimization. The conclusion is that one should avoid line minimizations unless one can not at all estimate the largest eigenvalue of the Hessian matrix. If this estimation is possible, steepest descent with some feedback is a recommendable strategy.

5.6 Steepest descent with energy feedback

A simple and powerful modification of the simple steepest descent method is the steepest descent with energy feedback. Assuming that the functional value represents the energy, we decrease the step size α if the energy rises in an iteration, otherwise we increase it. Since we know that in the case of an energy increase the parameter t (Eq.39) is roughly twice as large as would be optimal for the elimination of the stiff components, associated to large eigenvalues of the Hessian, we decrease α by a factor of $1/2$. If the energy goes down, as it should, we slightly increase α (e.g. by a factor of 1.05) to speed up the convergence.

5.7 Steepest descent with gradient feedback

In practice one finds that the following feedback gives faster convergence than the energy feedback. At each iteration one calculates the angle between the current gradient vector and the gradient vector from the previous iteration. If the angle is larger than let's say 60 degrees, the step size α is decreased by a factor of $1/2$, otherwise it is increased by 1.05. In this way one avoids that consecutive gradients are pointing in opposite directions, which is obviously not desirable for a fast convergence.

Exercise [4pt]: *An Lennard-Jones cluster is some artificial system where the 'atoms' interact through the Lennard-Jones potential. The potential energy E of such a cluster is*

$$E = \sum_{i=1,N_{at}} \sum_{j=1,i-1} 4\left(\frac{1}{|\mathbf{R}_i - \mathbf{R}_j|^{12}} - \frac{1}{|\mathbf{R}_i - \mathbf{R}_j|^{6}}\right) \tag{46}$$

Equilibrium geometries are given by minima of the potential energy. Minimize the energy to find an equilibrium geometry using the steepest descent method with energy and gradient feedback. α *will turn out to be of the order of 1.d-3. Stop the minimization if the gradient norm is less than 1.e-5. Which method is more efficient? The subroutine LJ_calc.py (contained in the exercise material file on http://comphys.unibas.ch/teaching.htm) can be used to calculate the energy and forces (= negative gradient) of a Lennard-Jones cluster and the file LJ38.xyz contains the coordinates of the lowest energy cluster containing 38 atoms. Displace the atoms slightly from the geometry given in this file and use either one of the above mentioned minimization methods. It can be assumed that the original configuration is regained if the energy after minimization agrees with the initial energy. If the atoms are strongly displaced one might however fall into another local minimum. The format of the file LJ38.xyz is such that it can be viewed with standard visualization software such as V_Sim provided at https://gitlab.com/l_sim/v_sim or ovito. The first line gives the number of atoms, the second is empty. The remaining lines give the x,y and z coordinates of each 'atom' followed by the atom type.*

5.8 The conjugate gradient (CG) method

Eq. 29 tells us that finding the minimum of a quadratic form is equivalent to solving a linear system. The (preconditioned) steepest descent method can therefore be considered as the simplest iterative method for solving a linear system of equations. There are however more powerful methods. One of the most popular method is the conjugate gradient method. It is based on a bi-orthogonal sequence $\vec{g}_i, \vec{h}_i$

$$\vec{g}_i^T \vec{g}_j = \sum_k g_i(k) g_j(k) = \delta_{i,j} \tag{47}$$

$$\vec{h}_i^T A \vec{h}_j = \sum_{k,l} h_i(k) A(k,l) h_j(l) = \delta_{i,j} \tag{48}$$

Solving the system of equations

$$A\vec{x} = \vec{y} \tag{49}$$

which arises from zeroing the gradient of $\frac{1}{2}\vec{x}^T A\vec{x} - \vec{y}^T\vec{x}$, is easy in the space spanned by the h_i's. Writing $\vec{x} = \sum_j c_j \vec{h}_j$ one obtains

$$\sum_j c_j A\vec{h}_j = \vec{y} \tag{50}$$

Multiplying from the left by $\vec{h}_i$ one obtains

$$\sum_j c_j \vec{h}_i^T A \vec{h}_j = c_i = \vec{h}_i^T \vec{y} \tag{51}$$

In implementations of the conjugate gradient method one is simultaneously generating the bi-orthogonal sequence and then updating the approximate solution $\vec{x}$. For a m dimensional matrix there are at most m non-zero vectors $\vec{h}_i$. Therefore the exact solution has to be found after at most m iterations. This property is sometimes stressed in mathematics books. It is however not the property that makes conjugate gradient so useful in practice because

- It only holds for linear systems, whereas in practice the conjugate gradient method is usually applied for minimization problems where the function is not a quadratic form.
- Even for linear systems, it is violated in finite precision arithmetic because of rounding errors
- m iterations are far too expensive for large matrices

What makes the conjugate gradient method superior to the steepest descent method is its faster convergence rate. It can be shown that the number of iterations l is

$$l \propto \sqrt{\kappa} \tag{52}$$

instead of Eq. 42. For badly conditioned systems a lot can thus be gained by using the conjugate gradient instead of the steepest descent method, for well conditioned (or pre-conditioned) systems not much can be gained.

Generation of bi-orthogonal sequence

Here is the conjugate gradient formulation for the minimization of an arbitrary function f. Given an initial input guess x_0 we calculate $\vec{g}_0 = \nabla f(\vec{x}_0)$ and put $\vec{h}_0 = \vec{g}_0$. Consecutive steps l:

- Determine by a line minimization the α_l that gives the lowest energy. That is usually done by finding the point where the derivative vanishes.

$$\frac{\partial}{\partial \alpha_l} f(\vec{x}_l + \alpha_l \vec{h}_l) = 0 \tag{53}$$

- Update the solution

$$\vec{x}_{l+1} = \vec{x}_l + \alpha_l \vec{h}_l \tag{54}$$

- Calculate new gradient

$$\vec{g}_{l+1} = \nabla f(\vec{x}_{l+1}) \tag{55}$$

- Calculate new $\vec{h}$

$$\vec{h}_{l+1} = \vec{g}_{l+1} + \gamma \vec{h}_l \tag{56}$$

where (Polak Ribiere)

$$\gamma = \frac{(\vec{g}_{l+1} - \vec{g}_l)^T \vec{g}_{l+1}}{\vec{g}_l^T \vec{g}_l} \tag{57}$$

For the case of a quadratic function one could simplify the Polak Ribiere formula by using the orthogonality of the vectors $\vec{g}_l$. However it turns out that for a general function where the orthogonality of the vectors $\vec{g}_l$ is not any more satisfied, the above Polak Ribiere formula is more stable.

5.9 The Newton method revisited

Even though the Newton method is not widely used for finding minima we will discuss in more detail a variant that is also applicable if the Hessian has zero or very small eigenvalues. The same approach can be used in similar methods such as the preconditioned steepest descent iteration. So let us assume that we know the Hessian matrix $A(\mathbf{x})$ at a point $\mathbf{x}$ together with the gradient $\mathbf{g}(\mathbf{x})$. We can diagonalize this Hessian matrix to obtain its eigenvalues λ_i and eigenvectors $\mathbf{v}_i$ using the routine from
https://numpy.org/doc/stable/reference/generated/numpy.linalg.eigh.html#numpy.linalg.eigh
The eigenvalues and the corresponding eigenvectors are in increasing order. For the moment we assume that all the eigenvalues are larger than zero. The case where some eigenvalues are zero will be discussed afterwards. We have now to transform the gradient in the new coordinate system spanned by the orthogonal set of eigenvectors. For this we have to calculate the coefficients g_i

$$g_i = \langle \mathbf{g} | \mathbf{v}_i \rangle = \sum_j g(j)\, v_i(j) \tag{58}$$

where $g(j)$ is the j-th component of $\mathbf{g}$ and $v_i(j)$ the j-th component of $\mathbf{v}_i$. g_i is the i-th component of the gradient vector in the new coordinate system. Since we are now in the principal axis coordinate system we can multiply each component by the ideal stepsize which is the inverse curvature. Since the curvature is given by the eigenvalues we have

$$\hat{g}_i = g_i/\lambda_i \tag{59}$$

Then we have to go back in our original coordinate system to get the preconditioned gradient $\hat{\mathbf{g}}$. The vector $\hat{\mathbf{g}}$ is what one would obtain by applying A^{-1} to $\mathbf{g}$:

$$\hat{\mathbf{g}} = \sum_i \hat{g}_i \, \mathbf{v}_i \tag{60}$$

Finally we update the atomic positions according to

$$\mathbf{R} \leftarrow \mathbf{R} - \hat{\mathbf{g}} \tag{61}$$

So far nothing has been gained in comparison with a standard implementation of the Newton method. However, these methods are frequently applied to geometry optimizations of molecules and solids where one has to find the atomic coordinates that minimize the energy. Since the energy is invariant under translations, the Hessian matrix has three eigenvectors with zero eigenvalues and is thus singular. Hence A^{-1} does not exist. Numerically the eigenvalues are not strictly zero but very small. These nearly zero eigenvalues can lead to problems in Eq. 59. Unless the system (which can be a molecule or periodic solid) is in the field of an external potential the overall translational force (negative gradient) has to be zero and so the three components $g(i)$ that correspond to the translations have to be zero. Analytically we have thus three cases in Eq. 59 where zero is divided by zero, in numerical work we will just divide two very small numbers. Since these numbers are essentially rounding noise the result would be completely wrong. For a molecule at equilibrium it can be shown that there are three more zero eigenvalues that correspond to rotations. If the molecule is close to a local minimum the three eigenvalues are not

exactly zero but very small which will lead as well to numerical problems. To avoid such problems we have to modify Eq. 59 to

$$\hat{g}_i = \frac{g_i}{\lambda_i + \gamma} \tag{62}$$

and this value of $\hat{g}(i)$ has then to be used in Eq. 60. γ has to be chosen such that the denominator is always positive and not too small. In the case of a molecule or cluster, a good empirical choice for γ is to set it equal to half the value of the first non-zero eigenvalue. In the case of a periodic solid this is the 4-th eigenvalue and in the case of a molecule the 7-th eigenvalue.

The practical implementation of the preconditioned steepest descent iteration is very similar to the Newton method. The main difference is that in the preconditioned steepest descent method one uses an approximate Hessian instead of the exact Hessian. Approximate Hessians have in general also zero eigenvalues which have to be treated in a similar manner as in the Newton method.

Exercise [3pt]: *Use the Newton method to find equilibrium geometries of the 38 atom LJ cluster of the previous exercise. Show that the convergence rate is much faster than with the steepest descent method. A routine hessian_LJ.py that calculates the Hessian matrix is available on http:/comphys.unibas.ch/teaching.htm. For the diagonalization use the routine available at*
https://numpy.org/doc/stable/reference/generated/numpy.linalg.eigh.html#numpy.linalg.eigh

5.10 Quasi Newton (QN) methods

The basic principle of quasi Newton methods is to build up information about the Hessian matrix from the gradient evaluations during the minimization iterations. This is possible because the Hessian matrix (Eq. 30) can be obtained by finite differences from the forces or gradients

$$A(\mathbf{R}) = \begin{pmatrix} . & . & \cdots & . \\ \frac{\mathbf{g}(\mathbf{R}+h\mathbf{e}_1)-\mathbf{g}(\mathbf{R})}{h} & \frac{\mathbf{g}(\mathbf{R}+h\mathbf{e}_2)-\mathbf{g}(\mathbf{R})}{h} & \cdots & \frac{\mathbf{g}(\mathbf{R}+h\mathbf{e}_n)-\mathbf{g}(\mathbf{R})}{h} \\ . & . & \cdots & . \end{pmatrix} \tag{63}$$

Denoting by $\mathbf{G}_i$ the finite difference vector between the two gradients $\mathbf{G}_i = \mathbf{g}(\mathbf{R}+h\mathbf{e}_i) - \mathbf{g}(\mathbf{R})$ we see that the matrix element $A_{i,j}$ is given by the scalar product $\frac{1}{h}\langle \mathbf{G}_i | \mathbf{e}_j \rangle$, where $\mathbf{e}_i$ is an orthonormal set of vectors. Now very similar quantities are a by-product of any gradient based minimization. If the system is moved in a minimization step from $\mathbf{R}$ to $\mathbf{R}+\mathbf{d}$ and the forces are evaluated at both points we can calculate the approximate curvature along the direction $\mathbf{d}_i$.

$$\frac{1}{\langle \mathbf{d}|\mathbf{d}\rangle} \frac{\partial^2}{\partial \alpha^2} E(\mathbf{R}+\alpha\mathbf{d})|_{\alpha=0} = \frac{1}{\langle \mathbf{d}|\mathbf{d}\rangle} \frac{\partial}{\partial \alpha} \langle \mathbf{g}(\mathbf{R}+\alpha\mathbf{d})|\mathbf{d}\rangle|_{\alpha=0} \approx \frac{\langle \mathbf{G}(\mathbf{d})|\mathbf{d}\rangle}{\langle \mathbf{d}|\mathbf{d}\rangle} \tag{64}$$

where we have again denoted by $\mathbf{G}$ the difference between two gradient vectors: $\mathbf{G}(\mathbf{d}) = \mathbf{g}(\mathbf{R}+\mathbf{d}) - \mathbf{g}(\mathbf{R})$. If we assume our function E to be a perfect quadratic form then $\mathbf{G}(\mathbf{d})$

is equal to $A\mathbf{d}$. The subspace Hessian matrix B with respect to an non-orthogonal set of vectors $\mathbf{d}_i$ is then given by

$$B_{i,j} = \langle \mathbf{d}_i | A | \mathbf{d}_j \rangle = \langle \mathbf{G}(\mathbf{d}_i) | \mathbf{d}_j \rangle \tag{65}$$

The eigenvalues of the Hessian A can consequently be obtained by solving the generalized eigenvalue problem for the matrices B and S

$$B\mathbf{v}_l = \lambda_l S\mathbf{v}_l \tag{66}$$

where S is the overlap matrix $S_{i,j} = \langle \mathbf{d}_i | \mathbf{d}_j \rangle$. This approach clearly fails if the vectors $\mathbf{d}_i$ are linearly dependent. Numerical problems actually already arise if the vectors $\mathbf{d}_i$ are nearly linearly dependent, i.e if the overlap matrix is nearly singular, which can be detected by very small eigenvalues of the overlap matrix. Standard Quasi Newton methods such as the popular BFGS variant, named after their inventors Broydens, Fletcher, Goldfarb and Shanno, can therefore fail in such cases. Linearly dependent vectors can be encountered if the minimization is started far away from the local minimum. If the minimiztion starts close to the local minimum where the function can be well approximated by a quadratic form such problems do generally not arise and rapid convergence is generally found. The most popular implementation of the BFGS method is the Limited memory LBFGS variant where second derivative information is exploited only from the few last iterations. Typically a history length of about 10 is choosen. Even if the dimension of the entire Hessian

matrix is in general much larger than this history length, it turns out that the convergence speed is not improved by a longer history. On the contrary, a too long history can lead to numerical instabilities.

Exercise [4pt]: *Extracting curvature information from a set of gradient vectors:*
Take the local minimum $\mathbf{R}_0$ *of the 38 atom LJ cluster of the previous exercise and calculate the Hessian matrix for this configuration using the subroutine hessian_LJ.py (available in the tar file at http:/comphys.unibas.ch/teaching.htm). Find the eigenvalues of this matrix by using the routine from*
https://numpy.org/doc/stable/reference/generated/numpy.linalg.eigh.html#numpy.linalg.eigh
Six of these eigenvalues should be zero corresponding to the three translations and the three rotations that leave the energy invariant. The entire set of eigenvalues will serve as reference values for the following part of this exercise.
Next, perturb this minimum by a random displacement

$$\mathbf{r}_0 = \mathbf{R}_0 + a\chi$$

where χ *is a random vector and the amplitude a should be about 1.e-2. Generate then a sequence of configurations* $\mathbf{r}_i$, $i = 1, \ldots, n$, *by performing a steepest descent geometry optimization with an energy or gradient feedback. Consider then the sequence of displacement vectors* $\mathbf{d}_i$

$$\mathbf{d}_i = \mathbf{r}_i - \mathbf{r}_{i-1}$$

Use at each configuration $\mathbf{r}_i$ *the corresponding force* $\mathbf{f}_i$ *to obtain the gradient difference* $\mathbf{G}_i$

$$\mathbf{G}_i = -(\mathbf{f}_i - \mathbf{f}_{i-1})$$

Calculate then the overlap matrix $S_{i,j} = \langle \mathbf{d}_i | \mathbf{d}_j \rangle$ *and the Hessian matrix in this basis,* $B_{i,j} = \langle \mathbf{G}_i | \mathbf{d}_j \rangle$ *for several values of n. For a purely quadratic form the Hessian matrix B would be symmetric, i.e* $B_{i,j} = B_{j,i}$*. Since this is not the case there will be small deviations from symmetry. Check that these deviations get smaller if the initial displacement amplitude is reduced. Next calculate the eigenvalues of the generalized eigenvalue problem of Eq. 66 using /newline https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.eig.html. Verify that you get for small values of n already with reasonable precision the large eigenvalues of the full Hession matrix and that all the eigenvalues lie within the spectrum of the full Hessian matrix. Verify that once n gets larger numerical instabilities arise which prevent obtaining all the* 3×38 *eigenvalues of the full Hessian matrix correctly.*

5.11 The DIIS (Direct Inversion in Iterative Subspace) minimization method

Be $\vec{c}_\infty$ the exact solution of a quadratic minimization problem and $\vec{c}_i$ (i=1, ..,m) a set of m approximate solution vectors. Their error vectors are defined by

$$\vec{e}_m = \vec{c}_m - \vec{c}_\infty \tag{67}$$

We form a new vector $\tilde{\vec{c}}_m$

$$\tilde{\vec{c}}_m = \sum_{i=1}^{m} d_i \vec{c}_i \tag{68}$$

If $\tilde{\vec{c}}_m$ was the exact solution, it would fulfill

$$\sum_{i=1}^{m} d_i \vec{c}_i = \vec{c}_\infty \tag{69}$$

$$\sum_{i=1}^{m} d_i (\vec{c}_\infty + \vec{e}_i) = \vec{c}_\infty \tag{70}$$

$$\sum_{i=1}^{m} d_i \vec{c}_\infty + \sum_{i=1}^{m} d_i \vec{e}_i = \vec{c}_\infty \tag{71}$$

This is satisfied if

$$\sum_{i=1}^{m} d_i = 1 \qquad ; \qquad \sum_{i=1}^{m} d_i \vec{e}_i = 0 \tag{72}$$

Last condition can only be fulfilled approximately, leading to the minimization problem

$$\min \left[\langle \sum_{i=1}^{m} d_i e_i | \sum_{i=1}^{m} d_i e_i \rangle \right] \tag{73}$$

under the constraint $\sum_{i=1}^{m} d_i = 1$. This leads to the system of equations

$$\begin{pmatrix} \langle e_1|e_1\rangle & \langle e_1|e_2\rangle & \ldots & \langle e_1|e_m\rangle & 1 \\ \langle e_2|e_1\rangle & \langle e_2|e_2\rangle & \ldots & \langle e_2|e_m\rangle & 1 \\ \ldots & \ldots & \ldots & \ldots & . \\ \langle e_m|e_1\rangle & \langle e_m|e_2\rangle & \ldots & \langle e_m|e_m\rangle & 1 \\ 1 & 1 & \ldots & 1 & 0 \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \\ . \\ d_m \\ d_{m+1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{74}$$

In practice the error vectors are approximated by $\vec{e}_i = P\vec{g}_i$
The new vector is then given by

$$\tilde{\vec{g}}_m = \nabla f(\tilde{\vec{c}}_m) \tag{75}$$

$$\vec{c}_{m+1} = \tilde{\vec{c}}_m - P\tilde{\vec{g}}_m \tag{76}$$

Variable preconditioning DIIS implementation

There are possibly two preconditioning matrices $P = P_m$ and $\tilde{P} = \tilde{P}_m$ that depend on the iteration step m.
At the m-th step do

- $$\vec{g}_m = \nabla f(\vec{c}_m) \tag{77}$$
- $$\vec{e}_i = P_m \vec{g}_i \qquad , \qquad i = 1, \ldots, m \tag{78}$$
- Solve Eq. 74 to get $\tilde{\vec{c}}_m = \sum_{i=1}^{m} d_i \vec{c}_i$. Under the assumption that we are in a quadratic region, the coefficients d_i allow us then also to calculate

$$\tilde{\vec{g}}_m = \nabla f(\tilde{\vec{c}}_m) = \nabla f(\sum_{i=1}^{m} d_i \vec{c}_i) = \sum_{i=1}^{m} d_i \nabla f(\vec{c}_i) = \sum_{i=1}^{m} d_i \vec{g}_i \tag{79}$$

- $$\vec{c}_{m+1} = \tilde{\vec{c}}_m - \tilde{P}_m \tilde{\vec{g}}_m \tag{80}$$

This implementation requires to store 3 sequences of vectors: $\vec{c}_i$, $\vec{g}_i$ and $\vec{e}_i$. If the application of the preconditioning matrix P_m is cheap the $\vec{e}_i$'s can be calculated on the fly from the $\vec{g}_i$'s and one does not have to store them. The most expensive step is usually the calculation of the gradient $\vec{g}_m$, which has to be done once during each iteration.

Fixed preconditioning DIIS implementation

Only the two sequences $\vec{c}_m$ and $\vec{e}_m$ have to be stored if there is a single preconditioning matrix P that does not change during the iterations.

- $$\vec{g}_m = \nabla f(\vec{c}_m) \tag{81}$$
- $$\vec{e}_m = P\vec{g}_m \tag{82}$$
- Solve Eq. 74 to get $\tilde{\vec{c}}_m = \sum_{i=1}^{m} d_i \vec{c}_i$ Under the assumption that we are in a quadratic region, the coefficients d_i would allow us then also to calculate $\tilde{\vec{g}}_m$, even though we do not actually calculate it

$$\tilde{\vec{g}}_m = \nabla f(\tilde{\vec{c}}_m) = \nabla f(\sum_{i=1}^{m} d_i \vec{c}_i) = \sum_{i=1}^{m} d_i \nabla f(\vec{c}_i) = \sum_{i=1}^{m} d_i \vec{g}_i \tag{83}$$

- $$\vec{c}_{m+1} = \tilde{\vec{c}}_m - P\tilde{\vec{g}}_m = \sum_{i=1}^{m} d_i \vec{c}_i - P\sum_{i=1}^{m} d_i \vec{g}_i = \sum_{i=1}^{m} d_i(\vec{c}_i - P\vec{g}_i) = \sum_{i=1}^{m} d_i(\vec{c}_i - \vec{e}_i) \tag{84}$$

5.12 Comparison of Minimization Methods

Steepest descent versus methods with faster convergence rates

Both CG, DIIS and QN methods assume that we are in a quadratic region. This is frequently not the case at the start of a minimization procedure. In this case steepest descent with feedback is the method of choice, since the other methods will frequently diverge in a strongly non-quadratic region. The line minimization makes the CG however somewhat more stable than the other methods.

DIIS versus CG and QN

The DIIS method has the advantage that it is more flexible than CG and QN. Even though there is also a preconditioned version of the CG method there is no preconditioned CG method that would allow for variable preconditioning. Since the set of approximate solution vectors $\vec{c}_m$ is arbitrary, the DIIS method can be applied to a constrained minimization problem. Imposing constraints after each iteration modifies the sequence of approximate solution vectors generated during the iterations and would be illegal in the CG method. In the DIIS method imposing the constraints does not bother. The disadvantage of the DIIS method compared to the CG method is that it needs more memory to hold the set of vectors $\vec{c}_i$ and $\vec{e}_i$. If memory is limited, the sequence of vectors can be restricted to a certain

maximum value. One keeps for instance only the sequence of the last 10 iterations, even though one might need 50 iterations to converge. It turns out that with a reasonably long truncated sequence the convergence is not slowed down significantly. Another advantage of the DIIS and QN methods over the CG method is that they require only a single force evaluation per step.

PROJECT: A geometry optimization method inspired by molecular dynamics

Physical background

In contrast to the previously introduced minimization method which are on the gradient, there are also minimization methods which are based on equations of motion. In this project the FIRE (fast inertial relaxation engine) method will be introduced. Following this equation of motion will lead the system to slide down the potential energy surface by accelerating in the downhill direction and to stop uphill motions. Like a skier sliding down some mountains, not every step is exactly along the gradient direction, but the momentum is also taken into account. Therefore the information of previous steps during the geometry optimization is exploited which are contained in the current velocity. In contrast, steepest descent for example keeps only limited information of previous steps (for example a feedback in energy or gradient) when performing a downhill relaxation. FIRE is especially useful if the energy and forces can not be computed to machine precision since the velocity will not allow the relaxation path to suddenly change the direction when the noise level is reached. This is for example the case for density functional theory calculations. The implementation of FIRE is especially easy if a molecular dynamics integrator has already been implemented, since only small modifications of the velocity update have to be performed

The equation of motion for the modified molecular dynamics of FIRE is given by

$$\dot{\mathbf{v}}(t) = \frac{\mathbf{F}(t)}{m} - \gamma(t)|\mathbf{v}(t)|(\hat{\mathbf{v}}(t) - \hat{\mathbf{F}}(t)) \tag{85}$$

Tasks

1. The subroutine lenjon.f90 (available at http:/comphys.unibas.ch/teaching.htm) which will provide the energy and forces of a system consisting of Lennard-Jones particles. On the Cambridge Cluster Database (http://www-wales.ch.cam.ac.uk/CCD.html) there are many putative global minimum configurations for Lennard-Jones clusters. The file format of the files is the xyz format and can be visualized with v_sim (http://inac.cea.fr/sp2m/L_Sim/V_Sim/index.en.html) (also available from the ubuntu repositories). Write a test program to make sure that the Lennard-Jones subroutine is working correctly by comparing the computed energies with the reference energies on the Combridge Cluster Database.

2. Write a program that will perform a classical molecular dynamics simulation with the Euler algorithm. The Euler algorithm can be derived using the forward finite difference formula for both the coordinates of the particles i and their velocities.

$$\mathbf{x}_i(t) \quad = \frac{\mathbf{x}_i(t+\Delta t) - \mathbf{x}_i(t)}{\Delta t}$$

$$\mathbf{a}_i(t) \quad = \frac{\mathbf{v}_i(t+\Delta t)-\mathbf{v}_i(t)}{\Delta t}$$

Solving for the terms describing the state at $t+\Delta t$ we arrive at the iterative scheme to update the particle coordinates and their velocities.

$$\begin{aligned}\mathbf{x}_i(t+\Delta t) &= \mathbf{x}_i(t)+\mathbf{v}_i(t)\Delta t \\ \mathbf{v}_i(t+\Delta t) &= \mathbf{v}_i(t)+\dot{\mathbf{v}}_i(t)\Delta t\end{aligned}$$

First use the Newtons's equation of motion to compute $\dot{\mathbf{v}}_i$, namely $\dot{\mathbf{v}}_i = \mathbf{F}_i/m_i$. Run a molecular dynamics simulation on a dimer of two particles using your binary Lennard-Jones subroutine. Choose a timestep which is not too large. Plot the total energy along the molecular dynamics trajectory. You will observe a slight increase in the total energy as the simulation progresses. Although the energy is not conserved, the Euler algorithm will be good enough to perform the FIRE relaxation.

3. Now we will modify the velocity verlet algorithm such as to obtain the FIRE geometry optimization algorithm. Modify the velocity update during the molecular dynamics simulation as follows:

   - set initial values: $\Delta t = \Delta t_0$, $\alpha = \alpha_{start}$, $\mathbf{v} = 0$

- use normal molecular dynamics to compute $\mathbf{x}$, $\mathbf{F} = -\text{grad}(E)$ and $\mathbf{v}$
- check if the force norm is converged, and if converged: stop!
- calculate the power $P = \mathbf{F} \cdot \mathbf{v}$. This value will tell us if moving along the downhill ($P > 0$) or uphill ($P < 0$) direction on the energy landscape.
- set $\mathbf{v} \to (1-\alpha)\mathbf{v} + \alpha\hat{\mathbf{F}}|\mathbf{v}|$. Try to understand this update mechanism. How do we need to choose α in order to arrive at the steepest descent algorithm?
- if the downhill motion is retained ($P > 0$) in at least N_{min} censecutive steps, then $\Delta t \to \min(\Delta t\, f_{inc}, \Delta t_{max})$ and $\alpha \to \alpha f_\alpha$. It is important not to accelerate the dynamics initially (for the first N_{min} steps) to get a stable relaxation.
- if we encounter an uphill motion ($P \leq 0$), then $\Delta t \to \Delta t\, f_{dec}$, $\mathbf{v} \to 0$ and $\alpha \to \alpha_{start}$. This means that we immediately stop the MD and do a restart if we encounter a negative power P during the relaxation.
- return to MD and repeat

Good parameters for the FIRE algorithm are:
$N_{min} = 5,\ f_{inc} = 1.1,\ f_{dec} = 0.5,\ \alpha_{start} = 0.3,\ f_\alpha = 0.99$. Good values for the timesteps are $\Delta t_0 = 5.d-3$ and $\Delta t_{max} = 1.d-2$, but feel free to tune the parameters.

4. Perform a statistical comparison to the steepest descent method implemented in the previous exercises (the best modification is with gradient feedback). As a statistical dataset you can use the putative global minima structure found on the Cambridge cluster database with small random displacements from equilibrium. How do the methods compare for small clusters (less than 15 atoms), how for large clusters (more than 100 atoms)? What do you think is the explanation of your results?

6 Atomistic simulations and molecular dynamics

Atomistic simulations are simulations where the atomistic structure of matter is fully accounted for. This is in contrast to macroscopic simulations where the basic quantities are macroscopic quantities such as density or pressure. In atomistic simulations the exact number of atoms in the system, the positions of the atoms and their velocities are typically known. This atomic resolution has of course its price. One can not simulate macroscopically large systems. Those systems would contain an astronomically large number of atoms and the simulation would take a quasi infinite time even on the fastest computer. The number of atoms that can be treated in an atomistic simulation varies depending on how accurately the interactions among the atoms are treated. In principle the interactions between atoms are mediated through the electrons which have to be treated by quantum mechanics. If the electrons are treated with the most accurate quantum mechanical methods one can at most treat molecules containing a few atoms. With less accurate but nevertheless predictive methods such as density functional methods one can treat up to a few hundred atoms. Quantum mechanical electronic structure methods are an advanced topic that will not be treated in this course. In this course we will assume that the interactions between the atoms are described by so-called force fields with sufficient accuracy. Force fields eliminate the electrons entirely and are therefore not quantum mechanical methods. One assumes that the interactions between the atoms can be described by some classical

potential. This classical potential is not of a simple form such as the gravitational potential but it contains in general fairly complicated terms. The general form of these terms is guided by chemical intuition. Nevertheless it varies for different force fields. The final form of a force field requires to fix many variables. The value of these variables is found by fitting such that various quantities, that can be calculated with a force field, agree either with experimental data or with data that has been calculated with highly accurate quantum mechanical methods. As a consequence of this construction force fields typically give fairly accurate results for systems that are similar to the systems that were used for the fitting procedure but for system that are very different they can fail.
A force field is an expression for the energy of the systems as a function of the atomic positions. The force fields from chemistry contain the following basic terms

- A bond stretching energy: $const\ (|\mathbf{R_i} - \mathbf{R_j}| - d_{i,j})^2$. Only atoms i and j that are connected by bonds of length $d_{i,j}$ give such an energy contribution. The definition which atoms are connected by bonds is based on simple geometric criteria and is fixed at the start of the simulation. As a consequence, bonds can not be broken or formed during the simulation

- Bond bending terms, that rise the energy if the angle formed by the two atoms bonded to a central atom, deviates from the ideal angle for the bonding configuration. The determination of the ideal angle is usually based on the expected hydbridisation of the central atom and is also fixed at the start of the simulation. Hence

a carbon atom can for instance not change from being sp^3 bonded to being sp^2 bonded during a simulation.

- Torsional terms, that depend on how planar a sequence of 3 bonds is

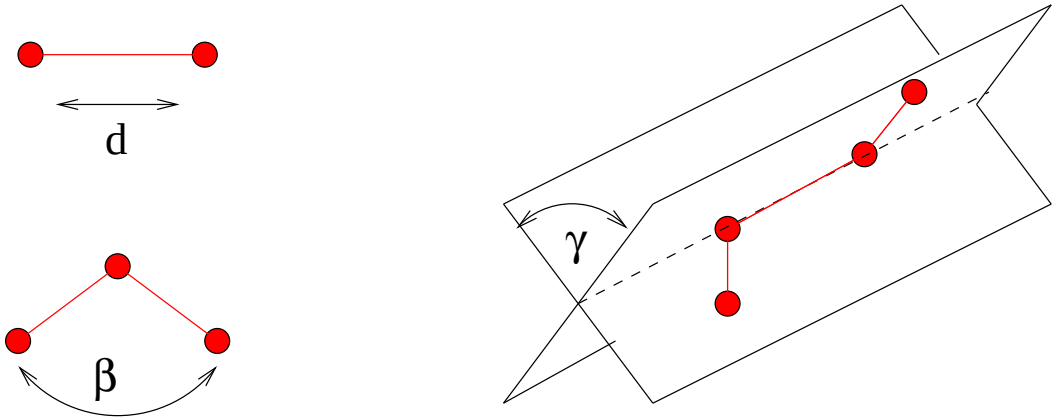


Figure 1: *bondstretching: the distance d changes, bond bending: the angle* β *changes, torsion: the angle* γ *changes*

Depending on the force field more sophisticated terms are included, such as terms that couple torsions to stretchings or higher than quadratic terms for the stretching part. All the terms listed up to now have the property that they are short range. Physically this means that the energy of an atom is not influenced by what is going on further away. Only close by bonded neighbor atoms count. Computationally this means that the energy and forces

can be calculated with linear scaling as will be discussed in more detail soon. Force fields of this type are also widely used in theoretical biology to simulate large biomolecules such as proteins.
In addition to these short range interactions there are long range terms:

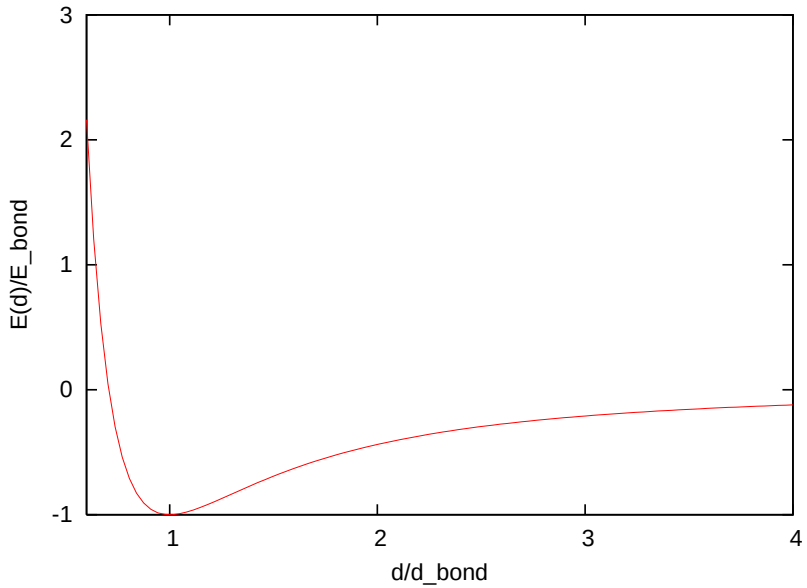
- Van der Waals terms
- Electrostatic terms

Some force fields that originated in the physics community are more sophisticated. They allow for bond breaking and forming as well as for hybridisation changes. At the start of a simulation one has to specify only the atomic positions, but not their connectivity. Their disadvantage is that they typically exist only for materials composed of a single type of atoms such as silicon and do not allow to put other chemical elements into the system. These kinds of force fields are also frequently called interatomic potentials. For technologically important materials such as silicon or carbon several different force fields can be found in the literature. Among the best force fields for silicon available today is the EDIP (Environment Dependent Interatomic Potential) force field that will be used in the project on silicon melting.

6.1 Structure determination

At zero temperature the structure of a molecule or solid is given by the condition that its energy is minimal. Hence, if we want to find the structure with the help of a force field we have to vary the positions of the atoms until we find a minimum. From the mathematical point of view we thus have to find the minimum of a function which is the force field energy expression. This function that represents the dependence of the energy on the atomic positions is called the potential energy surface. We have thus to find minima 'on' this potential energy surface. The potential energy surface is typically a complicated function with many minima. For the moment we will neglect the fact that many minima exist and assume that we just want to find a single minimum. Numerical methods to find the minimum have previously been discussed.

The situation is illustrated below for the simplest case of a diatomic molecule such as H_2. Obviously, in this case the energy depends only on the distance $d = |\mathbf{R}_1 - \mathbf{R}_2|$ between the two nuclei. Some reactive force field (i.e. a force field that allows for bond breaking) may give the curve shown below. The structure of such a diatomic molecule is described by a single number, which is the bond length. This bond length d_{bond} is the length that minimizes the force field energy expression $E(d)$.

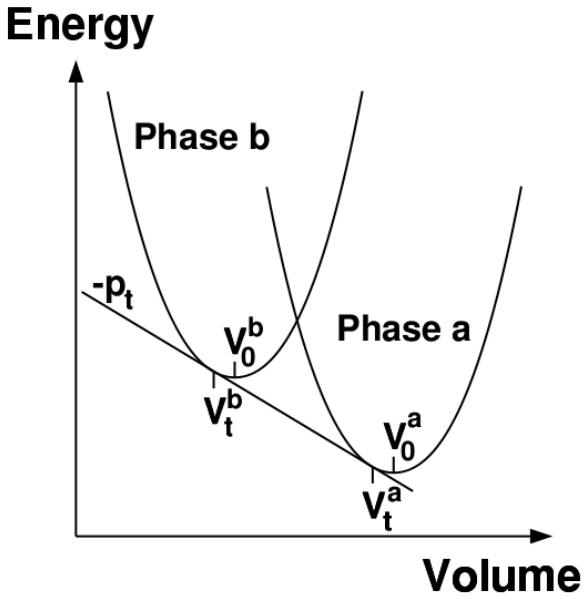


At finite temperature the structure of condensed matter is determined by the condition that the free energy $F = E - TS$ is minimal. In many cases the entropy S is small and the term TS can be neglected at room temperature. So the structure is again determined by the condition that the energy is minimal. There are however exceptions and it will be discussed later on how to calculate the entropy contributions to F.

PROJECT: Phase transition of Silicon from cubic diamond to β-Sn

Physical background

In this project you will calculate the transition pressure p_t for the cubic diamond to β-Sn phase transition of silicon under hydrostatic compressions. The method to determine the transition pressure is based on the fact that the negative derivative of the energy with respect to the volume is the pressure, $p = -\frac{\partial E(V)}{\partial V}$. So a tangent touching an energy curve $E(V)$ in a $E-V$ diagram will have a slope corresponding to $-p$ at the point of contact. For two phases a and b in a $E-V$ diagram, commonly approximated by two parabola, a common tangent will have the slope corresponding to the pressure $-p_t$. At this pressure both phases can coexist (see figure below). This common-tangent construction will also give the transition volumes of the two phases, which are different from the equilibrium volumes.



The calculations will be performed with density functional theory (DFT) calculations using the *abinit* plane wave package. DFT calculations are used to obtain an approximate solution of the many-electron Schroedinger equation and is the most common method to perform *ab initio* calculations. With DFT many-electron systems can be treated with a reasonable amount of computational cost while still giving good accuracy on most physically interesting quantities (like equilibrium geometries). It is being successfully used not only in physics, but also in chemistry, biology and material science. You will learn how to use the *abinit* plane wave package to perform simple energy calculations as a function of the unit cell volume. The results will then be used to creat a $E-V$ diagram of the two silicon bulk phases cubic diamond and β-Sn and the transition pressure is then computed.

Tasks

1. First install *abinit* either by downloading the source file from http://www.abinit.org/ or, if you have ubuntu installed on your system, by getting an older version of the package from the ubuntu repositories. You may want to run some tests to be sure that *abinit* was installed correctly, or do some tutorial tasks also available on http://www.abinit.org/.

2. The structure of the cubic diamond (relaxed at 0GPa) and the β-Sn (relaxed at 12GPa) phase of silicon are available at http:/comphys.unibas.ch/teaching.htm in a format which can be visualized using the
v_sim tool http://inac.cea.fr/sp2m/L_Sim/Vi_Sim/index.en.html (also available from

the ubuntu repositories). Before running *abinit* you need to prepare the input files. In chemistry, the valence electrons provide the most important contribution to the interatomic interaction. Therefore, in DFT calculations, the core electrons are very often replaced by pseudopotentials fitted to all-electron calculations. For silicon the following pseudopotential file is needed:
ftp://ftp.abinit.org/pub/abinitio/Psps/LDA_HGH/14si.4.hgh. The *abinit* software will read the input parameters for the calculations from a file, which has usually the extension ".in". This file also includes all unit cell parameters. We have prepared two input files for a single energy calculation called diamond.in and betatin.in. Read the input file and try to understand the meaning of each line. All keywords are commented in the file and there is a good user guide on the *abinit* web page with a detailed documentation on all keywords. In *abinit*, the keywords are followed by the values of the parameters related to the keyword. However, the ordering of the keywords within the input file is not important.

Sample the energies with respect to the unit cell volume using modified input files. For this you need to compress and expand the given unit cell and compute the energies. This can be done by modifying the parameters of "acell", which contain the unit cell vector length. Of course, the atomic positions, contained in the keyword "xcart", also need to be scaled according to the change of unit cell volume. The unit cell volume will be computed automatically by *abinit* and can be found by looking

for the variable “ucvol” in the output file. Do not compress or expand the cell to much since the quadratic region of the energy vs. volume function is not very large. So a volume change of max. $\pm 5\%$ around the energy minimum should be enough. Be careful since the β-Sn structure is not at the energy minimum, but at the enthalpy minimum at 12 GPa. The energy values can be found in the output file if you search for “Etotal”. To run *abinit*, prepare a file (for example diamond.files) containing the filenames with the following meaning:

ab.in	The main input file
ab.out	The main output will be put into the file
abi	The name of input wavefunctions (if any)
abo	The output wavefunctions will be written to abo_WFK
tmp	The temporary files will have a name that use the root ”tmp”
14si.4.hgh	The pseudopotential needed for this job

Then, use `abinit < diamond.files >& log` to run the job (or `abinis` for old versions).

Hint: instead of preparing a new input file for each unit cell volume you can use the keyword “ndtset”, followed by the number n_{data} of data sets you would like

to compute. Then, where the unit cell is defined, you can enumerate "acell" and "xcart" from 1 to n_{data}, like: acell1, xcart1, acell2, xcart2, and so on.

3. Plot the results you obtained from both cubic diamond and β-Sn structure in an energy vs. volume plot (don't forget to use the energy per atom). Then, fit two parabola into the two data sets. Find a tangent that touches both parabola and compute its slope. The slope is exactly the pressure p_t at which a phase transition between cubic diamond and β-Sn structure is possible. Remember that all calculations in *abinit* are performed in atomic units, and that 1Ha/Bohr^3 is 29421.033GPa.

6.2 Vibrational properties

At finite temperature the atoms of a molecule or solid are not exactly clamped at the atomic positions that give the lowest energy. Instead they perform oszillations around these equilibrium positions. Typically the thermal energies are very small compared to the variations of the potential energy surface and so the oscillations explore only a small volume around the equilibrium positions. Within this small volume the potential energy surface can be approximated by a quadratic form. For simplicity, let us first consider again the case of the H_2 molecule and let us assume that the two atoms are aligned along the x axis and that they can only move along the x axis. In this case the potential energy surface from the previous figure can be approximated around the equilibrium bond length d_{bond} by

$$
\begin{aligned}
E &= E_{bond} + c(d - d_{bond})^2 = E_{bond} + c(X_2 - X_1 - d_{bond})^2 && (86)\\
&= E_{bond} + c(X_2 - d_{bond})^2 - 2cX_1(X_2 - d_{bond})^2 + cX_1^2 && (87)\\
&= E_{bond} + \begin{pmatrix} X_2 - d_{bond} \\ X_1 \end{pmatrix}^T \begin{pmatrix} c & -c \\ -c & c \end{pmatrix} \begin{pmatrix} X_2 - d_{bond} \\ X_1 \end{pmatrix} && (88)\\
&= E_{bond} + \begin{pmatrix} X_2 - X_2^0 \\ X_1 - X_1^0 \end{pmatrix}^T \begin{pmatrix} c & -c \\ -c & c \end{pmatrix} \begin{pmatrix} X_2 - X_2^0 \\ X_1 - X_1^0 \end{pmatrix} && (89)
\end{aligned}
$$

where $X_2^0 = d_{bond}$ and $X_1^0 = 0$. The previous approximate form is identical to a Taylor development of a multivariate function up to second order terms. The first order term vanishes because of the condition that the gradient vanishes in a minimum. The generalization to the case of an arbitrary molecule or solid is straightforward. The Taylor expansion of the potential energy surface takes on the following form

$$E^0 + \frac{1}{2}\sum_{I,J}(\mathbf{R}_I - \mathbf{R}_I^0)D_{I,J}(\mathbf{R}_J - \mathbf{R}_J^0) \tag{90}$$

$\mathbf{R}_I$ is the position of the atom I and it is therefore a vector of length 3. The superscript zero (such as in $\mathbf{R}_I^0$) always refers to the equilibrium configuration. The elements of the 3 by 3 matrices $D_{I,J}$ are called the interatomic force constants.

$$D_{I,J} = \begin{pmatrix} \frac{\partial^2 E}{\partial X_I \partial X_J}\Big|_{\mathbf{R}_I=\mathbf{R}_I^0,\mathbf{R}_J=\mathbf{R}_J^0} & \frac{\partial^2 E}{\partial X_I \partial Y_J}\Big|_{\mathbf{R}_I=\mathbf{R}_I^0,\mathbf{R}_J=\mathbf{R}_J^0} & \frac{\partial^2 E}{\partial X_I \partial Z_J}\Big|_{\mathbf{R}_I=\mathbf{R}_I^0,\mathbf{R}_J=\mathbf{R}_J^0} \\ \frac{\partial^2 E}{\partial Y_I \partial X_J}\Big|_{\mathbf{R}_I=\mathbf{R}_I^0,\mathbf{R}_J=\mathbf{R}_J^0} & \frac{\partial^2 E}{\partial Y_I \partial Y_J}\Big|_{\mathbf{R}_I=\mathbf{R}_I^0,\mathbf{R}_J=\mathbf{R}_J^0} & \frac{\partial^2 E}{\partial Y_I \partial Z_J}\Big|_{\mathbf{R}_I=\mathbf{R}_I^0,\mathbf{R}_J=\mathbf{R}_J^0} \\ \frac{\partial^2 E}{\partial Z_I \partial X_J}\Big|_{\mathbf{R}_I=\mathbf{R}_I^0,\mathbf{R}_J=\mathbf{R}_J^0} & \frac{\partial^2 E}{\partial Z_I \partial Y_J}\Big|_{\mathbf{R}_I=\mathbf{R}_I^0,\mathbf{R}_J=\mathbf{R}_J^0} & \frac{\partial^2 E}{\partial Z_I \partial Z_J}\Big|_{\mathbf{R}_I=\mathbf{R}_I^0,\mathbf{R}_J=\mathbf{R}_J^0} \end{pmatrix} \tag{91}$$

The force as obtained from Eq. 90 is given by

$$\mathbf{F}_I = -\frac{\partial E}{\partial \mathbf{R}_I} = -\sum_J D_{I,J}(\mathbf{R}_J - \mathbf{R}_J^0) \tag{92}$$

Since the restoring force is linear in the displacement away from equilibrium $\mathbf{R}_J - \mathbf{R}_J^0$ the motion will be oszillatory and we can therefore make the ansatz

$$\mathbf{R}_I(t) - \mathbf{R}_I^0 = U_I^l \exp(i\omega_l t) \tag{93}$$

Inserting this ansatz into Newton's equation of motion

$$M_I \ddot{\mathbf{R}}_I(t) = \mathbf{F}_I \tag{94}$$

gives

$$\omega_l^2 M_I \mathbf{U}_I^l = \sum_J D_{I,J} \mathbf{U}_J^l \tag{95}$$

This is an generalized eigenvalue problem. Upon solving it by well known numerical methods one obtains the eigenvalues and eigenvectors of this eigenvalue problem. The eigenvalues are the squares of the frequencies ω_l of the different vibrational modes and the eigenvectors tell us which atom moves in which way if this mode is excited. A mode is called delocalized if all the atoms participate significantly to the oszillatory motion. If only a subgroup of atoms is participating a mode is called localized.

Exercise [3pt]: *Properties of vibrational modes with rotational character:*
Even though the energy of a cluster is invariant under rotations, moving by a finite amount along a rotational mode does change the energy. Show graphically and analytically that interatomic distances increase if the system moves along a mode with rotational character. To demonstrate this analytically consider a move where the new atomic position $\mathbf{r}_i'$ *are given by* $\mathbf{r}_i' = \mathbf{r}_i + \varepsilon \mathbf{u} \times \mathbf{r}_i$. $\mathbf{r}_i$ *are the old atomic positions, and* $\mathbf{u}$ *is the axis of rotation that is perpendicular to the plane of rotation. All atomic positions are measured with respect to the center of mass. Calculate the length of the initial coordinates* $\mathbf{r}_i$ *and and the moved coordinates* $\mathbf{r}_i'$. *The file LJ13.xyz contains the positions of the global minimum structure of a 13 atom Lennard Jones cluster. Using the routines hessian_LJ.py and LJ_calc.py verify first that there are 6 modes that have exactly (up to numerical noise) zero eigenvalues. The three translational modes are in general still closer to zero than the rotational modes. Generate then a random displacement vector whose elements are in the intervall [-1/2:1/2] and add this vector with varying amplitudes to the atomic positions. Start with amplitudes of 1.d-8 and double them until 1.d-2 is reached. Monitor the evolution of the 3 rotational modes. Do these eigenvalues decay linearly, quadratically or exponentially to zero?*
Next plot the energy when the cluster is moving along a rotational mode (such as mode 6) and a non-rotational mode (such as mode 7), i.e. $\mathbf{r}_i' = \mathbf{r}_i + l\varepsilon\mathbf{v}_i^k$. *where* $\mathbf{v}^k$ *is the k-th eigenvector of the hamiltonian calculated at the exact global minimum (i.e k=6,7). Plot the energy with 51 points, ie.e l=-25,25. Chose the step size* ε *such that the the energy*

varies by about 0.01 Lennard Jones units along the trajectory. Perform a fit of this energy. With gnuplot this can be done in the following way

```
gnuplot>  plot 'mode6.dat' using 1:2
gnuplot> f(x)= a + b*x**2 + c*x**4
gnuplot> fit f(x) 'mode6.dat' using 1:2 via a,b,c
gnuplot> replot f(x)
```

Is it justified that the the 3 rotational modes have $\omega = 0$*?*

6.3 Molecular dynamics

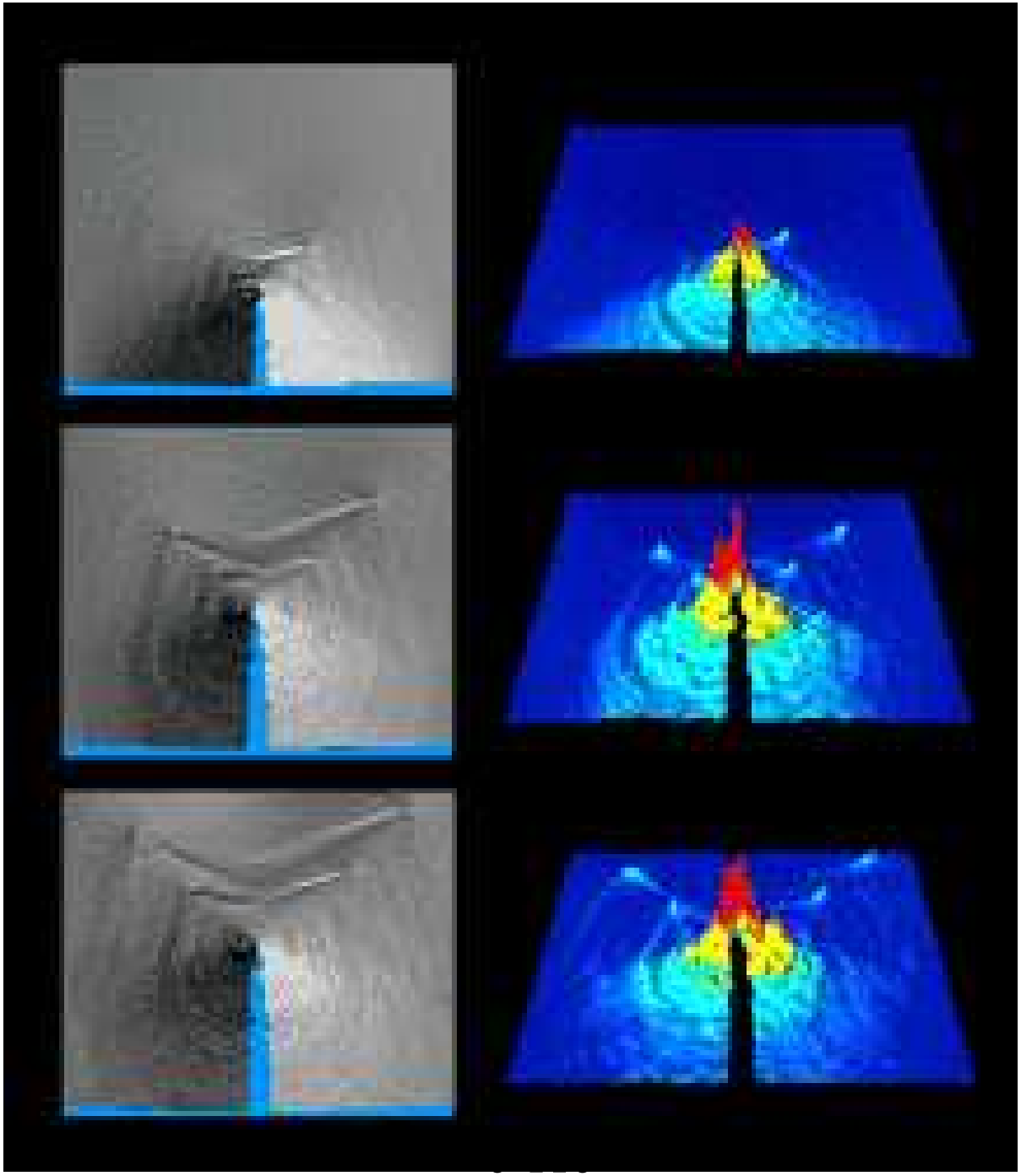
Molecular dynamics (MD) is a widely used tool in atomistic simulations. In a molecular dynamics simulation one calculates the trajectories of all the atoms in a system using Newton's equations of motion. In a numerical simulation the continuous time variable is discretized into finite time steps and for each time step one updates all the atomic positions. Molecular dynamics is like a very powerful microscope that shows all the atomic positions and their evolution in time. It can be used to

- calculate thermodynamic averages, since for an ergodic system (realistic systems are ergodic) time averages are equal to ensemble averages. For such an application MD is an alternative to Monte Carlo methods.

- In contrast to Monte Carlo methods it can also be used to calculate non-equilibrium thermodynamic properties such as transport properties.

- MD can be used to follow any atomistic process, such as a chemical reaction or the opening of a crack in a material shown below
  (taken from www.almaden.ibm.com/st/Simulate/Fracture/).

Velocity and temperature due to a crack propagation in a volume containing $\approx 10^6$ atoms



The forces acting on the nuclei (i.e. on the atoms) in a given configuration are needed for a MD simulation. These forces can be obtained in two ways

- From an ab-initio electronic structure calculation. This is in principle the prefered way, but it is numerically very costly. As a consequence one can afford only a relatively small number of time steps. The basic time scale in a MD simulation is the oscillation period of fast phononic vibrations. The MD time step is a small fraction (perhaps 1/20) of this period. Consequently one can follow with ab-initio MD only events that occur within a few vibrational periods.

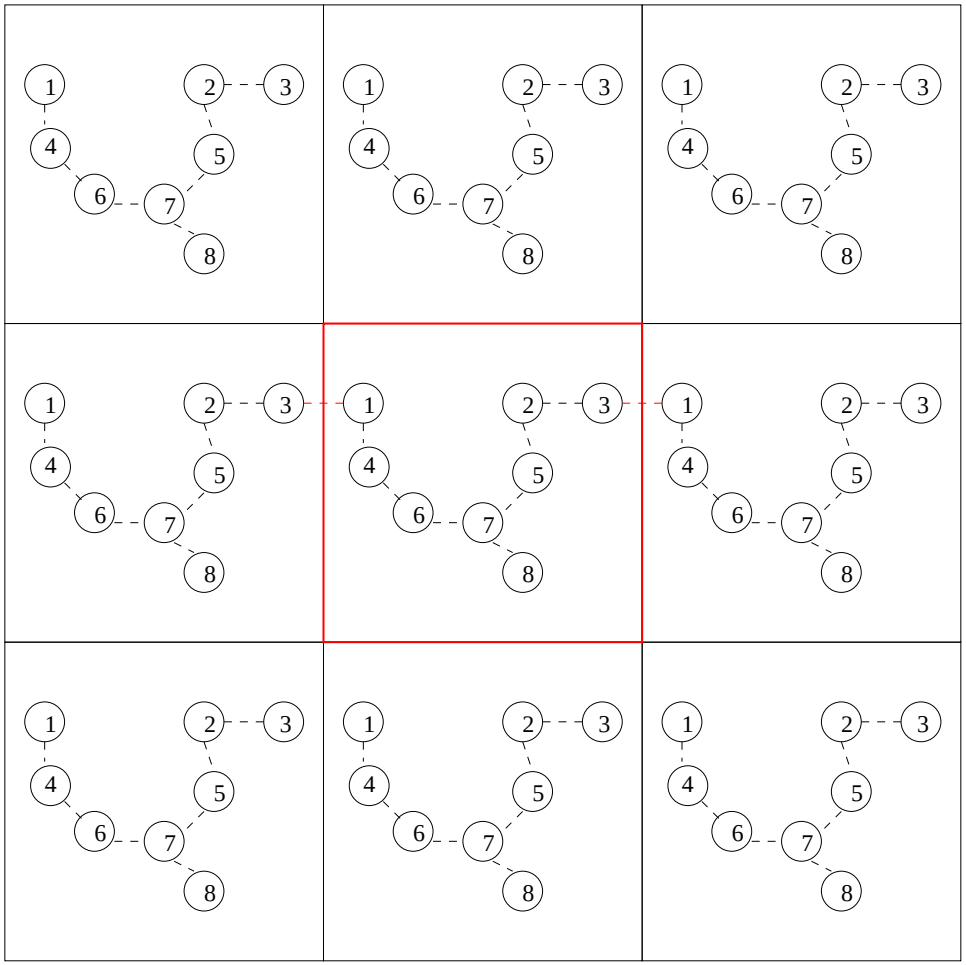
- Because of these limitations force fields are frequently used to obtain the forces. Even though the forces obtained from force fields are in general less reliable than ab initio forces, they are orders of magnitude faster to compute.

- Recently machine learning methods have been developed to represent potential energy surfaces. The can frequently achieve the accuracy of ab-initio methods but are much faster.

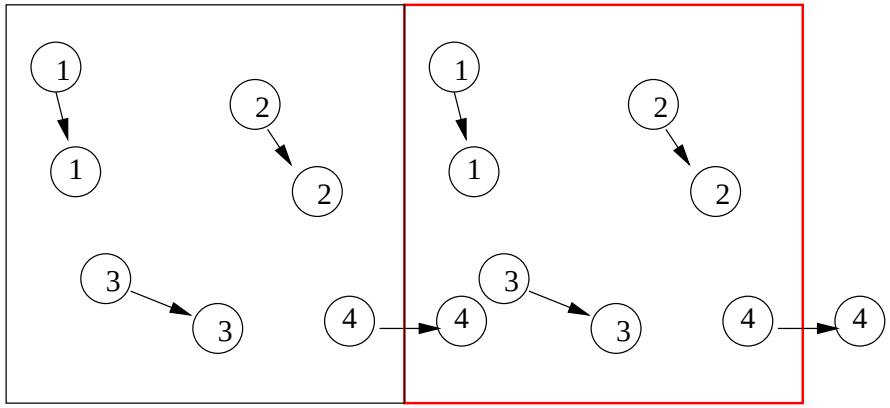
6.4 Boundary conditions

Periodic boundary conditions are a natural choice if one wants to describe crystalline solids where the basic building block is repeated quasi ad infinitum. With periodic boundary conditions, a particle near the boundary of the periodic volume interacts across the boundary with the periodic images of particles at the opposite end of the periodic volume. Periodic boundary conditions are also useful in many other situations. If one wants to simulate a liquid, one could put it into some kind of virtual container that could in a simulation for instance be modeled by a strongly repulsive potential. Close to the wall one would then have surface effects and in order to have the influence of these surfaces small, a very large system with a larger volume to surface ratio would need to be simulated. It turns out that periodic boundary conditions eliminate to a large extent surface effects and the thermodynamic limit can be approached much faster than with other boundary conditions. This is intuitively understandable. With periodic boundary conditions any particle is surrounded by other particles and no particle is close to a surface. The same considerations hold also true if one is studying a molecule in solvation.

Periodic boundary conditions are illustrated below. The particles 1 and 3 that would only interact with particles 4 and 2 under non-periodic boundary condition because no other particles are within the cutoff range of the potential, are now interacting with the periodic images of 3 and 1 respectively.



The motion of particles in a molecular dynamics simulation is also affected by periodic boundary conditions. A particle that leaves the box on one side is entering with the same velocity through the opposite side. This is illustrated in the figure below for particle 4. The figure shows the positions of 4 particles at two consecutive time steps in a molecular dynamics simulation. Whereas in the previous figure all the periodic images were shown, only the relevant periodic image is shown below.



6.5 Time propagation algorithms for MD

In this section we will assume that the forces are given and concentrate onto how to use them to numerically solve Newton's equation of motion:

$$M_i \frac{d^2\mathbf{R}_i}{dt^2} = \mathbf{F}_i \tag{96}$$

A good time propagation scheme should have the following properties:

- Short term accuracy
  The accuracy of an algorithm measures the difference in the true trajectory satisfying Newton's equation of motion and the numerical trajectory which is a finite sequence of atomic positions. One has to distinguish between the short term error, i.e. the error that one encounters if one follows a trajectory only over a short time interval and the long term error. The short term error is obviously related to the order of the finite difference formula that is used to calculate the second derivative in Newtons equation.

- Long term accuracy
  The long term accuracy is not related to the short term accuracy. As a matter of fact long term accuracy is more important than short term accuracy since in molecular dynamics one is frequently doing millions of time steps. The most important

aspect concerning the long term stability is the conservation of energy. As for the continuous case energy conservation is satisfied if one has time reversibility. Time reversibility means that the system would trace back its trajectory in phase space if one were to reverse all the velocities at a given instant in time. Let us now give a simple argument to show that a reversible time propagation algorithms can not give rise to a systematic drift of some quantity away from the true value, but that it can only fluctuate around this true value. The proof of this property is by contradiction. Let us assume that some quantity such as the total energy, which should be conserved, is increasing during a simulation. The total energy at the end configuration E_B would thus be larger than at the initial configuration E_A, $E_B > E_A$. Running backward we would get $E_A > E_B$ which is a contradiction.

- Area preservation in phase space
  As does the true Newtonian dynamics, the numerical time propagation scheme should conserve any volume element in phase space.

- Insensitivity to rounding
  Since the time propagation scheme is used in floating point arithmetic, it should be insensitive to rounding errors.

In spite of numerous research efforts the simple Verlet algorithm, obtained from a simple first order finite difference approximation to Newton's equation, has turned out to be one of the best according to the above criteria:

$$\mathbf{R}_i(t+h) = 2\mathbf{R}_i(t) - \mathbf{R}_i(t-h) + \frac{h^2}{M_i}\mathbf{F}_i(\mathbf{R}_i(t)) \tag{97}$$

h denotes the time step. It is easy to see that the Verlet algorithm is time reversible. Replacing h by $-h$ in Eq. 97 gives back the same prescription. Its short term accuracy is only moderate, namely h^2. There is a second version of the Verlet algorithm, the velocity Verlet algorithm, which is identical in exact arithmetic to the original Verlet algorithm of Eq. 97 but which is more stable in finite precision arithmetic.

$$\mathbf{R}_i(t+h) = \mathbf{R}_i(t) + h\mathbf{V}_i(t) + \frac{h^2}{2M_i}\mathbf{F}_i(\mathbf{R}_i(t)) \tag{98}$$

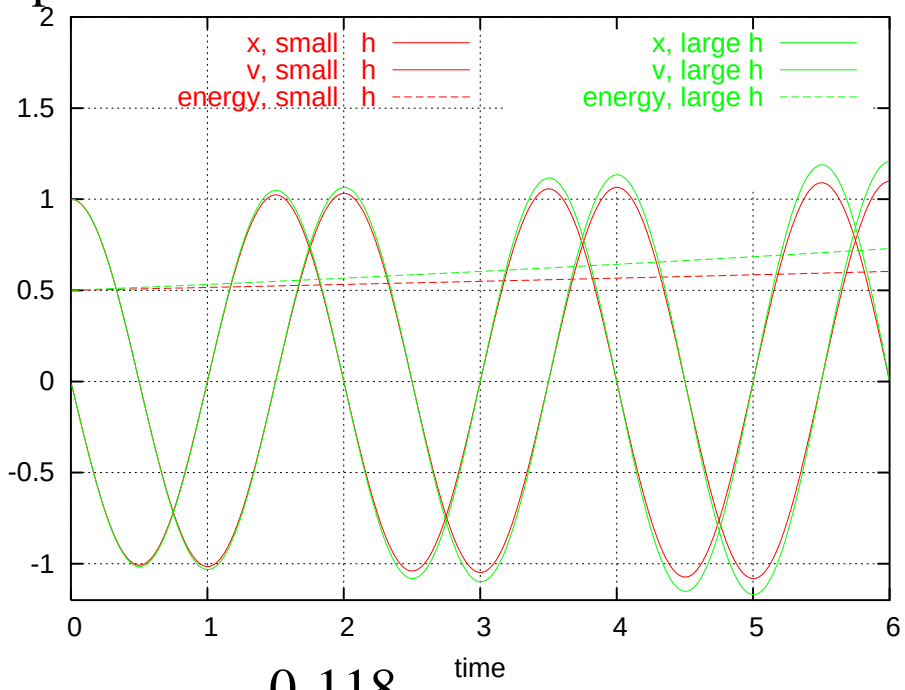
$$\mathbf{V}_i(t+h) = \mathbf{V}_i(t) + \frac{h}{2M_i}\left(\mathbf{F}_i(\mathbf{R}_i(t+h)) + \mathbf{F}_i(\mathbf{R}_i(t))\right) \tag{99}$$

The array $\mathbf{V}$ in the velocity Verlet algorithm is initially just a dummy variable that holds some partial results. However Eq. 99 suggests to look upon it as a velocity.
Exercise [1pt]: *By eliminating the velocities in Eq. 98,99 show that it is identical to Eq. 97.*

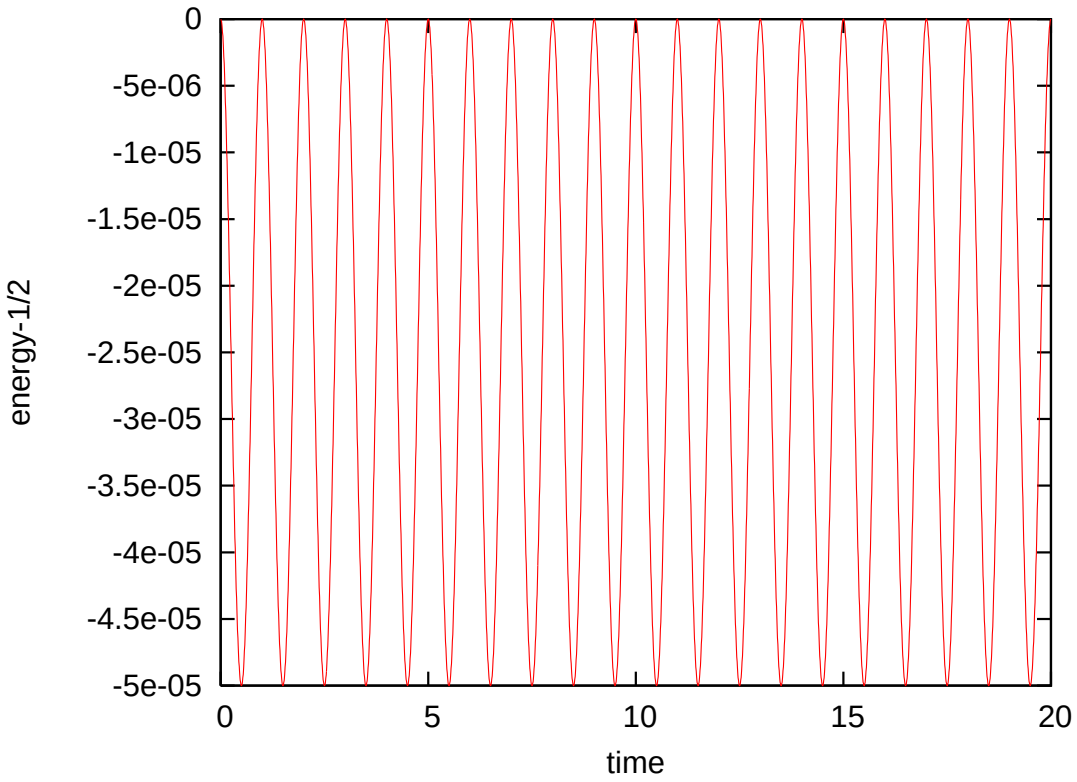
To illustrate the stability of the velocity Verlet algorithm let's contrast it with another simple integration method, namely Euler's method:

$$\mathbf{V}_i(t+h) = \mathbf{V}_i(t) + \frac{h}{M_i}\mathbf{F}_i(\mathbf{R}_i(t)) \qquad ; \qquad \mathbf{R}_i(t+h) = \mathbf{R}_i(t) + h\mathbf{V}_i(t) \tag{100}$$

Since Euler's method is not time reversible, it leads to a systematic drift of the energy away form its exact value. This is illustrated in Figure below for the case of an harmonic oscillator with an energy of $1/2$. The amplitude of both the position x and the velocity v increase leading to a rapid rise of the total energy. The increase in the total energy is slower for smaller time steps, but it does not entirely disappear and it would still be unacceptably large for simulations that follow the evolution of the system for a duration of more than a few vibrational periods.



Propagating the same harmonic oscillator with the velocity verlet algorithm does in contrast not show any drift, even for extremely long propagation times. The energy conservation over a short time interval is shown in the Figure below. The energy is oscillating with a small amplitude around the exact value without showing any drift.



Even though average quantities such as the energy are well conserved over long time intervals with the Verlet algorithm, an individual trajectory is not reliable. There is the so-called Lyapunov instability which tells us that trajectories with slightly different ini-

tial conditions diverge exponentially fast from each other. For this reason any numerical trajectory has to diverge exponentially fast from the true trajectory and in the same way two numerical trajectories with slightly different initial conditions diverge exponentially fast. This is unavoidable and can not even be cured by the best possible time propagation algorithm. This Lyapunov instability does not invalidate the MD method. As a matter of fact, all the exponentially diverging trajectories give very similar average properties.

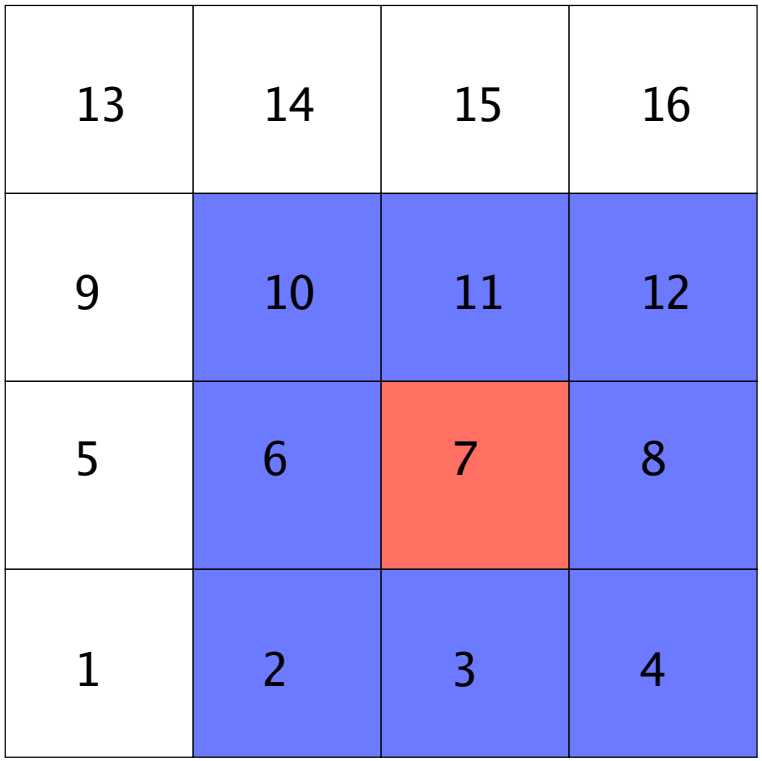
6.6 Calculating the short range forces from a force field

By definition, the short range forces cause only interactions between atoms that are close by. We will denote the radius beyond which the interation is zero by *cut*. Once one knows which atoms are close by, the calculation of these short range forces is quick. First one has however to find out which atoms are close to each other. This information is typically stored in a list that is called the nearest neighbor or Verlet list. Let us now discuss how to calculate this Verlet list.

In a trivial implementation one searches for each atom over all other atoms as shown below. The identities (i.e. their number) of the nearest neighbors are stored in the array $lstb$. The array elements $lsta(1,iat)$ and $lsta(2,iat)$ point to the starting and ending positions of the section in $lstb$ that contains the neighbors of atom iat.

```
      indlst=0
      do iat=1,nat
!        starting position
         lsta(1,iat)=indlst+1
         do jat=1,nat
         if (jat.ne.iat) then
           xrel1= rxyz(1,jat)-rxyz(1,iat)
           xrel2= rxyz(2,jat)-rxyz(2,iat)
           xrel3= rxyz(3,jat)-rxyz(3,iat)
           rr2=xrel1**2 + xrel2**2 + xrel3**2
            if ( rr2 .le. cut**2 ) then
              indlst=indlst+1
!             nearest neighbor numbers
              lstb(indlst)=jat
            endif
         endif
         enddo
!        ending position
         lsta(2,iat)=indlst
      enddo
```

The above loop has obviously quadratic scaling and would thus become for large systems the dominating part in the presence of short-range potentials only. This quadratic scaling can easily be eliminated in the following way. We subdivide the system into cells whose side length is equal to or larger than the interaction range of the potential and we assign all the particles in the system to one such cell. This can be done with linear scaling. Thus we can calculate with constant effort (independent of the total number of particles in the system) the nearest neighbors of each particle, since we have to search only in the same cell and in the neighboring cells. In the two-dimensional case illustrated below we have to search over 9 cells in the three-dimensional case over 27 cells.



0-123

The most elegant way to store the information which particle belongs to which cell is the so called linked cell list. A linked list consists of the two arrays called HEAD and LIST. The array HEAD has one element for each cell and this element contains the number of one ('the first') particle in the cell. The array LIST is of length N, where N is the number paticles, and tells us where in the list the next atom index of the atoms in this cell is stored. Both arrays together with the two-dimensional system of 16 cells containing 18 particles they are describing is shown on the next page

Navigating the list:

Head:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	9		15			20		19	2	8		13	6	12	18

List:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
				3	4					1		7		10	5	11	16	14	

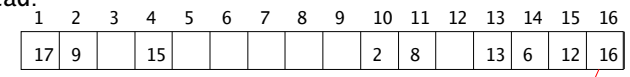
(If a place in the table is empty, the value of the corresponding array element is zero)

13 (13) (7)	(6) 14 (4)	(12) 15	(16) 16 (3) (5) (18)
9 (14) (19)	10 (2)	11 (8)	12
5	6	7 (20)	8
(17) (11) 1 (1)	(9) 2	3	(15) 4 (10)

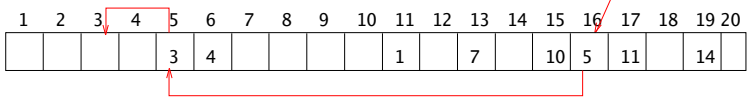
Exercise [3pt]: *Write a subroutine that constructs a linked cell list for a two-dimensional system containig N particles interacting with a potential that has a range of 1. The construction of the linked list is done 'backwards', i.e. to construct the list of particles in the subcells, one adds the elements to the heads of the list. This is illustrated on the next page. Assume that we have already assigned the atoms 1 to 17 to the arrays HEAD and LIST. Now we consider the atom 18. It is located in the cell 16. Thus we make this atom the new head of the cell 16. The cell 16 already contains the atoms 3,5 and 16.*

Add the atom 18 to the cell 16, where there are already atoms 3, 5 and 16:

Head:

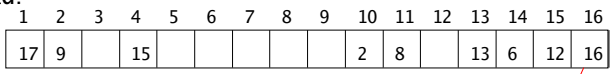


List:

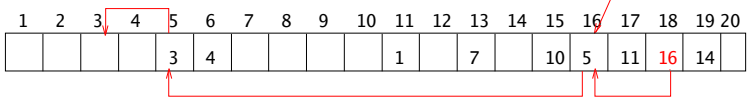


Add the atom 18 to the cell 16, where there are already atoms 3, 5 and 16,
step 1:

Head:



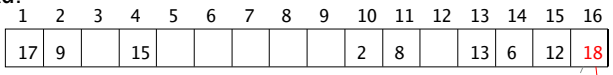
List:



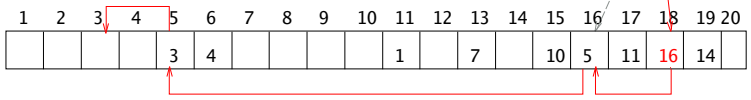
- Make the atom 18 point at the head of the cell (I.e. at the atom 16)

Add the atom 18 to the cell 16, where there are already atoms 3, 5 and 16,
step 2:

Head:

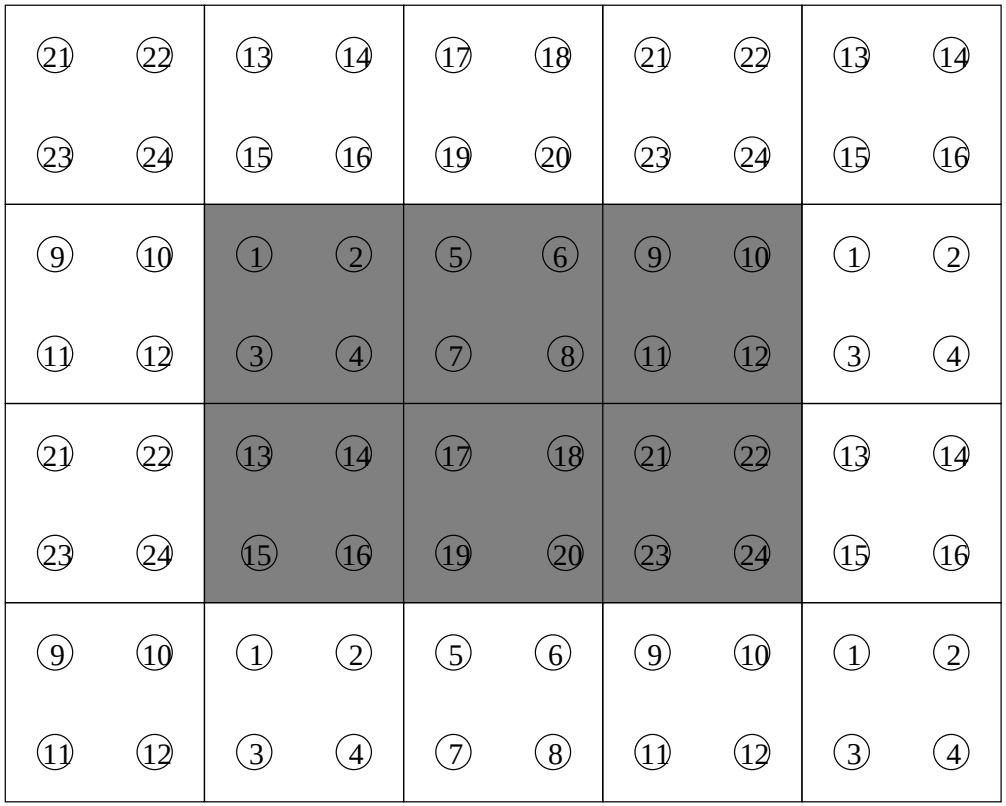


List:



- Make the atom 18 the new head for the cell 16.

If we have periodic boundary conditions, we have to duplicate the cells at the boundary of the system as shown below.



6.7 Long range forces

A straightforward evaluation of the long range forces obviously leads to a quadratic scaling. Various algorithms have been developped to reduce the scaling for the calculation of the electrostatic interactions. Some of them will be discussed later in the course. Some proposals also exist to reduce the scaling of the van der Waals interactions, but they will not be covered in this course. Even if such low complexity algorithms are used they have a large prefactor and so the computational effort for the long range part will always dominate in a simulation of larger systems. Actually because the prefactor is so big, it is not worthwhile using these low complexity algorithms for medium size systems containing a few thousand atoms.

Multiple time step MD

The largest possible time step in a MD simulation is related to the time period of a fast vibrational mode. To get good accuracy the MD step h is typically 1/10-th of this period. The problem is that large molecules and in particular biomolecules have a very large spectrum of vibrational frequencies and the period of the slowest frequencies can be larger by a factor of 1000 than the period of the fastest mode. This means that one would need 20 000 MD steps to observe just a single oszillation of the slowest mode. Since, in general,

one wants to observe many periods, the number of MD steps is gigantic and the MD simulation might require an unacceptable amount of computer time.
So-called multiple time step methods can alleviate the problem to a certain extent. In these methods the forces are subdivided into a long range and a short range part.

$$\mathbf{F} = \mathbf{F}^{long} + \mathbf{F}^{short} \tag{101}$$

The dividing line between short and long range is somewhat arbitrary and essentially dictated by the numerical savings that can be obtained from the resulting scheme. The basic idea is that the long range part of the forces changes little if the atom on which it is acting is displaced by a small amount. Such small displacements take place on the time scale of the fast vibrational modes that have mainly the character of bond stretchings. Since a bond stretching leads to a strong increase in the energy, the associated atomic displacements must be small. In a multiple time step scheme one therefore calculates the expensive long range forces not at every MD step but only let's say every tenth step, assuming that the long range forces would be nearly constant during these 10 MD steps.
Multiple time steps can be done in the context of the velocity verlet algorithms in such a way that time reversibility is retained.

6.8 Calculating the temperature in a MD simulation

The calculation of the temperature and the pressure are based on the generalized equipartition theorem

$$\langle p_k \frac{\partial \mathcal{H}}{\partial p_k} \rangle = k_B T \tag{102}$$

$$\langle q_k \frac{\partial \mathcal{H}}{\partial q_k} \rangle = k_B T \tag{103}$$

where $\mathcal{H}$ is the Hamiltonian and p_k and q_k are generalized coordinates and momenta. For the standard case where the kinetic energy part of the Hamiltonian is a simple quadratic function $\frac{1}{2M} p_k^2$ we obtain the well known equipartition principle which says that each quadratic degree of freedom has an average energy of $k_B T$.

$$\langle \frac{1}{2M} p_k^2 \rangle = k_B T/2 \tag{104}$$

For an unconstrained system of N_{at} atoms the velocities of the atoms give rise to $3N_{at}$ quadratic degrees of freedom. Hence one obtains

$$\langle \sum_{i=1}^{N_{at}} \frac{M_i}{2} \mathbf{V}_i^2 \rangle = \frac{3N_{at}}{2} k_B T \tag{105}$$

where $\mathbf{V}_i$ is the 3-component velocity vector of atom i. Since the expectation value in Eq. 105 can be calculated as an average value in a MD run, one can determine the temperature from this relation. There is one caveat. For an isolated system, that is interacting only through internal forces, the velocity of the center of mass is constant due to to the fact that the sum of all the forces vanishes. So there are three translational degrees of freedom that will not thermalize and the number of thermal degrees of freedom N_d that one encounters in the simulation of a periodic solid or liquid with periodic boundary conditions is not $3N_{at}$ but $3N_{at}-3$. If one simulates an isolated molecule there are in addition 3 types of rotational motions that can not thermalize because the torque is conserved and so the number of degrees of freedom N_d is $3N_{at}-6$. For these reasons one has to do the following two things if one wants to calculate the temperature. from MD. First of all one has to make sure that the center of mass of the system is not moving, i.e one has to apply 3 constraints. In the case of an isolated molecule 3 additional constraints have to be applied by making sure that it is not rotating. Then one can calculate the temperature from the modified relation

$$T = \frac{1}{k_B N_d} \langle \sum_{i=1}^{N_{at}} M_i \mathbf{V}_i^2 \rangle \tag{106}$$

with the appropriate value for N_d.
The temperature calculated from Eq. 106 is in principle a time dependent quantity since the velocities at time t are used for its evaluation. As a matter of fact this temperature

will not be constant but fluctuate around some average value after the system has been equilibrated. The fluctuations are getting smaller when the system gets larger and they would disappear in the thermodynamic limit of an infinitely large system. This limit is of course unattainable in a numerical simulation. For a system that is not equilibrated the temperature can take on completely unrealistic values. In order to equilibrate a system one has to allow it to do a substantial number of oszillations, which can be done with some few thousand MD steps.

6.9 Calculating the pressure in a MD simulation

If we choose Cartesian coordinates Eq. 103 becomes

$$\frac{1}{3}\langle\sum_{i=1}^{N_{at}}\mathbf{R}_i\cdot\mathbf{F}_i^{tot}\rangle=-N_{at}k_BT \tag{107}$$

The total forces can be subdivided into intermolecular forces and external forces, $\mathbf{F}_i^{tot}=\mathbf{F}_i+\mathbf{F}_i^{ext}$. The latter represent the interactions of the molecules with the walls of the container and they are therefore related to the pressure P and volume V:

$$\frac{1}{3}\langle\sum_{i=1}^{N_{at}}\mathbf{R}_i\cdot\mathbf{F}_i^{ext}\rangle=-PV \tag{108}$$

Hence we obtain

$$PV=N_{at}k_BT+\frac{1}{3}\langle\sum_{i=1}^{N_{at}}\mathbf{R}_i\cdot\mathbf{F}_i\rangle \tag{109}$$

The expectation value in the above equation is called the internal virial. It can be brought into a more convenient form that is independent of the origin

$$\sum_i\mathbf{R}_i\cdot\mathbf{F}_i=\sum_i\sum_{j\neq i}\mathbf{R}_i\cdot\mathbf{F}_{ij}=\frac{1}{2}\sum_i\sum_{j\neq i}(\mathbf{R}_i\cdot\mathbf{F}_{ij}+\mathbf{R}_j\cdot\mathbf{F}_{ji}) \tag{110}$$

where $\mathbf{F}_{ij}$ is the part of the force acting on atom i that is due to the interaction with atom j. Because $\mathbf{F}_{ij} = -\mathbf{F}_{ji}$ we can further simplify it

$$\sum_i \mathbf{R}_i \cdot \mathbf{F}_i = \frac{1}{2} \sum_i \sum_{j \neq i} \mathbf{R}_{ij} \cdot \mathbf{F}_{ij} \tag{111}$$

where $\mathbf{R}_{ij} = \mathbf{R}_i - \mathbf{R}_j$. The final form is then

$$PV = N_{at} k_B T + \frac{1}{3} \langle \sum_{i<j} \mathbf{R}_{ij} \cdot \mathbf{F}_{ij} \rangle \tag{112}$$

The same remarks that were made for the temperature apply to the pressure. To get a reliable pressure the system has to be equilibrated for a sufficiently long time. Nevertheless the pressure will always oszillate around an average value for a finite system. The expression in Eq. 112 for the virial can only be used for pairwise interactions. The calculation becomes quite complicated if the forces are calculated quantum mechanically.

PROJECT: Determination of the melting temperature of silicon by a molecular dynamics simulation

Physical background: The melting temperature of a material is by definition the temperature where a solid and liquid phase can coexist. This definition can also be used for the determination of the melting temperature by a molecular dynamics simulation. We add a piece of crystalline material to a piece of a very hot liquid and we let the system equilibrate. The equilibration might result in 3 different scenarios. If the crystalline part is very cold and/or the liquid not very hot the whole system will finally crystallize and the equilibrium temperature will be below the melting temperature.. If the liquid is extremely hot and/or the crystalline part rather warm as well, the whole system will melt, resulting in an equilibrium temperature above the melting temperature. Since melting consumes a lot of energy and since crystallization frees a lot of energy, there is however a wide range of initial conditions, where one will finally have a coexistence of the liquid and solid phase. If the initial configuration was overall hotter just a larger part will after equilibration be found in the liquid phase than if the initial configuration was overall colder.
In order to model the interaction of the silicon atoms we will use the EDIP force field (http://www-math.mit.edu/~bazant/EDIP/). A subroutine bazant_lib.f90 that implements the EDIP potential as well as all the other files needed for this project are avilable at http:/comphys.unibas.ch/teaching.htm. The bazant subroutine returns the forces needed within the velocity Verlet algorithm for the propagation of the positions.

Tasks

- Write a little program that implements the velocity Verlet algorithm (Eq. 98). Test the program for an harmonic oscillator. Check that the total energy (potential plus kinetic) is approximately conserved and that that the trajectories are periodic. Observe the quality of the energy conservation as a function of the size of the time step h.

- To do the MD simulation for silicon, a MD program md.f90 is provided. This program contains all the lines needed to read an input file with the atomic positions and velocities as well as a part that writes intermediate and final results into files that can then be used to visualize the system using the V_Sim software provided at https://gitlab.com/l_sim/v_sim. What is missing are the few lines that implement the velocity Verlet algorithms. Take over this part form the previous program for an harmonic oscillator.

- Two input files are provided: hot.dat and cold.dat. Run the MD program for 1000 time steps for one or the two input files and verify that the energy is conserved reasonably well.

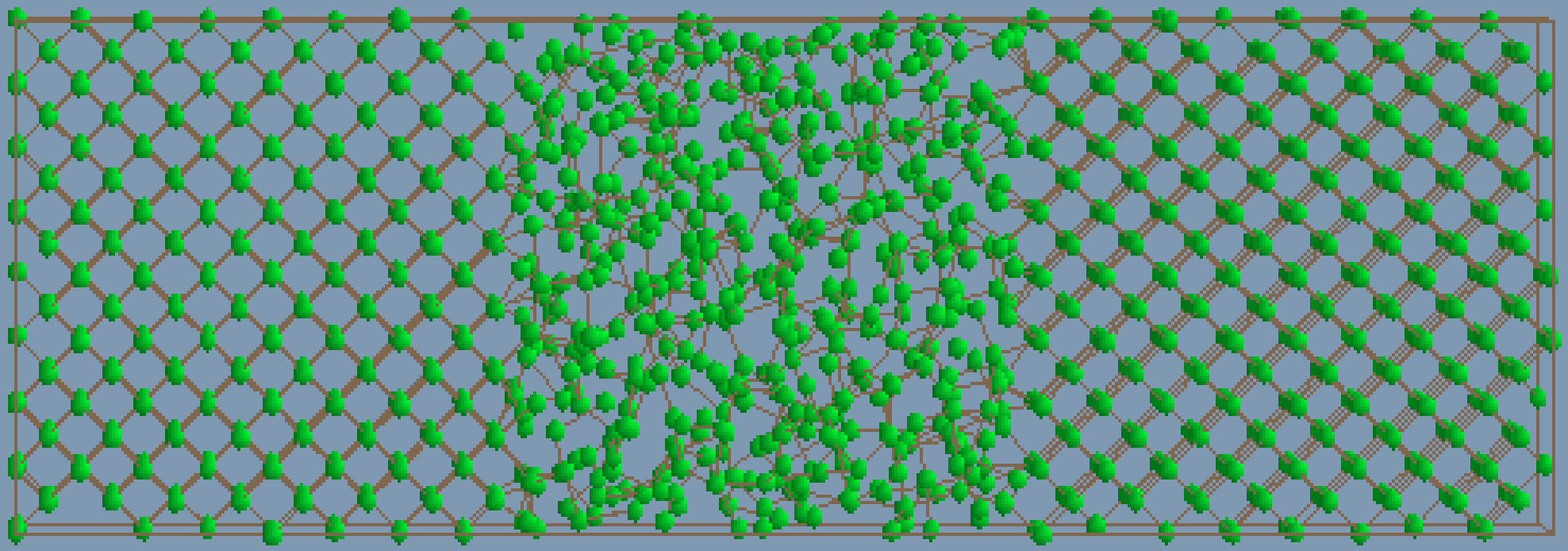
- Add a few lines to calculate the temperature T of the system using the relation

$$T = \frac{M}{(3N_{at} - 3)k_B} \sum_{i=1}^{N_{at}} |\mathbf{v}_i|^2 \tag{113}$$

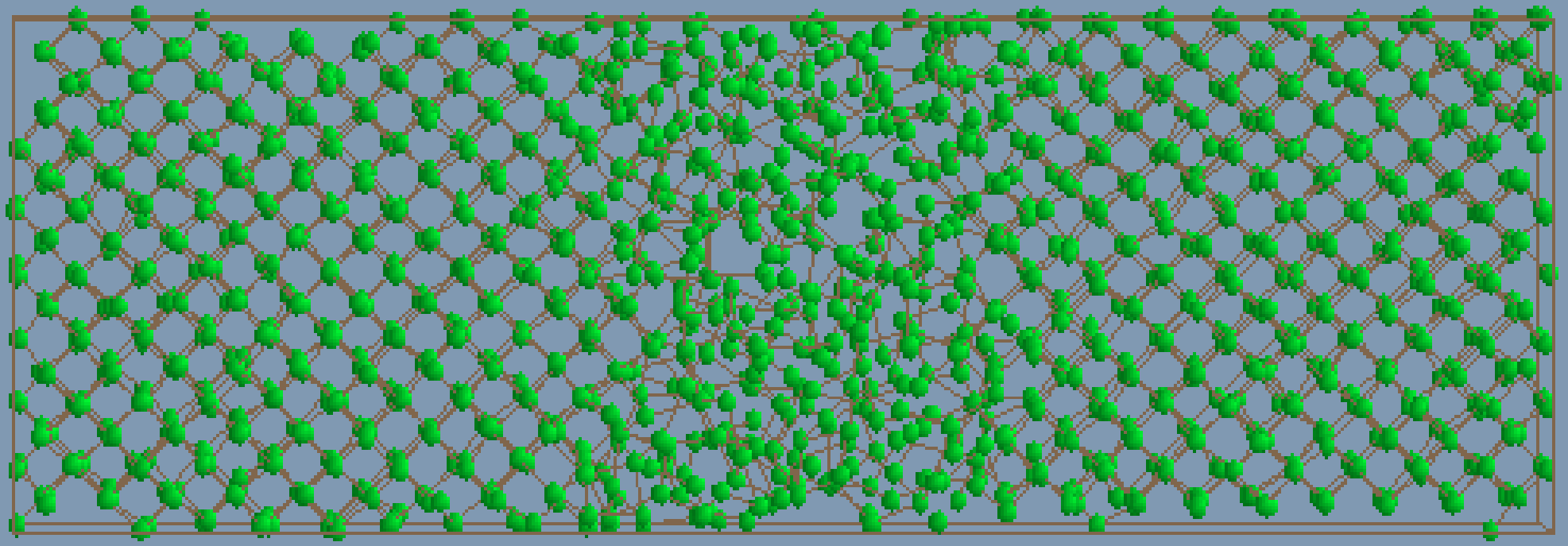
  where $\mathbf{v}_i$ is the velocity vector of the i-th silicon atom. The value of the Boltzmann constant k_B in the appropriate units as well as the mass M of the silicon atoms are contained as parameters in the program.

- Determine the melting temperature. To do this run two different molecular dynamics simulation where you use as input files input.dat either the file hot.dat or cold.dat. The atomic positions of the file cold.dat are shown on the next page, the positions in the file hot.dat are very similar. In both cases a chunk of liquid silicon is sandwiched between crystalline silicon as can be seen from the figure on the next page. The important difference is that the initial velocities of the liquid part are much higher in the file hot.dat than in the file cold.dat. The file hot.dat represents a very hot liquid embedded in a crystal, whereas the file cold.dat represents a moderately hot liquid of the same size embedded in the same crystal. After equilibration the liquid region extends therefore over a much larger volume if hot.dat is used as the input file than if cold.dat is used as input file. Two representative configurations, obtained starting from the hot and cold input files, are shown on the following pages

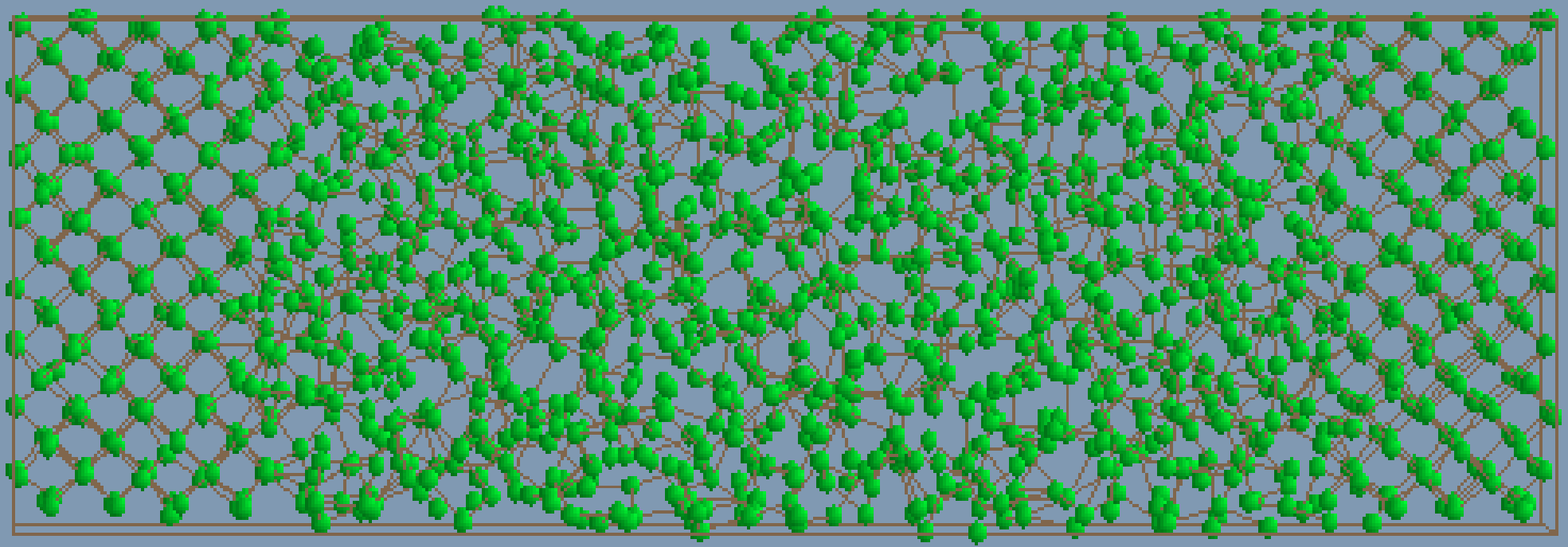
INITIAL CONFIGURATION



FINAL CONFIGURATION FORM COLD LIQUID



FINAL CONFIGURATION FORM HOT LIQUID

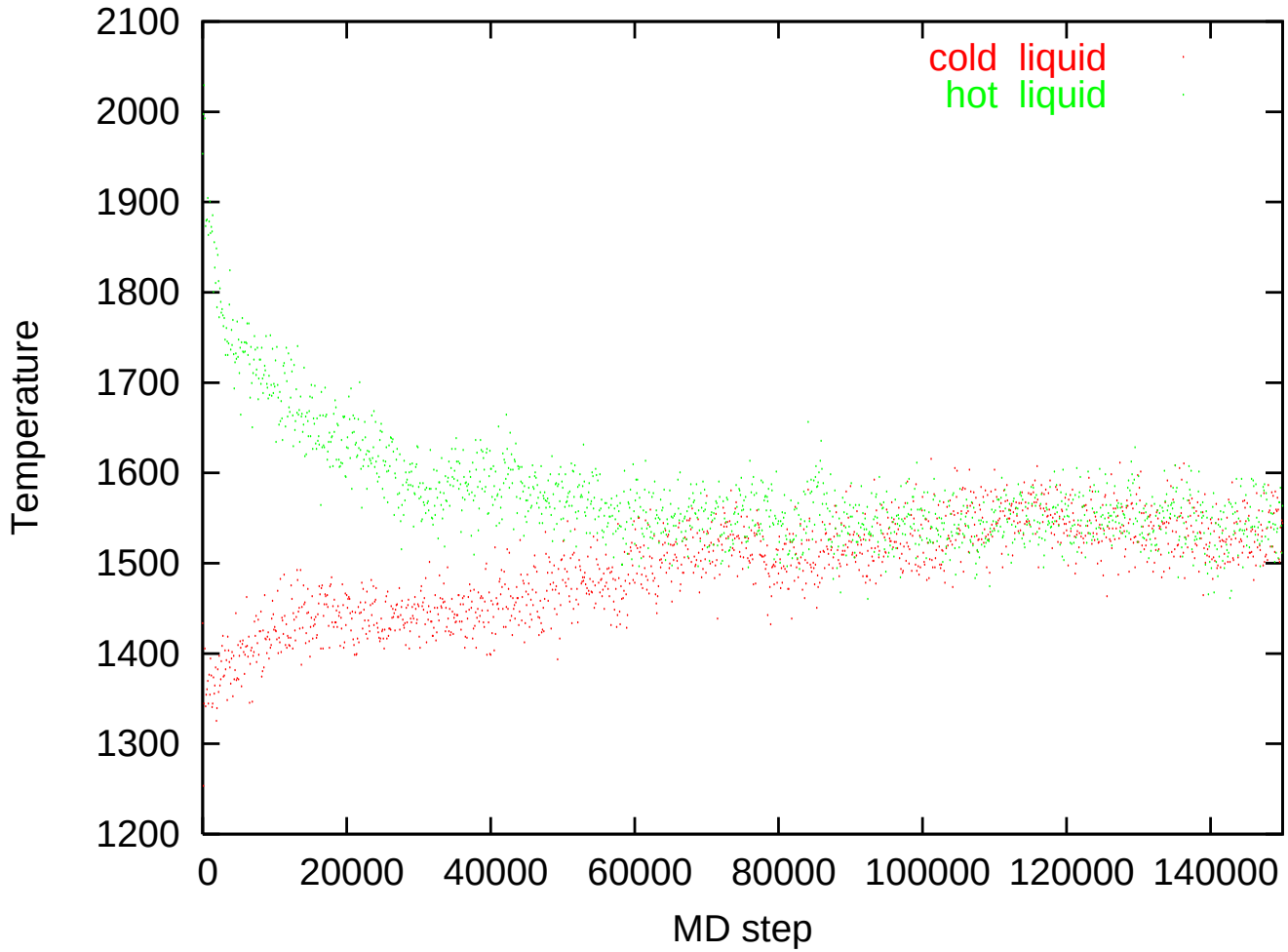


Equilibration is reached after some 100000 time steps. If computer time allows you should however run for up to 200000 time steps. Since the system is finite the temperature fluctuates and as a matter of fact these fluctuations are considerable for the system size considered here containing some 1500 atoms. So the temperature can only be estimated to within perhaps 50 degrees. Verify that both runs give an equilibrium temperature that differs by less than this uncertainty. The experimental melting temperature is around 1680 degrees. This experimental melting temperature was not used to fit the parameters of the EDIP potential.

Up to this point we have neglected one technical problem. The melting temperature depends on the pressure. Since the equilibrium volumes at a certain temperature of the liquid and solid phase are different, the pressure is different in the final configurations with a smaller or larger fraction of liquid. A clean solution to this problem would be to do a MD simulation at constant pressure instead of at constant volume as we did. Calculating the pressure is not so easy in a system with periodic boundary conditions and a MD simulation at constant pressure is technically more demanding than at constant volume. For these reasons and because the effect of pressure on the melting temperature is not very strong we will simply ignore the pressure issue.

Results

For the evolution of the temperature of the two systems you should get a plot which looks similar to the following one. Such a plot can be obtained for instance with the gnuplot software.



6.10 Calculating the diffusion coefficients in a MD simulation

Diffusion plays a fundamental role in many branches of sciences. It is described by a macroscopic law that is known as Fick's law. It states that the flux $\mathbf{j}$ of the diffusing species is proportional to the negative gradient in the concentration c of this species.

$$\mathbf{j} = -D\nabla c \tag{114}$$

Even though all the quantities involved are macroscopic, diffusion has a microscopic origin since it is caused by the thermal motion of the particles. Hence it should be possible to calculate the diffusion coefficient D with atomistic simulations. In the following it will be shown how to do this.

We will consider here the simplest case of diffusion which is called self diffusion. One assumes that all the particles are identical except for some kind of label that allows to keep track of individual particles, but that does not influence their interactions. Let us assume that at time $t = 0$ the tagged species was concentrated at the origin of our coordinate system. To compute the time evolution, we must combine Fick's law with the continuity equation

$$\frac{\partial c(\mathbf{r},t)}{\partial t} + \nabla \cdot \mathbf{j}(\mathbf{r},t) = 0 \tag{115}$$

This gives us then the diffusion equation, a differential equation for c

$$\frac{\partial c(\mathbf{r},t)}{\partial t} - D\nabla^2 c(\mathbf{r},t) = 0 \tag{116}$$

Solving it with the boundary condition

$$c(\mathbf{r},0) = \delta(\mathbf{r}) \tag{117}$$

gives

$$c(\mathbf{r},t) = \frac{1}{(4\pi Dt)^{3/2}} \exp\left(-\frac{r^2}{4Dt}\right) \tag{118}$$

We can now calculate the expectation value $\langle \mathbf{r}^2(t) \rangle$ as

$$\langle \mathbf{r}^2(t) \rangle = \int \mathbf{r}^2 c(\mathbf{r},t) d\mathbf{r} \tag{119}$$

Solving this integral one obtains

$$\frac{\partial \langle \mathbf{r}^2(t) \rangle}{\partial t} = 6D \tag{120}$$

This relation was derived by Einstein and establishes the relation between the macroscopic constant D and the microscopic mean square displacement of a particle. For a particle that

is initially not at the origin but at the position $\mathbf{R}_i^0$ the mean square displacement is simply given by $(\mathbf{R}_i(t) - \mathbf{R}_i^0)^2$. If we are interested in selfdiffusion where all particles are equal, we get better statistics if we sum over all particles and we obtain

$$6D = \lim_{t\to\infty} \sum_i \frac{1}{N_{at} t} (\mathbf{R}_i(t) - \mathbf{R}_i^0)^2 \tag{121}$$

Care has to be taken when using the above expression in the case of periodic boundary conditions. Periodic boundary conditions restrain the particle positions to remain in the finite simulation box. Consequently, the diffusion constant would tend to zero for any long time simulation. Therefore only particle positions that were not periodized are allowed to be used in Eq. 121. For this reason one has to introduce two sets of particle positions in a MD simulation that measures the diffusion coefficients. One periodized set that is used for the calculation of the interactions and another non-periodized set that is used for the calculation of the diffusion constant according to Eq. 121.
Alternatively, the displacement from the initial positions $\mathbf{R}_i(t) - \mathbf{R}_i^0$ can be obtained from an integral over the velocities:

$$\mathbf{R}_i(t) - \mathbf{R}_i^0 = \int_0^t \mathbf{V}_i(t')dt' \tag{122}$$

Hence

$$\begin{aligned}(\mathbf{R}_i(t)-\mathbf{R}_i^0)^2 &= \int_0^t\int_0^t \mathbf{V}_i(t')\mathbf{V}_i(t'')dt'dt'' \\ &= 2\int_0^t dt'\mathbf{V}_i(t')\int_0^{t'} dt''\mathbf{V}_i(t'')\end{aligned}$$

The quantity $\frac{1}{N_{at}}\sum_{i=1}^{N_{at}}\mathbf{V}_i(t')\mathbf{V}_i(t'')$ is called the velocity autocorrelation function. It measures the correlation between the velocities of a particle at time t' and t''. It is an equilibrium property of the system and as such it is invariant under a change of the time origin.

$$\frac{1}{N_{at}}\sum_{i=1}^{N_{at}}\mathbf{V}_i(t')\mathbf{V}_i(t'') = \frac{1}{N_{at}}\sum_{i=1}^{N_{at}}\mathbf{V}_i(t'-t'')\mathbf{V}_i(0) \tag{123}$$

Hence the diffusion coefficient is related to the velocity autocorrelation function in the following way.

$$6D = \lim_{t\to\infty}\int_0^t \frac{2}{N_{at}}\sum_i \mathbf{V}_i(t-t')\mathbf{V}_i(0)dt' \tag{124}$$

which becomes after taking the limit

$$6D = \int_0^\infty \frac{2}{N_{at}}\sum_i \mathbf{V}_i(\tau)\mathbf{V}_i(0)d\tau \tag{125}$$

6.11 Green-Kubo formulas

Eq. 125 is an example of a so-called Green-Kubo relation. They can be established for many other transport coefficients and relate those to various correlation functions. The electrical conductivity σ_e in the z direction is for example given by

$$\sigma_e = \frac{1}{V\,k_B\,T}\int_0^\infty \frac{1}{N_{at}}\sum_{i=1}^{N_{at}} J_i^z(\tau)J_i^z(0)d\tau \tag{126}$$

where the electrical current $\mathbf{J}_i$ is the product of the charge of the particle and its velocity, $\mathbf{J}_i = q_i\mathbf{V}_i$. J^z is the z component of this current. Similar formulas exist for the thermal conductivity and the shear viscosity.

Exercise [5pt]: *Calculation of the velocity-velocity autocorrelation function of molten sodium:*

In the supplementary material you will find files that contain a template (autocorrelation.py) and the initial positions of a 36 atom periodic sodium cell (Na36.extxyz). Particles that leave the simulation cell during the MD can be brought back within ASE with the command atoms.wrap(). In Fortran the routine back2cell.f90 is doing this. To perform the MD, implement the velocity Verlet algorithm and check that the energy is conserved up to some small amplitude oszillations. For simplicity all the atomic masses can be put to one and a reasonable time step is of the order of 0.3. The mass of sodium is 23 in atomic units. Describe the interactions of the sodium atoms with the embedded atom method

(Na_v2.eam.fs in MATERIAL folder). Set an initial temperature of 2000 K and detect when the system melts. This can be easily seen from the radial distribution function which is provided for instance in the Ovito software under "Add modification/coordination analysis. A cutoff of 8 and 100 bins are good choices. During melting the initially sharp peaks of the solid become smeared out in a nearly continuous distribution upon melting. Run then 500 MD steps for the molten system and extract from these 500 steps the velocity-velocity autocorrelation function. To get a smooth curve average over all atoms in the system and over several choices of time origins. To get the correlation after 100 time steps, use for instance not only frame 0 and 100 but also frame 1 and 101, 2 and 102 and so on up to 400 and 500.

6.12 Entropies and free energies

The free energy $F = E - TS$ is one of the central quantities in thermodynamics. At finite temperature a stable structure is given by the condition that the free energy is minimal. In this way the free energy drives also phase transitions in solids and liquids. The interpretation of these effects is simple. At finite temperature a system is not necessarily in the macro configuration of the lowest energy, but it might be in another macro configuration if many more micro states are associated to this configurations. The notion of micro state refers here to a quantum mechanical state. Let us recall that the free energy $F(T,V,N)$ depends on the temperature, volume and the number of particles and it is given by

$$F = -k_B T \ln\left(\sum_i \exp(-E_i/(k_B T))\right) \tag{127}$$

The quantum mechanical states that we have to consider if we are interested in the ground state structure are the vibrational states for this ground state configuration. At low temperature the harmonic approximation of the potential energy surface (Eq. 90) is usually valid. For this approximation it is derived in advanced solid state or electronic structure courses that the energy levels of a single frequency ω_l are given by

$$E_{l,n} = (n + \frac{1}{2})\hbar\omega_l \tag{128}$$

where ω_l is one of the non-zero frequencies obtained from solving the eigenvalue problem of Eq. 95. The vibrational state of our system is then specified by the $3N_{at}-3$ (if the only zero modes are the 3 translational modes) quantum numbers $n_1, n_2, \ldots, n_{3N_{at}-3}$ and its energy is given by

$$E = \hbar\left((n_1+\frac{1}{2})\omega_1 + (n_2+\frac{1}{2})\omega_2 + \ldots\right) \tag{129}$$

If we insert Eq. 129 into Eq. 127 we can simplify the expression for the vibrational part of the free energy by summing all the geometric series:

$$\begin{aligned}\sum_i \exp(-E_i/k_BT) &= \left(\sum_{n_1}\exp(-(n_1+\frac{1}{2})\frac{\hbar\omega_1}{k_BT})\right)\left(\sum_{n_2}\exp(-(n_2+\frac{1}{2})\frac{\hbar\omega_2}{k_BT})\right)\ldots. \\ &= \left(\frac{\exp(-\frac{1}{2}\hbar\omega_1/(k_BT))}{1-\exp(-\hbar\omega_1/(k_BT)}\right)\left(\frac{\exp(-\frac{1}{2}\hbar\omega_2/(k_BT))}{1-\exp(-\hbar\omega_2/(k_BT)}\right)\ldots\end{aligned}$$

So, within the harmonic approximation, the vibrational free energy is given by

$$F_{vib} = -k_BT\sum_l \ln\left(\frac{\exp(-\frac{1}{2}\hbar\omega_l/(k_BT))}{1-\exp(-\hbar\omega_l/(k_BT)}\right) = \sum_l \frac{1}{2}\hbar\omega_l + k_BT\ln\left(1-\exp(-\hbar\omega_l/(k_BT))\right) \tag{130}$$

where the sum is over all the nonzero frequencies ω_l

At high temperatures the harmonic approximation fails because the system can explore a larger region around the equilibrium positions where a quadratic approximation of the potential energy surface is not any more valid. It also fails for systems that have very low vibrational frequencies. In this case the harmonic approximation would imply that the system can oszillate a long distance in the direction of the eigenvector associated to this low frequency, while the energy always increases quadratically. Generally, this is however not true because for large displacements the quadratic approximation is not any more valid and other important potential contributions arise. Such systems are also called strongly anharmonic.
A technique that allows to calculate the free energy without the assumptions of the harmonic approximation is thermodynamic integration. The problem with determining the classical free energy is, that it can not be calculated as the time average of some quantity that depends on the positions or velocities.

$$F = -k_B T \ln \left(\frac{\int d\mathbf{p}^N d\mathbf{r}^N \exp(-\mathcal{H}(\mathbf{r}^N, \mathbf{p}^N)/(k_B T))}{(2\pi\hbar)^{3N} N!} \right) \tag{131}$$

Things are however different for derivatives of the free energy. For instance, it can be seen from either Eq. 127 or Eq. 131 that

$$\frac{\partial}{\partial T} \frac{F}{T} = -\frac{E}{T^2} \tag{132}$$

In the classical case the energy expectation value E is given by

$$E = \frac{1}{(2\pi\hbar)^{3N} N! Z} \int d\mathbf{p}^N d\mathbf{r}^N \mathcal{H}(\mathbf{r}^N, \mathbf{p}^N) \exp(-\mathcal{H}(\mathbf{r}^N, \mathbf{p}^N)/(k_B T)) \tag{133}$$

where

$$Z = \frac{1}{(2\pi\hbar)^{3N} N!} \int d\mathbf{p}^N d\mathbf{r}^N \exp(-\mathcal{H}(\mathbf{r}^N, \mathbf{p}^N)/(k_B T)) \tag{134}$$

E can be calculated for an ergodic system from a MD simulation as $E = \langle \mathcal{H}(\mathbf{r}^N, \mathbf{p}^N) \rangle$. Eq. 132 gives thus rise to the simplest thermodynamic integration scheme which allows us to calculate the change in the free energy $F - F_0$ as we go from an initial temperature T_0 to a final temperature T.

$$\frac{F}{T} - \frac{F_0}{T_0} = -\int_{T_0}^{T} \frac{E(\tau)}{\tau^2} d\tau \tag{135}$$

If E is evaluated through a MD simulation, quantum mechanical effects are of course absent, in contrast to Eq. 130 that contains the quantum mechanical effects. In order to get accurate results the fluctuations in the temperature have to be small. This requires long runs for large systems which require a lot of computer time.
In the quantum mechanical framework the equations corresponding to Eq. 133 and Eq. 134

are

$$E = \frac{1}{Z}\sum_i E_i \exp(-E_i/(k_B T)) \tag{136}$$

and where Z is the partition function

$$Z = \sum_i \exp(-E_i/(k_B T)) \tag{137}$$

In the thermodynamic integration scheme described above the integration was along a physical variable, namely the temperature. In a simulation one can also integrate along non-physical variables. Such variables can for instance smoothly transform one system into another one. This kind of transformation is for example necessary if one wants to know how the free energy changes upon the replacement of a side chain by another side chain in a large molecule. Let us assume that the system with the initial side chain is described by an interacting potential U_A and the system with the final side chain by U_B. We can then introduce a transformation

$$U(\lambda) = U_A + \lambda(U_B - U_A) \tag{138}$$

Obviously for $\lambda = 0$ the system is in the initial state and for $\lambda = 1$ in the final state. The free energy for a intermediate system is given by Eq. 131 except that the Hamiltonian

depends now on λ

$$F = -k_B T \sum_l \ln \left(\frac{\int d\mathbf{p}^N d\mathbf{r}^N \exp(\mathcal{H}(\lambda, \mathbf{r}^N, \mathbf{p}^N)/(k_B T))}{(2\pi\hbar)^{3N} N!} \right) \tag{139}$$

Taking the derivative of the free energy with respect to λ we get

$$\frac{\partial F}{\partial \lambda} = \frac{\int d\mathbf{p}^N d\mathbf{r}^N \frac{\partial U(\lambda)}{\partial \lambda} \exp(\mathcal{H}(\lambda, \mathbf{r}^N, \mathbf{p}^N)/(k_B T))}{\int d\mathbf{p}^N d\mathbf{r}^N \exp(\mathcal{H}(\lambda, \mathbf{r}^N, \mathbf{p}^N)} \tag{140}$$

$$= \langle \frac{\partial U(\lambda)}{\partial \lambda} \rangle \tag{141}$$

since $\mathcal{H}(\lambda, \mathbf{r}^N, \mathbf{p}^N) = \sum_i \frac{1}{2M} \mathbf{p}_i^2 + U(\lambda, \mathbf{r}^N)$. $\langle \frac{\partial U(\lambda)}{\partial \lambda} \rangle$ is again an average value that can be obtained from a MD simulation by taking the average of $\frac{\partial U(\lambda)}{\partial \lambda} = U_B - U_A$ over all the MD steps. By integration $\langle U_B - U_A \rangle$ from $\lambda = 0$ to $\lambda = 1$ one obtains the free energy difference between the systems described by U_A and U_B.

PROJECT: Concentration of point defects in silicon

Physical background: Perfect crystals can not exist at a finite temperature. There will always be defects, i.e. deviations from the perfect periodic structure. Thermodynamics can predict the concentration of various defects in crystalline materials. In this project we will consider the simplest point defect in a crystal, namely the vacancy. One obtains a vacancy simply be taking an atom out of a perfect crystal. This leads of course to some relaxations of the surrounding atoms. The concentration N_d of the vacancies is given by

$$N_d = N_{at} \exp\left(-\frac{\Delta F}{k_B T}\right) \tag{142}$$

The difference in the free energy consists of two parts, the difference in energy between the perfect crystal and the crystal with a defect ΔE and the difference in the vibrational free energies ΔF_{vib} between the same two configurations, i.e $\Delta F = \Delta E + \Delta F_{vib}$. Since we want to obtain quantities that are quite independent of the size of our simulation system we subtract from the values obtained for the system (i.e. crystal with defect) only the values that a perfect crystaline system with $N_{at} - 1$ atoms would have:

$$\Delta E = E_{sys}(N) - \frac{N_{at} - 1}{N_{at}} E_{crys}(N) \tag{143}$$

$$\Delta F_{vib} = F_{vib,sys}(N) - \frac{N_{at} - 1}{N_{at}} F_{vib,crys}(N) \tag{144}$$

Tasks

- In the file Si216.ascii (avilable at http:/comphys.unibas.ch/teaching.htm) you can find the atomic positions for a piece of crystaline silicon in the V_Sim format. The routine provided in the previous project can be used to read in this file. Calculate the energy of the crystaline system with the Bazant EDIP force field that was also used in the previous project. Verify that the forces returned by the bazant routine are virtually zero.

- Take out of the file one atom to obtain a system of 215 silicon atoms with one vacany. Relax the system by doing a simple stepest descent with energy feedback or even with a constant stepsize. A good step size is 1/50. Then you can use Eq. 143 to calculate ΔE.

- Next we have to calculate the vibrational free energy. For this purpose you have to write a subroutine that calculates the Hessian matrix of Eq. 90. Remember that the dimension of this matrix is $3N_{at}$ times $3N_{at}$. The best way to do this is to take numerical first derivatives of the gradient as returned by bazant. Take a finite difference formula that uses 4 points (i.e. 2 to the right and 2 to the left). There are various checks that you should do to verify whether this Hessian matrix was calculated correctly. It should be symmetric (up to a certain small numerical error) and it should have 3 zero (up to numerical precision) eigenvalues corresponding

to the 3 translations of the system with periodic boundary conditions. Set up the vectors corresponding to translations along the x,y and z directions and check that multiplying these vectors with the Hessian gives a nearly zero vector. These tests should be repeated for several values of h. An optimal value of h is obtained when these conditions are satiesfied with the smallest numerical error.

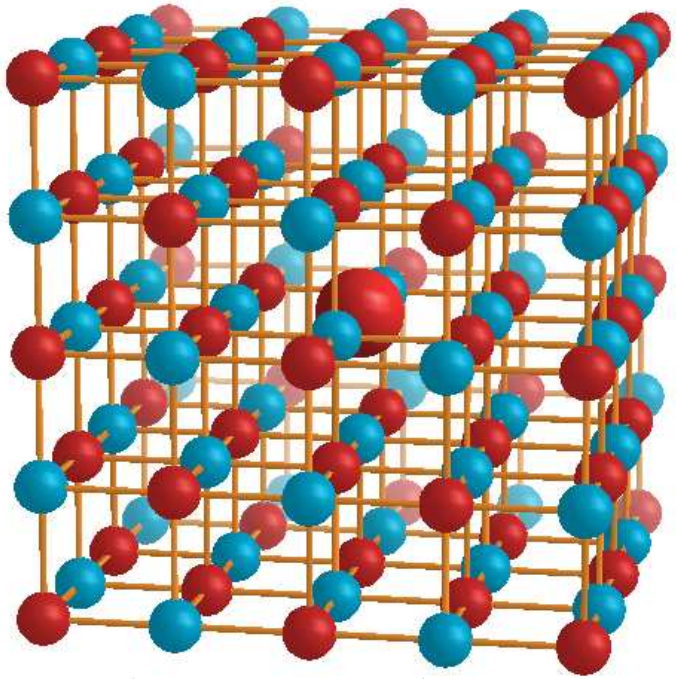
- Next we have to diagonalize the Hessian matrices for the system with and without vacancy. This can be done by using the routine jacobi.f90 (provided at http:/comphys.unibas.ch/teaching.htm) which is taken from numerical recipes. Check again whether there are 3 nearly zero eigenvalues. The eigenvalues returned by jacobi are not ordered by magnitude and so you have to search for them.

- Once we have the two sets of eigenvalues, we can calculate the two vibrational free energies according to Eq. 130, excluding the 3 zero frequencies corresponding to translations. Calculating the dimensionless quantities $\frac{\hbar\omega}{k_B T}$ requires converting all the quantities to a common system of units. Remember that the Hessian as calculated from the gradient returned by the subroutine bazant is given in units of $\frac{eV}{A^2}$. The mass of silicon found in Eq. 95 can be taken as 28 nucleonic masses. The various conversion factors needed can for instance be found on http://physics.nist.gov/cuu/constants/energy.html. Finally we can then calculate the difference in the vibrational free energy from Eq. 144.

- Plot the total difference in free energy ΔF as a function of temperature up to the melting temperature of silicon (take 1500 Kelvin). Plot the concentration of vacancies as a function of temperature using Eq. 142. You will get a very low concentration at room temperature, which would imply that even in a macroscopic sample there is not a single vacancy.

Exercise [4pt]: *Electrostatic energy of ionic materials:*

The energy of ionic materials can be approximated by a simple electrostatic model where the anions and cations have constant charges. The simplest example is sodium chloride which forms a cubic lattice (see figure below) where the Na's have a charge q_i of minus one electron and the Cl's of plus one electron. For simplicity we assume that the distance between the Na and Cl atoms is one.



Structure of the NaCl crystal. Na's are shown in red and Cl's in blue. The Madelung energy is the electrostatic energy of the large Na atom in the center in the limit where the cube gets infinitely large.

Calculating the energy E for a finite crystalline chunk of N atoms (such as shown above) is trivial

$$E = \sum_{i<j} \frac{q_i q_j}{|\mathbf{R}_i - \mathbf{R}_j|} \tag{145}$$

and by increasing the number of atoms in the summation one would assume that the average energy per atom $e = E/N$ converges to a well defined bulk value. This is however not true. The average energy e depends on the shape of the growing cluster even for very large cluster sizes. To show this effect lets consider three different cluster shapes, a cube together with the sphere and octahedron inscribed in this cube of sidelength $2n+1$. So the cube contains $N = (2n+1)^3$ atoms. First calculate just the electrostatic energy of the atom at the center of the cube (and therefore also at the center of the sphere and octahedron). This is the so-called Madelung energy. The sum required for calculating the electrostatic energy is well known to be conditionally convergent, i.e the final result depends on the summation order. Observe the convergence (divergence) behaviour with increasing n for the sphere, the cube and the octahedron. Show that the convergence is best for volumes that have charge neutrality. The exact value of the Madelung constant is 1.7475645946331 8. This value can relatively easily be calculated with an accuracy of some 10 decimal places in double precision floating point arithmetic if one assumes that the charges have an extremely small but non-zero extent. In this case only a certain fraction of the charges is contained in the growing cube that is used for the summation. Charges at a surface contribute only half to the sum, those on a edge a quarter and those

at a corner only one eight. Use this modified summation within the cube to calculate the Madelung constant with high accuracy. Finally calculate the total electrostatic energy for a cube and an octahedron both containing about one million atoms. An octahedron has only (111) surfaces. So its 8 surfaces are perpendicular to the eight vectors $(\pm 1, \pm 1, \pm 1)$. *These surfaces are either purely Na or purely Cl terminated. Check with the v_sim visualization software whether you have correctly generated the octahedral shape.*

7 Treatment of electrostatic and gravitational long range potentials

The calculation of the energy and forces coming from long range electrostatic forces (monopole, dipole interaction etc) can be numerically expensive. Let us concentrate on the slowest decaying monopole interaction. The electrostatic problem is equivalent to a gravitational problem. Because the gravitational problem lends itself more easily to a pictorial description of the algorithm, we will consider the gravitational problem.

$$E = \sum_{i<j} \frac{m_i m_j}{|\mathbf{r}_i - \mathbf{r}_j|} \tag{146}$$

Trivial algorithm

```
e=0.
do j=1,n
do i=1,j-1
  rij=sqrt( (x(i)-x(j))**2 + (y(i)-y(j))**2 + (z(i)-z(j))**2 )
  e=e+M(i)*M(j)/rij
enddo
enddo
```

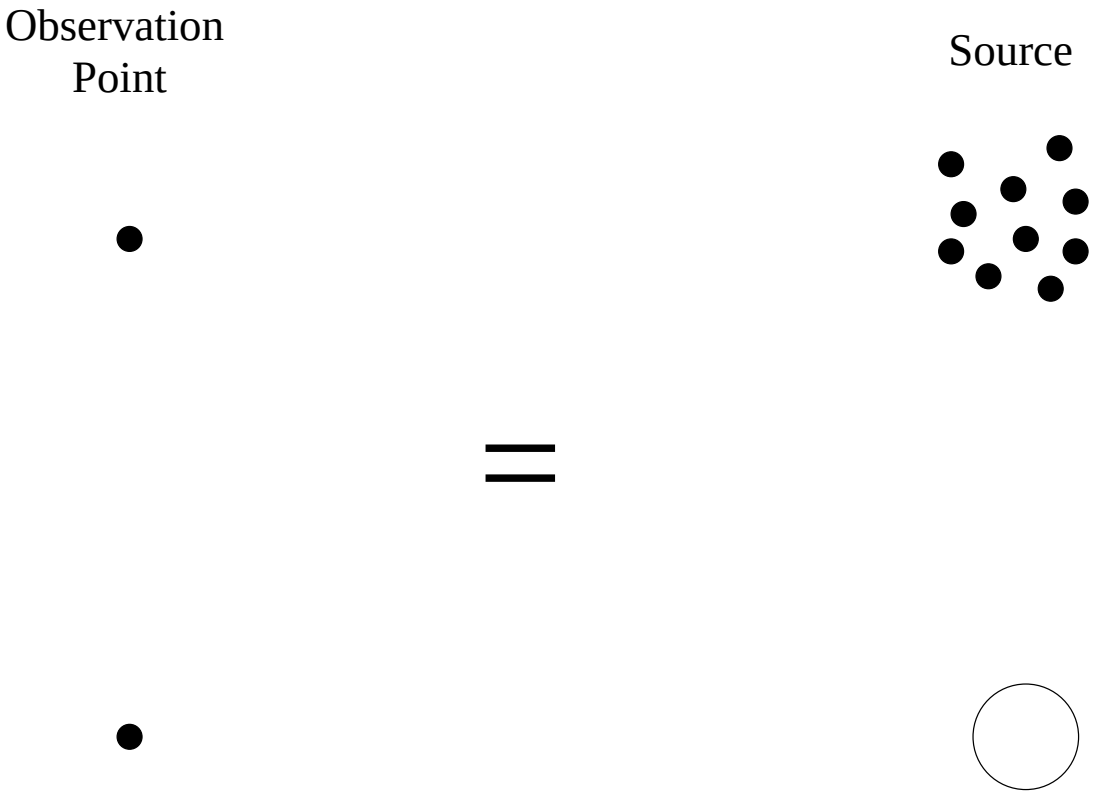
Quadratic scaling:

$$T_{CPU} \propto N^2 \tag{147}$$

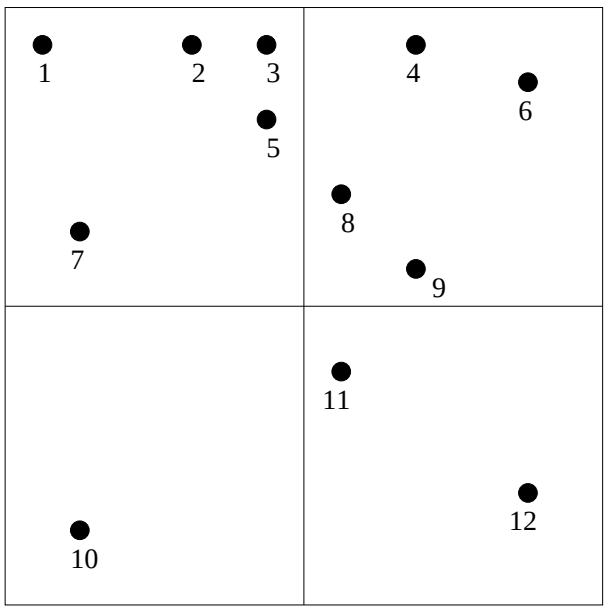
7.1 The Barnes Hut algorithm

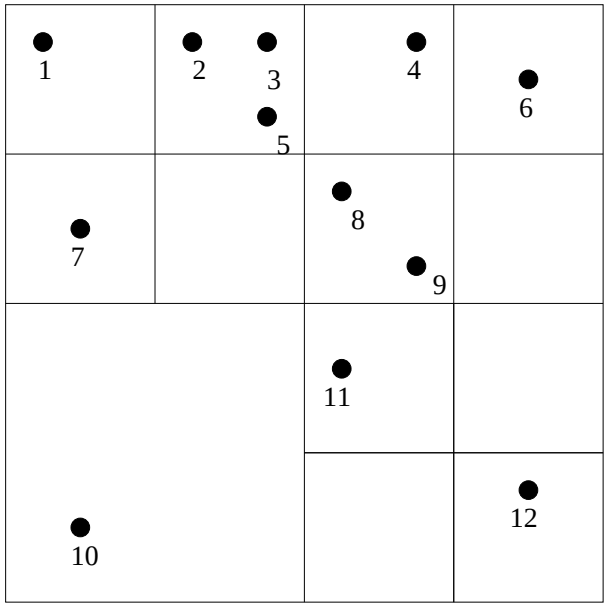
$$T_{CPU} \propto N \log(N) \tag{148}$$

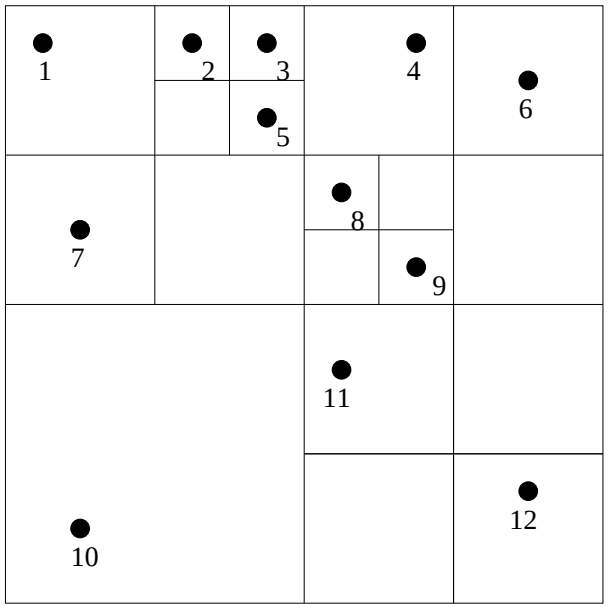
Central idea: *From far away, a bunch of stars looks like a single larger star*

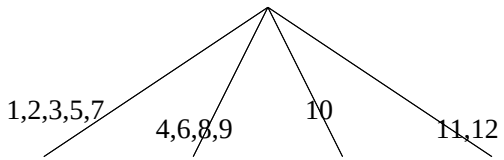


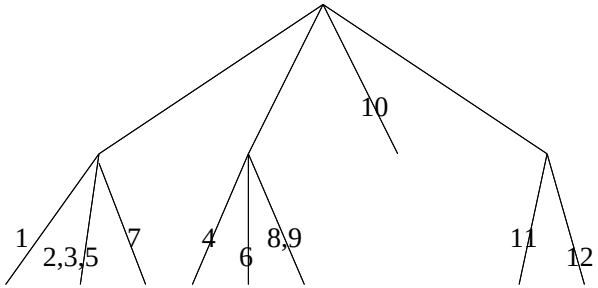
Step 1: Subdivide system and generate tree

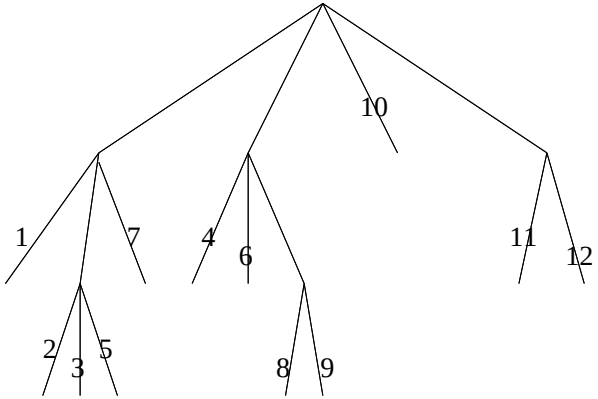






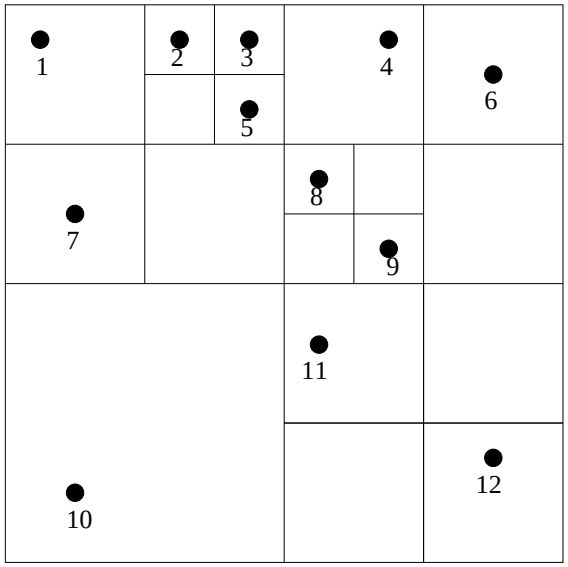


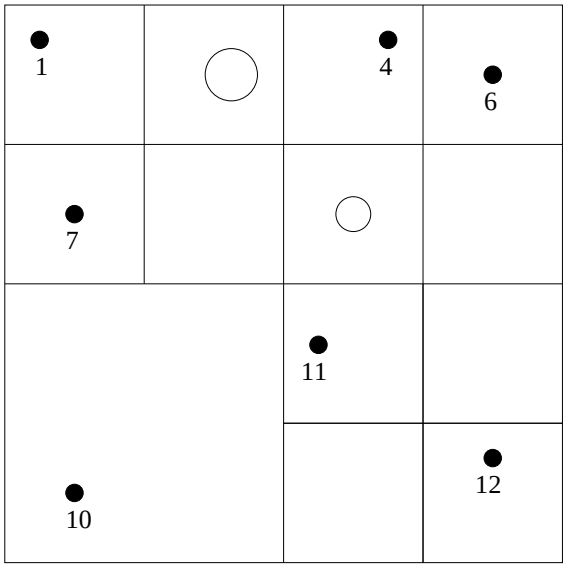


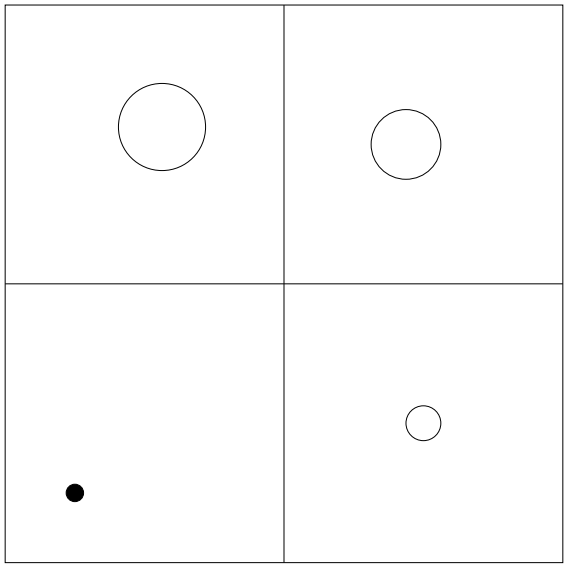


The initial box is chosen such that it contains the entire system. This initial box is then successively subdivided into sub-boxes, 4 in the 2-dim case, 8 in the 3-dim case. This subdivision is stopped when the sub-box contains a single star.

Step 2: Combine stars into super-stars, super-super-stars, ..., galaxies

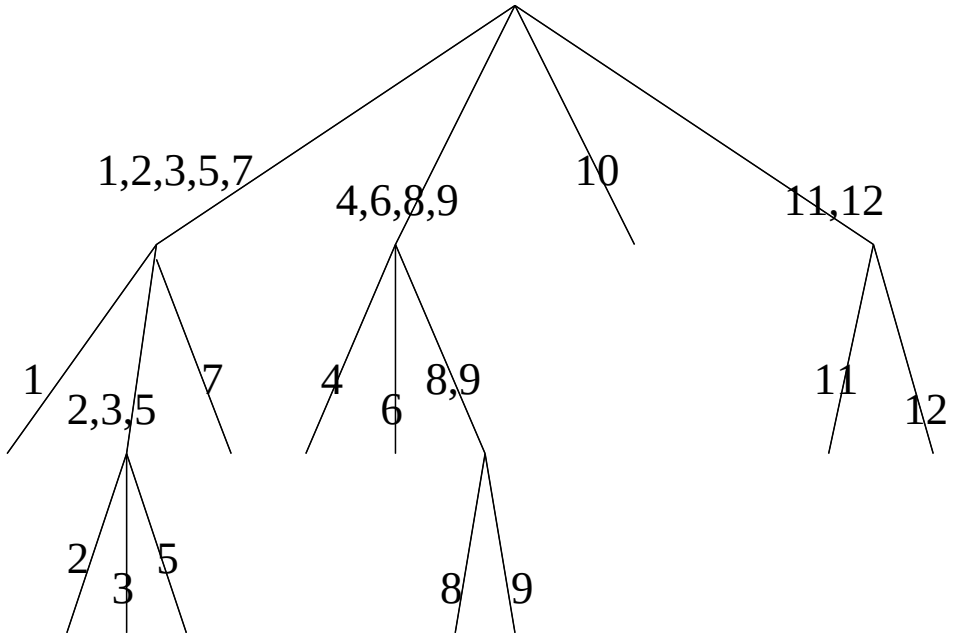






Super-stars are formed out of individual stars contained in the boxes at the penultimate subdivision level. The mass of a super-star equals the sum of the masses of its constituent stars and it is centered at their center of mass. In the same way second order super-stars (super-super-stars) are formed out of super-stars, third order super-stars out of second order super-stars and so on.

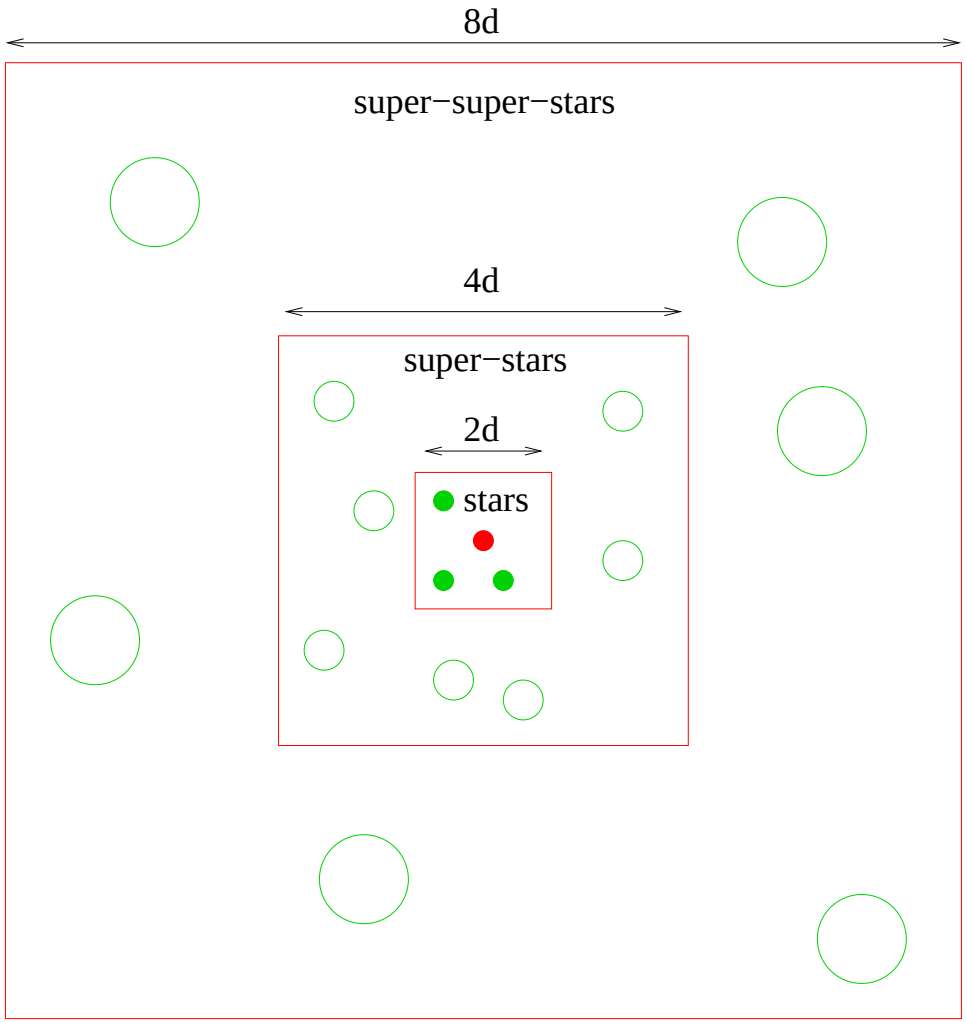
Final redundant data structure in tree form



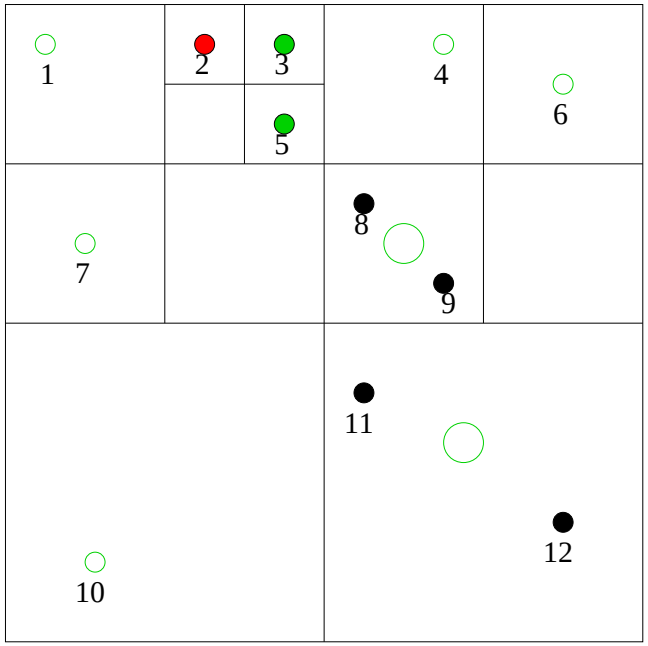
The final redundant data structure contains the positions and masses of the original stars as well as the positions and masses of all the super-stars of various order.

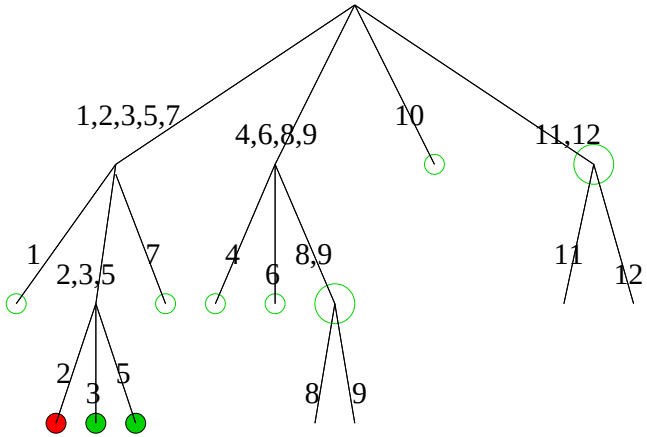
Step 3: Calculate forces

A distance d is introduced. The interactions of the red star with all the other green stars within a box of size $2d$ are calculated exactly. The interactions with the additional stars that are within a box of size $4d$ are not calculated exactly but are taken into account as interactions with the super-stars formed out of these additional stars. In the largest box shown below only the interactions of the red star with the second order super-stars are used.



Since the size of the boxes increases exponentially, there are of the order of $\log(n)$ types of interaction regions and each region contains a few super-stars of a certain order. Hence one has to sum $\log(n)$ terms to take into account the interaction of one fixed star with all the other $n-1$ stars. The total computational effort is consequently $c\,n\,\log(n)$, where c is the prefactor. The interaction to be considered for one particular (red) star of our previous example are shown below, both in real space and within the tree structure.

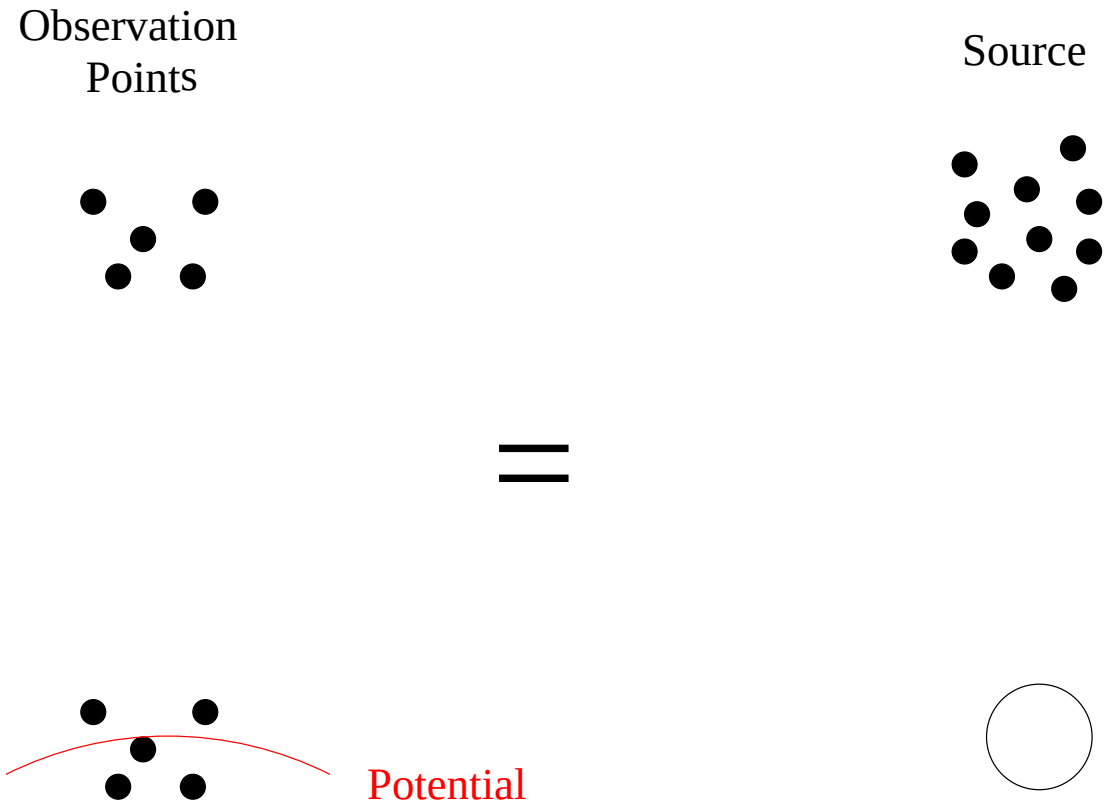




If one wants to obtain higher accuracy, one has to increase the distance d. This increases of course the prefactor c since more interactions will be calculated directly.

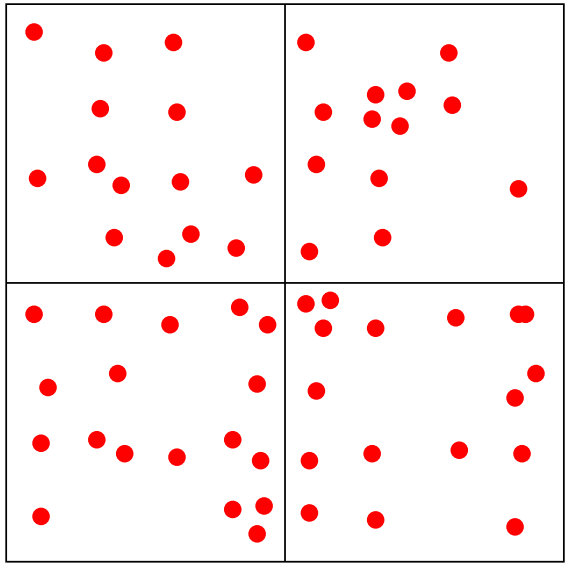
7.2 The fast multipole method

The fast multipole method (FMM) goes one step further than the Barnes Hut method. It uses not only the fact that the potential at an observation point does not depend on the details of a charge/mass source distribution far away, but also on the fact that the potential of such a charge/mass distribution is only slowly varying in the neighborhood of an observation point.

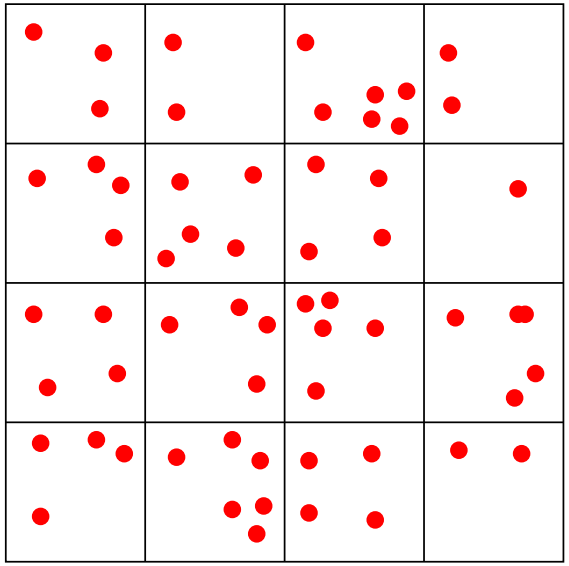


Even though both methods are hierarchical methods, there are differences. FMM is based on a hierarchy of cells that are combined into larger cells, whereas BH combines stars (particles) into super-stars, super-super-stars, etc. The FMM hierarchy is shown below. The cells at the highest (resolution) level (level 3 in the figure below) can contain more than 1 particle.

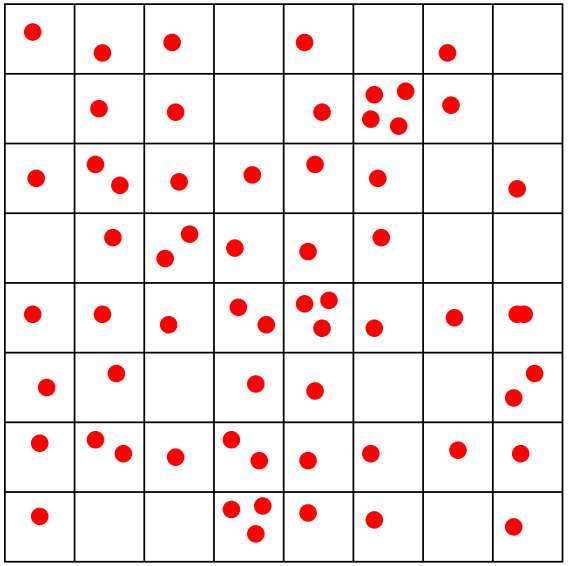
Level 1



Level 2



Level 3



FMM also uses redundant data structures. For each cell on any level of the hierarchy, the following 2 items are stored:

- The multipole coefficients $M_{l,m}$ of the charge distribution consisting of particles with charge q_i:

$$M_{l,m} = \sum_i q_i s_i^l Y_{l,m}^*(\hat{\mathbf{s}}_i) \tag{149}$$

where $\mathbf{s}$ is the position of the particle relative to the cell under consideration. How many multipoles are stored, depends on the precision that has to be achieved. The potential V is related to $M_{l,m}$ by

$$V_{multipole}(\mathbf{r}) = \sum_l \frac{4\pi}{(2l+1)\, r^{l+1}} \sum_{m=-l}^{m=l} M_{l,m}\, Y_{l,m}(\hat{\mathbf{r}}) \tag{150}$$

- The coefficients $L_{l,m}$ of the local Taylor expansion of the potential

$$V_{Taylor}(\mathbf{r}) = \sum_l \sum_{m=-l}^{m=l} r^l L_{l,m}\, Y_{l,m}(\hat{\mathbf{r}}) \tag{151}$$

The conventions of Jackson (1975) were used for the spherical harmonics and multipoles. h denotes in the following the length of the cells at a certain level of subdivision.
FMM uses the following mathematical transformations. The origin is thereby always the center of the cell.

- COMBINE: 8 multipole coefficients $M^h_{l,m}$ from a high level are combined to obtain the multipole coefficient $M^{2h}_{l,m}$ of the parent cell. This requires to first shift the multipoles into the origin of the parent cell and then to add them. The shifted multipoles $M'^h_{l,m}$ are given by

$$M'^h_{l',m'} = \sum_{l,m} T^{MM}_{l',m',l,m} M^h_{l,m} \tag{152}$$

$$T^{MM}_{l',m',l,m} = 4\pi \frac{(-t)^{l'-1}(2l'+1)}{2(l+1)(2(l'-l)+1)} \frac{Y^*_{l-l',m-m'}(\hat{\mathbf{t}})\, a'_{l'-l,m'-m}\, a_{l,m}}{a_{l',m'}} \tag{153}$$

$$a_{l,m} = (-1)^{l+m} \frac{2\sqrt{l+1}}{\sqrt{4m\,(l+m)!\,(l-m)!}} \tag{154}$$

where $\mathbf{t}$ is the translation vector between the 2 origins.

$$M^{2h}_{l,m} = \sum_{\substack{8\ multipoles \\ from\ children}} M'^h_{l',m'} \tag{155}$$

- FLIP: The coefficients $M^h_{l,m}$ are transformed into the coefficients $L^h_{l,m}$

$$L^h_{l',m'} = \sum_{l,m} T^{LM}_{l',m',l,m} M^h_{l,m} \tag{156}$$

$$T^{LM}_{l',m',l,m} = 4\pi \frac{(-1)^{l+m} Y^*_{l+l',m-m'}(\hat{\mathbf{t}}) a_{l,m} a_{l',m'}}{t^{l'+l+1}(2l+1)(2l'+l) a_{l'+l,m'-m}} \tag{157}$$

where $\mathbf{t}$ is the position of the origin of the Taylor expansion relative to the origin of the multipole.

- SHIFT: The local Taylor expansion coefficients $L^h_{l,m}$ of 8 children are generated from the $L^{2h}_{l,m}$ of the parent cell.

$$L^h_{l',m'} = \sum_{l,m} T^{LL}_{l',m',l,m} L^{2h}_{l,m} \tag{158}$$

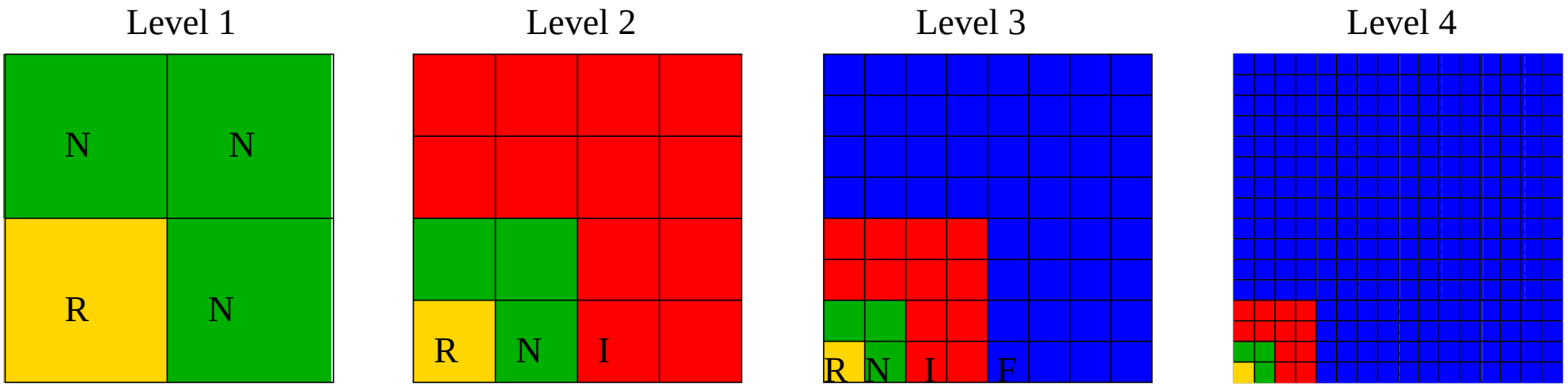
$$T^{LL}_{l',m',l,m} = 4\pi \frac{t^{l-l'} Y_{l-l',m'-m}(\hat{\mathbf{t}}) a_{l',m'} a_{l-l'',m-m'}}{(2l'+1)(2(l-l')+1) a_{l,m}} \tag{159}$$

$\mathbf{t}$ is again the translation vector between the 2 origins.

The FMM algorithm uses the following classification in the calculation of the interactions:

- A cell is near (N) to a reference cell (R) if it shares with this cell a side, edge or corner.
- A cell is interactive (I) to a reference cell (R) if both have parents that were near and if they are themselves not near
- A cell is far (F) in all other cases

The situation is illustrated below:

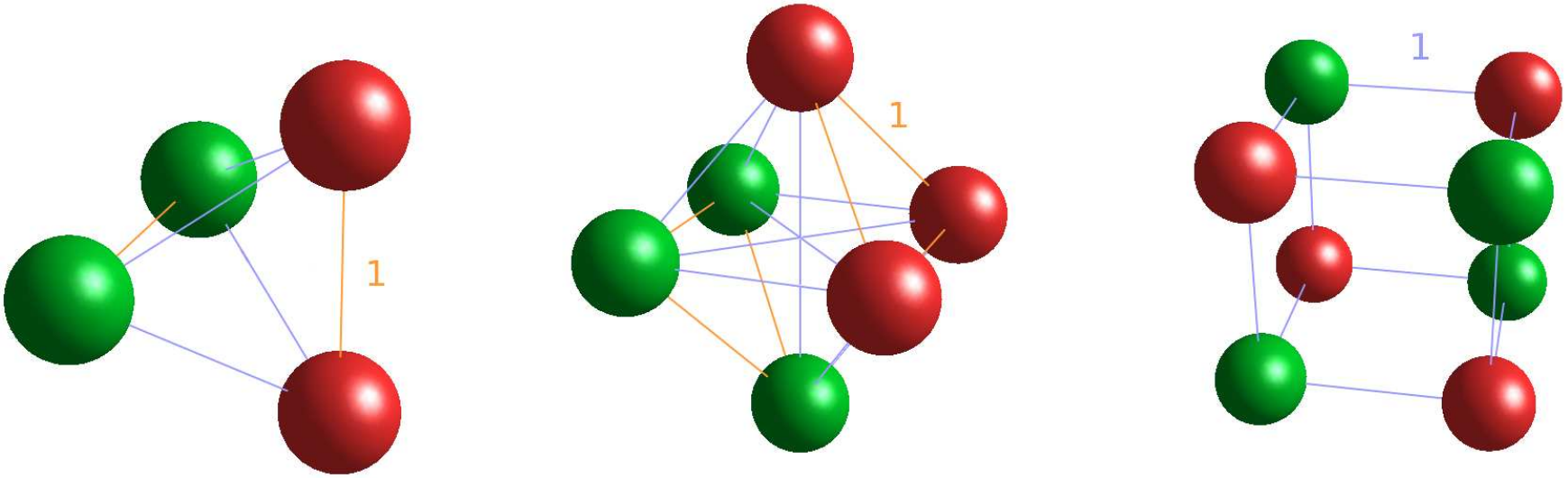


The FMM algorithm goes now as follows:

- Assign the particles to the cells at the finest level and calculate the multipole expansion coefficients $M_{l,m}$ on the finest level.
- Go from the finest level down to the coarsest level and use the COMBINE operation to generate the multipole expansion coefficients for all the bigger cells
- Go back from the coarsest level to the finest level. On each level calculate the coefficients $L_{l,m}$ that are due to interactions with INTERACTIVE cells using the FLIP operation. Use the SHIFT operation to obtain the $L_{l,m}$'s on the finer scales from the previously obtained $L_{l,m}$ on the coarser scales. The total $L_{l,m}$'s on the finest level are obtained by recursively summing both contributions from SHIFT and FLIP operations each level.
- Using the Taylor expansion, calculate on the finest level the potential/forces at the position of all the particles. This represents the potential/forces of all the particles in cells that are not NEAR cells on the finest level. The influence of the NEAR cells at the finest level is obtained by direct summation of the contributions of all the particles in the NEAR cells.

The most important workload is be done at the level of the finest cells. The work at the other levels is small compared to this one (e.g. $1/8$th at the second finest level). Hence the FMM algorithm exhibits linear scaling with respect to the number of the finest cells.

Exercise [2pt]: *Calculate the monopole, dipole and quadrupole moments for the tetrahedral, octohedral and cubical arrangements of point charges shown below. Red spheres represent a positive charge of 1 and green spheres a negative charge of -1.*



Each polyhedron is centered at the origin and has an edge length of one. A tetrahedron can be obtained by filling four non-neighboring corners of a cube, and an octrahedron can be constructed by filling the six faces. The monopole and the dipole moments for a system of point charges are defined as,

$$Q = \sum_k q_k \; ; \; \mathbf{p} = \sum_k q_k \mathbf{r}_k \tag{160}$$

where q_k and $\mathbf{r}_k$ stand for the value and position of the k^{th} point charge, respectively. The quadrupole moment tensor is given by the following equation.

$$Q_{i,j} = \sum_k q_k \left(3r_{i,k}r_{j,k} - \delta_{i,j}|\mathbf{r}_k|\right) \tag{161}$$

The indices i and j run over the x, y and z components, so we can write for example the xx− or xy−element.

$$Q_{x,x} = \sum_k q_k \left(3x_k^2 - r_k^2\right) \; ; \; Q_{x,y} = \sum_k q_k \left(3x_k y_k\right) \tag{162}$$

These moments correspond to the multipole coefficients $M_{l,m}$ of equation 149 expressed with real-valued linear combinations of spherical harmonics.
Show that the value of the dipole is independent of the choice of the origin if the monopole of the system is zero.

7.3 Analytical methods for the solution of Poisson's equation

Poisson's equation establishes the relation between a charge density ρ and its resulting potential V

$$\nabla^2 V(\mathbf{r}) = -4\pi\rho(\mathbf{r}) \tag{163}$$

For non-periodic systems such as atoms and molecules, free boundary conditions where the potential vanishes at infinity are the appropriate ones. Formally the solution can then be written as

$$V(\mathbf{r}) = \int \frac{\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}' \tag{164}$$

The numerical solution of Poissons equation is frequently based on the differential form (Eq. 163) rather than the integral form (Eq. 164).
Poisson's equation can be solved analytically in a few cases that are also relevant for numerical methods:

- The trivial case of a delta function with free boundary conditions:

$$\rho(\mathbf{r}') = \delta(\mathbf{r}') \tag{165}$$

$$V(\mathbf{r}) = \frac{1}{r} \tag{166}$$

- For a spherically Gaussian charge distribution

$$\rho(\mathbf{r}') = \frac{1}{(a\sqrt{\pi})^3}\exp(-(r'/a)^2) \tag{167}$$

the potential satisfying free boundary conditions is related to the error function erf:

$$V(\mathbf{r}) = \frac{erf(r/a)}{r} \tag{168}$$

- Under periodic boundary conditions a plane wave charge distribution

$$\rho(\mathbf{r}') = \exp(I\mathbf{K}\cdot\mathbf{r}') \tag{169}$$

gives a periodic potential

$$V(\mathbf{r}) = \frac{4\pi}{K^2}\exp(I\mathbf{K}\cdot\mathbf{r}) \tag{170}$$

Exercise [1pt]: *Show that evaluating the potential* $\frac{erf(r/a)}{r}$ *at the origin,* $r = 0$ *gives*

$$\frac{2}{a\sqrt{\pi}} \tag{171}$$

Remainder: $erf(x) = \frac{2}{\sqrt{\pi}}\int_0^x exp(-t^2)dt$

7.4 Plane wave techniques

Using the above mentioned analytical properties of plane waves, the solution of Poisson's equation under periodic boundary conditions is simple. We have to know the values of the continuous charge distribution $\rho(\mathbf{r}')$ on an equally spaced real space grid. The grid has to be dense enough such that the variation of the charge density between neighboring grid points is small. This real space data set can be transformed into Fourier space by using the fast Fourier transformation. If there are N 3-dim grid points the cost of the Fast Fourier transformation is of the order of $N\log_2(N)$. Once we have the Fourier space representation of ρ

$$\rho(\mathbf{r}) = \sum_{\mathbf{k}} c_{\mathbf{k}} \exp(I\mathbf{k}\cdot\mathbf{r}) \tag{172}$$

the Fourier space representation of the potential is

$$V(\mathbf{r}) = \sum_{\mathbf{k}} \frac{4\pi c_{\mathbf{k}}}{k^2} \exp(I\mathbf{k}\cdot\mathbf{r}) \tag{173}$$

Under periodic boundary conditions it is necessary that the system has no net charge, i.e. that $c_{\mathbf{0}} = 0$. The real space values of the potential on the grid are obtained by using a backward Fourier transformation.

7.5 Ewald techniques

The Ewald method is a standard method to calculate the energy and forces for a system of charged point particles under periodic boundary conditions. Compared to the case of free boundary conditions, we have here an additional difficulty. Because of the long range character of the forces, we would have to sum not only the interactions among particles in the same box, but also the interactions between particles in the box and the ghost particles in the periodic images of the box. As we have seen, periodic images can be handled naturally and efficiently by plane wave techniques. The problem is that a charge density that is a sum of delta functions can not be represented by plane waves. The basic idea of the Ewald technique is to introduce two charge densities that have the property to sum up to the correct charge density composed of delta functions. The first charge density $\rho_r(\mathbf{r})$ consists of a sum of localized charge densities $\rho_j^{loc}(\mathbf{r})$. Each localized charge density is the sum of the original delta function and a Gaussian charge distribution of opposite charge. This is visualized in the upper panel of the next figure. Each $\rho_j^{loc}(\mathbf{r})$ is shown by a different color. The delta function is visualized by an arrow.

$$\rho_r(\mathbf{r}) = \sum_j \rho_j^{loc}(\mathbf{r}) = \sum_j Z_j \left(\delta(\mathbf{r}-\mathbf{R}_j) - \frac{1}{(a\sqrt{\pi})^3} \exp(-(|\mathbf{r}-\mathbf{R}_j|/a)^2) \right) \tag{174}$$

Because the total charge of each localized charge term vanishes, there is no monopole and by symmetry all the higher multipoles vanish as well. Consequently the potential arising

from one $\rho_j^{loc}(\mathbf{r})$ decays exponentially. It is easy to show that the total potential is given by

$$V_r(\mathbf{r}) = \sum_j Z_j \frac{1 - erf(|\mathbf{r} - \mathbf{R}_j|/a)}{|\mathbf{r} - \mathbf{R}_j|} = \sum_j Z_j \frac{erfc(|\mathbf{r} - \mathbf{R}_j|/a)}{|\mathbf{r} - \mathbf{R}_j|} \tag{175}$$

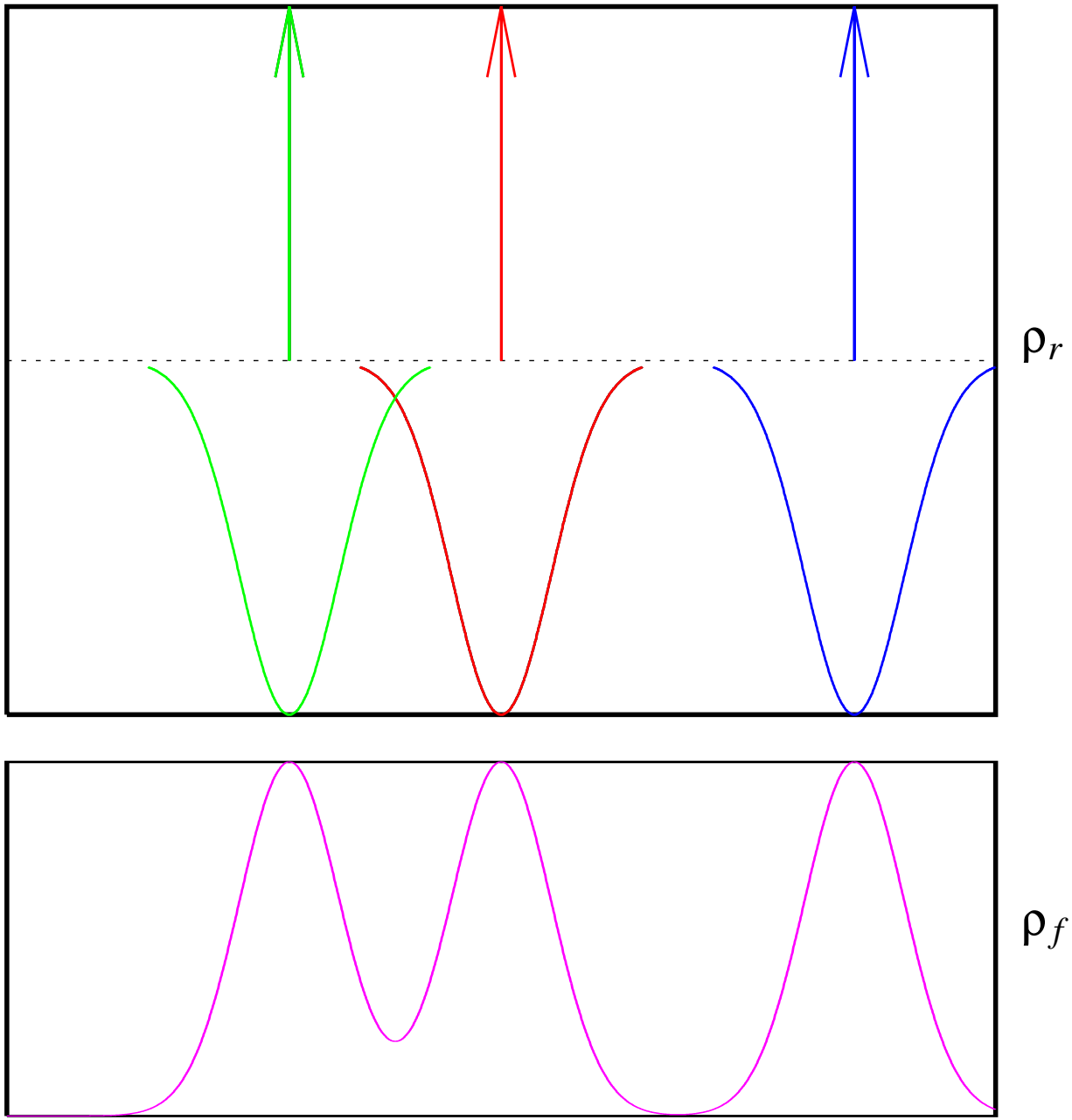
Consequently V_r can efficiently be evaluated in real space. If we want to calculate $V_r(\mathbf{r})$ at a certain point $\mathbf{r}$ we do not have to evaluate all terms j in the above sum (Eq.175), but only those that arise from localized charge distributions that are close to $\mathbf{r}$ on the scale of a. Because of the periodic boundary conditions we have to include in the sum of Eq. 175 not only terms coming from localized charge distributions within the cell, but also terms arising from close by periodic image charges.

Exercise [1pt]: *Determine numerically the distance* $|\mathbf{r} - \mathbf{R}_j|/a$ *where the potential of Eq. 175, has decayed to 1.d-14.*

The second charge density $\rho_f(\mathbf{r})$ is just the sum of the compensating Gaussians and is shown in the lower panel of the next Figure. It is a smooth function and it can therefore be represented with high accuracy by a modest number of plane waves. In addition the Fourier transformation of a Gaussian can be calculated analytically,

$$\frac{1}{Vol} \int d\mathbf{r} \frac{1}{(a\sqrt{\pi})^3} \exp(-(r/a)^2) \exp(-I\mathbf{k} \cdot \mathbf{r}) =$$

$$\frac{1}{Vol} \int_0^\infty dr \int_0^\pi d\theta \frac{2\pi}{(a\sqrt{\pi})^3} r^2 \exp(-(r/a)^2) \exp(-I\,k\,r\,cos(\theta)) = \frac{1}{Vol} \exp\left(-\left(\frac{ka}{2}\right)^2\right)$$


ρ_r
ρ_f

Hence the potential V_f arising from ρ_f is

$$V_f(\mathbf{r}) = \frac{1}{Vol} \sum_{\mathbf{k}\neq 0} \frac{4\pi}{k^2} \left(\sum_j Z_j \exp\left(I\,\mathbf{k}\cdot(\mathbf{r}-\mathbf{R}_j)\right) \right) \exp\left(-\left(\frac{ka}{2}\right)^2 \right) \tag{176}$$

We have now calculated the potential $V(\mathbf{r}) = V_r(\mathbf{r}) + V_f(\mathbf{r})$ arising from a set of point particles under periodic boundary conditions. In many applications we want however to calculate energies and forces. In this context we have to take into account that point particles are not interacting with each self, i.e the total energy is given by

$$E = \frac{1}{2} \sum_{i\neq j} \frac{Z_i Z_j}{|\mathbf{R}_i - \mathbf{R}_j|} \tag{177}$$

Evaluating the potential at all the positions $\mathbf{R}_i$ would however give

$$\frac{1}{2} \left(\sum_i Z_i (V_r(\mathbf{R}_i) + V_f(\mathbf{R}_i)) \right) = \frac{1}{2} \sum_{i,j} \frac{Z_i Z_j}{|\mathbf{R}_i - \mathbf{R}_j|} \tag{178}$$

Taking out the self-interaction in the real space part (Eq. 175) is simple and one obtains

$$E_r = \frac{1}{2} \sum_j \sum_{i\neq j} Z_i Z_j \frac{erfc(|\mathbf{R}_i - \mathbf{R}_j|/a)}{|\mathbf{R}_i - \mathbf{R}_j|} \tag{179}$$

The sum over j in Eq. 179 runs over all localized charges in the system, the sum over i over charges interacting with charge j. Remember that these interacting charges can be periodic image charges. Taking the self-interaction out of the Fourier space part (Eq. 176) is also not very difficult. We have just to subtract the potential of a Gaussian charge distribution evaluated at the origin (cf. Eq. 171). The Fourier space contribution is thus given by

$$\begin{aligned} E_f &= \frac{1}{2}\frac{1}{Vol}\sum_{\mathbf{k}\neq 0}\frac{4\pi}{k^2}\left(\sum_{i,j} Z_i Z_j \exp\left(I\,\mathbf{k}\cdot(\mathbf{R}_i-\mathbf{R}_j)\right)\right)\exp\left(-\left(\frac{ka}{2}\right)^2\right)-\frac{1}{a\sqrt{\pi}}\sum_i Z_i^2 \\ &= \frac{1}{2}\frac{1}{Vol}\sum_{\mathbf{k}\neq 0}\frac{4\pi}{k^2}|S(\mathbf{k})|^2\exp\left(-\left(\frac{ka}{2}\right)^2\right)-\frac{1}{a\sqrt{\pi}}\sum_i Z_i^2 \end{aligned} \tag{180}$$

Where $S(\mathbf{k})$ is the structure factor $S(\mathbf{k})=\sum_i Z_i \exp(-I\,\mathbf{k}\cdot(\mathbf{R}_i))$. The Ewald method for the energy consists thus of calculating the energy $E=E_r+E_f$ of Eq. 177 as a sum of a real space term E_r (Eq. 179) and a Fourier space term E_f (Eq. 180). A similar formula can be derived for the forces.

Let us now discuss the scaling properties of the Ewald method with respect to the number of charges N in the system and the relation to the choice of the width a of the Gaussians. Let us assume that the average number of charges per unit volume $\rho_c = N/Vol$ remains constant while we are increasing the number of charged point particles. The real space

part E_r has a linear scaling. For each charge we have to calculate all the interactions with charges that are localized within a few a's. Consequently there are of the order of $\rho_c a^3$ such charges and the total scaling is $N\rho_c a^3$. In the Fourier space part, E_f, the number of plane waves we have to include for an accurate representation of the Gaussian is proportional to $Vol/a^3 = N/(\rho_c a^3)$. For each plane wave $\mathbf{k}$ we have to calculate the structure factor $S(\mathbf{k})$. Hence the overall scaling is $N^2/(\rho_c a^3)$. If we enlarge N without modifying a, the Fourier space sum of E_f will finally dominate the computational cost because of its N^2 scaling. We can however enlarge a as we increase N. If $a^3 \propto \sqrt{N}$ then the computational cost for both E_f and E_r grows as $N^{3/2}$.

The Ewald technique achieves thus two remarkable things. It incorporates at virtually no extra cost all the interactions with image charges under periodic boundary conditions. In addition it allows us to evaluate the energy sum of Eq. 177 with a scaling that is less than the trivial N^2 scaling. Even though the scaling is not quite as good as in other fast methods the Ewald method has a small prefactor and beats usually other methods for small and medium size systems.

7.6 Particle-Particle Particle-Mesh (P^3M) methods

The P^3M is another method to calculate the electrostatic energy (Eq. 177) of a system of charged particles under periodic boundary conditions. As in the Ewald method the charge is represented as the sum of a smooth part and localized charges. The expressions for this later part, that is treated in real space, are as a matter of fact identical to the expressions used in the Ewald method. What differs is how one treats the Fourier space potential of Eq. 176. This part is not calculated analytically but numerically in the following way. The smooth charge distribution ρ_f is evaluated on a sufficiently dense grid. Then the potential V_f is calculated on the same grid by using the plane wave techniques described previously. Finally the potential at the positions of the point charges $\mathbf{R}_j$ is calculated by interpolation methods from the values on the grid. Consequently E_f is given by

$$E_f = \frac{1}{2}\sum_i Z_i V_f(\mathbf{R}_i) - \frac{1}{a\sqrt{\pi}}\sum_i Z_i^2 \tag{181}$$

As in the Ewald method the last term is needed to cancel self-interactions.
Compared to the Ewald method the scaling of the Fourier space part is more favorable: $N\log_2(N)/(\rho_c a^3)$ instead of $N^2/(\rho_c a^3)$. Because of this favorable scaling of the Fourier space part, it is not necessary to enlarge a as the system grows larger and the overall scaling is $N\log_2(N)$. The prefactor is however larger than in the Ewald method.

7.7 Multigrid for the solution of Poisson's equation

Multigrid is the standard method for solving Poisson's equation in a finite difference scheme. For simplicity we will only consider the 1-dim Poisson equation

$$\frac{\partial^2}{\partial x^2} V(x) = -4\pi\rho(x) \tag{182}$$

Using the simplest discretization of the second derivative Eq. 182 becomes

$$V_{i+1} - 2V_i + V_{i-1} = -4\pi h^2 \rho_i \tag{183}$$

Eq. 183 is a linear system of equations

$$Ax = y \quad ; \quad x = V\,,\, y = \rho \tag{184}$$

Under periodic boundary conditions, the matrix A is given by

$$\begin{pmatrix}
-2 & 1 & & & & & & & 1 \\
1 & -2 & 1 & & & & & & \\
 & 1 & -2 & 1 & & & & & \\
 & & \ddots & & & & & & \\
 & & & \ddots & & & & & \\
 & & & & \ddots & & & & \\
 & & & & 1 & -2 & 1 & & \\
 & & & & & 1 & -2 & 1 & \\
1 & & & & & & 1 & -2 &
\end{pmatrix}$$

The solution of a linear system of equations is equivalent to finding the stationary points of

$$x^T(\frac{1}{2}Ax - y) \tag{185}$$

as can easily be seen by taking the partial derivatives of Eq 185 with respect to all the components of x. Differentiating once more gives the Hessian matrix which turns out to be A. It can be shown that the conditioning number grows quadratically as a function of the lateral dimensions of the system divided by the grid resolution. Hence either very large systems or very high grid resolutions lead to conditioning problems.

The elementary local iterative solution methods for Eq. 183 are

- The steepest descent method discussed previously

$$g_i = V_{i+1} - 2V_i + V_{i-1} + 4\pi h^2 \rho_i \tag{186}$$

$$V_i = V_i + \alpha g_i \tag{187}$$

Exercise [2pt]: *Find a discrete variational quantity (a function of all the V_i's) that has the property that if one zeroes its gradient with respect to the V_i's, one obtains Eq. 183. Hint: The functional derivative of $\int(\frac{1}{2}(\frac{\partial V}{\partial x})^2 - 4\pi V(x)\rho(x))dx$ gives the continuous Poisson equation 182.*

- The Jacobi relaxation (which is equivalent to a steepest descent with $\alpha = 1/2$).

$$\tilde{V}_i = \frac{1}{2}(V_{i+1} + V_{i-1}) + 2\pi h^2 \rho_i \tag{188}$$

$$V_i = \tilde{V}_i \tag{189}$$

- The Gauss-Seidel relaxation

$$V_i = \frac{1}{2}(V_{i+1} + V_{i-1}) + 2\pi h^2 \rho_i \tag{190}$$

- Red-black Gauss-Seidel relaxation

$$V_i = \frac{1}{2}(V_{i+1} + V_{i-1}) + 2\pi h^2 \rho_i \qquad \text{for all even } i \tag{191}$$

$$V_i = \frac{1}{2}(V_{i+1} + V_{i-1}) + 2\pi h^2 \rho_i \qquad \text{for all odd } i \tag{192}$$

The three relaxation methods were written down in programming style, meaning that each equation corresponds to a loop. Hence one works on all the indices of the first equation before proceeding to the next. Thus the difference between Jacobi, Gauss-Seidel and red-black Gauss-Seidel is a very subtle one. In the Jacobi method the solution is updated only

after all the old values were used for the calculation of $\tilde{V}$. In the Gauss-Seidel method the new values are immediately used for the calculation of further new values. Hence relaxations where the index is running from 1 to n are not equivalent to relaxations where the index is running from n to 1, even though one relaxation type is in generally not better than the other. In the red-black Gauss-Seidel all the even points are first updated using the information of the odd points and then the odd point are updated using the information of the already updated even points. The red-black Gauss-Seidel iteration is usually the most efficient one for reducing error components that have a wavelength that is comparable to the grid spacing. All the methods fail however badly for error components that have a wavelength much larger than the grid spacing. The failure of the steepest descent method was studied before. The failure of the relaxation methods is easy to understand. For small grid spacings h the iterations have all the form

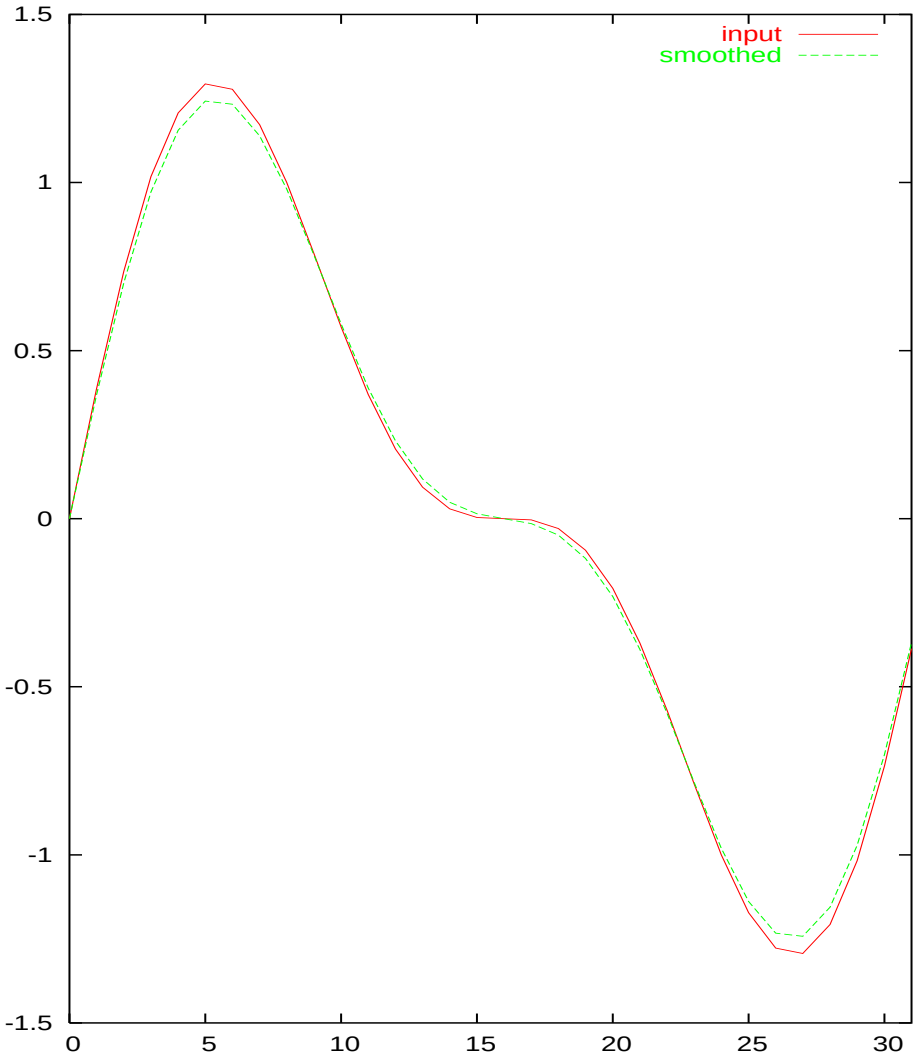
$$V_i = \frac{1}{2}(V_{i+1} + V_{i-1}) \tag{193}$$

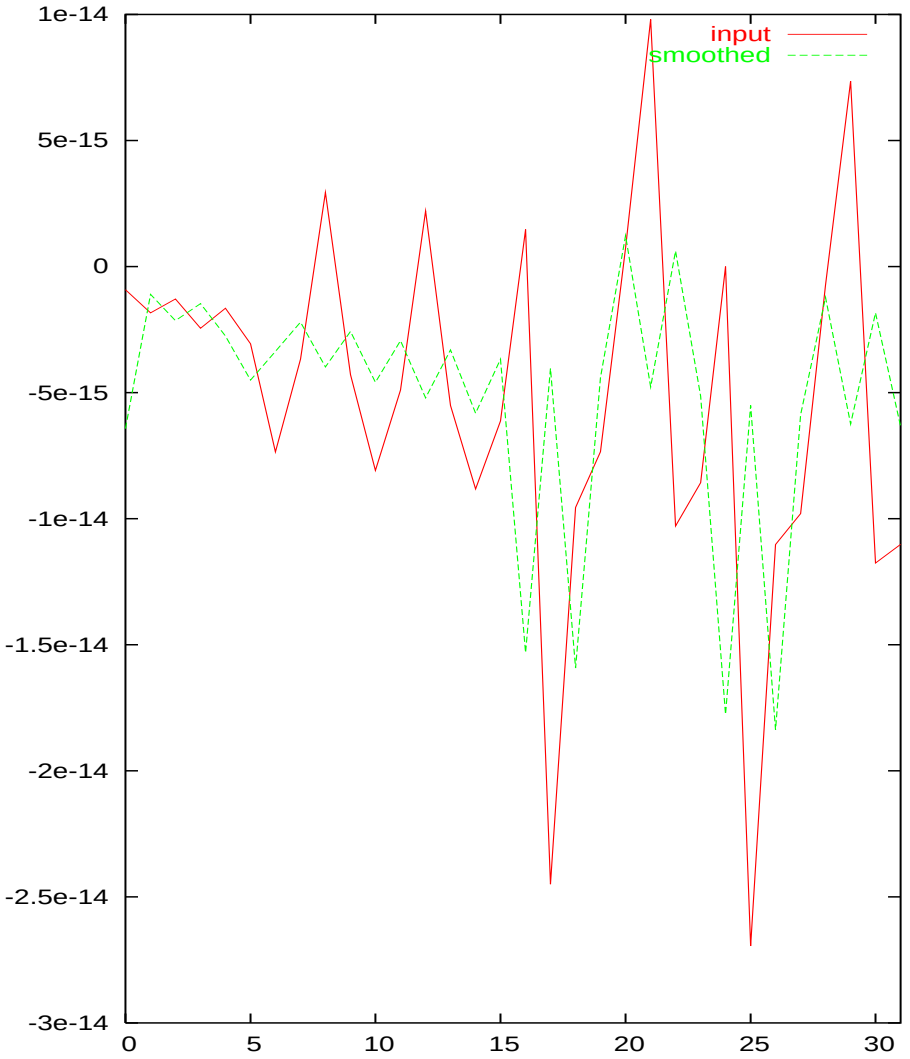
respectively

$$\tilde{V}_i = \frac{1}{2}(V_{i+1} + V_{i-1}) \tag{194}$$

The value in the center is the average of the values to the right and to the left. On small scales any smooth function is to first order a linear function and taking the average does not modify a linear function. Hence the convergence becomes very slow for small h. The

effect is demonstrated below on a grid of 32 grid points. The red curve represents the original data V_i and the green curve the smoothed data set $\tilde{V}_i = \frac{1}{2}(V_{i+1} + V_{i-1})$. For data that are slowly varying over the grid spacing h the smoothed curve is very similar to the input curve, where as for a non-smooth curve both curves differ significantly.





What happens therefore if a relaxation (or steepest descent) method is used to solve Poisson's equation with a small h is the following. During the first iteration the convergence is fast, since the short wavelength components, of the error are eliminated. These short wavelength components have a wavelength $\lambda \approx 2h$. As soon as the remaining long wavelength components have to be eliminated the convergence becomes extremely slow. The basic idea of multigrid is to eliminate these longer wavelength error components on a hierarchy of grids whose spacing h matches the wavelengths under attack. The fundamental equation that allows for such an approach is the error correction equation. If we have an approximate solution V then the difference ΔV between the exact solution and the approximate solution V fulfills as well a Poisson equation

$$\nabla^2 \Delta V(\mathbf{r}) = -\nabla^2 V - 4\pi\rho(\mathbf{r}) \tag{195}$$

since the right hand side, which is called the residue, can again be considered as some charge density. Using these two ingredients we obtain the 2-grid algorithm:

1. Do a few (2 to 5) relaxations on the original grid. This will give an approximate solution V_i^h and a residue

$$R_i^h = -\frac{1}{h^2}\left(V_{i+1}^h - 2V_i^h + V_{i-1}^h\right) - 4\pi\rho_i \tag{196}$$

2. Transfer the residue R_i^h to a grid with double spacing $2h$ and let's call this residue

R_i^{2h}. Since we have to cut the number of data into half, the simplest way is just to select every second data point, i.e $R_i^{2h} = R_{2i}^h$ This transfer from a fine to a coarse grid is called restriction.

3. Find the solution on the coarse grid for a Poisson equation where the charge density is given by R_i^{2h}, i.e. solve

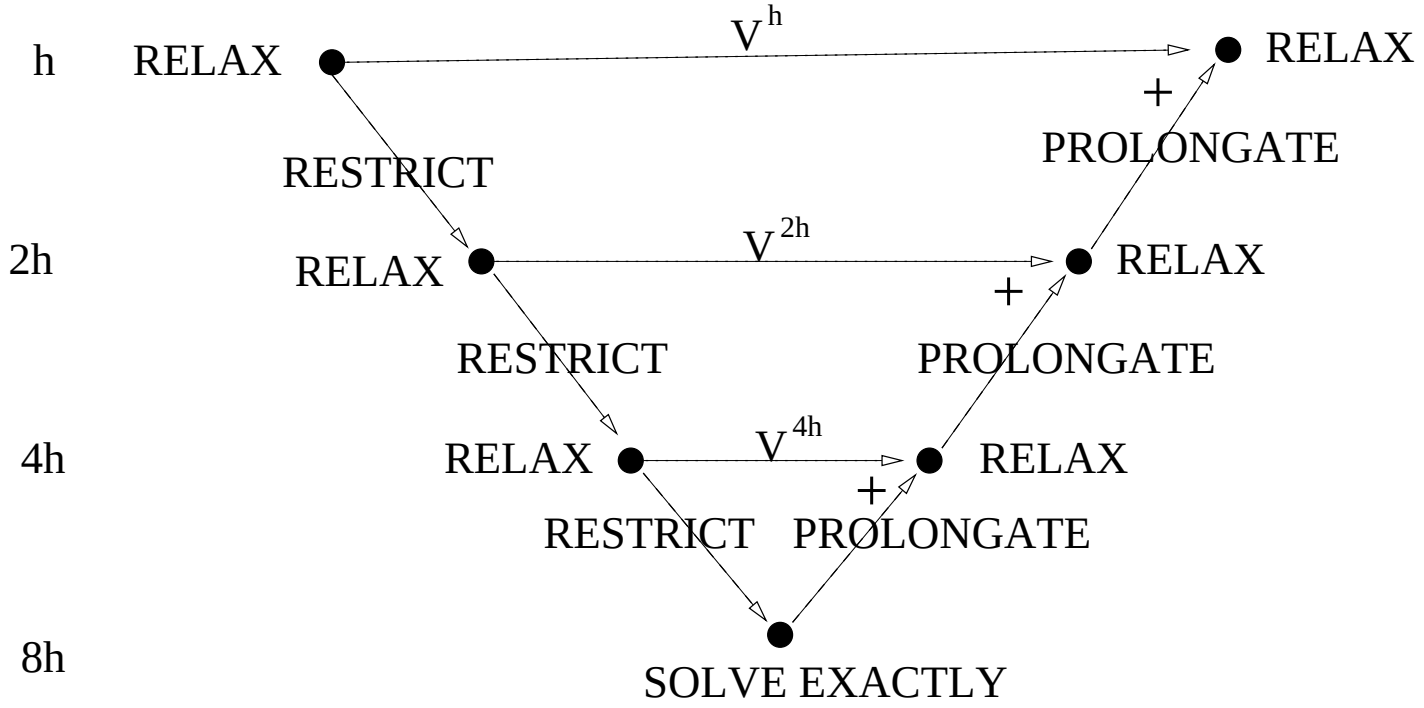
$$V_{i+1}^{2h} - 2V_i^{2h} + V_{i-1}^{2h} = (2\,h)^2 R_i^{2h} \tag{197}$$

V_i^{2h} represents the correction $\Delta V(\mathbf{r})$ of Eq. 195.

4. Bring V_i^{2h} back onto the fine grid and add it to V_i^h The process of calculating a quantity that was defined on a coarse grid on a fine grid is called prolongation. The simplest prolongation scheme is to use on all the even grid points of the fine grid the values on the coarse grid and to generate the values on the odd grid points by interpolation from the neighboring even grid points.

5. Unfortunately the potential that we have now obtained by adding V_i^h and the prolongation of V_i^{2h} on the fine grid is not yet the final solution. the reason for this is that the prolongation step introduces some small short wavelength errors. These errors can be rapidly eliminated by a few (2 or 3) additional relaxation steps on the fine grid.

We left it open how to find the solution V_i^{2h} on the coarse grid. Obviously we can again use exactly the same procedure that we used for the fine grid by going to an grid with an even larger grid spacing of $4h$. Doing this recursively until we arrive at a grid with a very small number of grid points, where the problem can easily be solved exactly, results in the multigrid V cycle illustrated below.



Exercise [1pt]: *What is the complexity (i.e. the scaling of the numerical effort with respect to the number of grid points) of the multigrid V cycle if we neglect the cost of solving the equation exactly at some very coarse grid level and if we use a fixed number of relaxation on each grid level?*

PROJECT: A 1-dim multigrid program

The multigrid method can be applied to problems in one, two, three and more dimensions. Here we will study its behaviour for the simplest 1-dim case with periodic boundary conditions. For simplicity the length of the periodic volume can be taken to be one and the charge density ρ is a sine function $\rho(x) = sin(x2\pi)$. Find the analytical solution for the potential. Is the potential unique? Next, solve the problem numerically using the multigrid method

- Discretize the interval with 256 grid points.
- Take random numbers as your initial guess for the potential.
- Write a subroutine that performs Gaus-Seidel relaxations and another subroutine that calculates the residue(Eq. 196). Never set up the matrix of Eq. 184 explicitly to calculate these quantities.
- Do a few Gaus-Seidel relaxations on the 256 grid. Monitor graphically how the convergence rate slows down and calculate the residue after each Gaus-Seidel relaxation.
- Implement next a two grid method. Bring the residue from the 256 point grid to a 128 point grid by the restriction procedure described in the lecture notes and

do Gaus-Seidel relaxation on the 128 grid to solve Equation 195 approximately. Observe again how the convergence rate slows down when you are doing Gaus-Seidel relaxations on the 128 point grid. Once the convergence rate saturates, add the correction to the potential calculated on the 128 grid to the solution on the 256 grid. Before adding the two quantities on the fine grid use the prolongation procedure described in the lecture notes with a simple linear interpolation to bring the correction to the fine grid. Do 5 more Gaus-Seidel relaxations on the 256 grid and look by how much the residue was reduced by this excursion to the 128 grid.

- Replace the Gaus-Seidel relaxation on each grid level by a red-black Gauss-Seidel relaxation and and replace the simple restriction scheme by the full weightening scheme where

$$R_i^{2h} = \frac{1}{4}R_{2i-1}^h + \frac{1}{2}R_{2i}^h + \frac{1}{4}R_{2i+1}^h \tag{198}$$

- Now implement a true multigrid method where you have on the coarsest grid level only 4 grid points. A major difficulty is to find convenient data structures that hold the various quantities on the different grids. Since in the one dimensional case memory is not problematic you can choose some simple but wasteful data structures (where for instance you use the same amount of memory for each grid level). You are however also invited to design some more efficient data structures.

- Determine how many multigrid cycles you need to reduce the norm of the residue by 6 orders of magnitude both for the combination simple-restriction/Gaus-Seidel and full-weightening/red-black-Gaus-Seidel. How many iterations would you need to obtain the same error reduction if you performed only Gaus-Seidel relaxations on the 256 grid?

- Even if the procedure has converged (i.e. if the residue is very small), you have only an approximation to the analytical solution. This comes from the fact that the second derivative operator was replaced by a finite difference formula. Determine the difference between your numerical solution and the analytical solution.

PROJECT: Finite Difference Method for solving Laplace Equation

Introduction

The electric potential V in the absence of any charge, satisfies the Laplace equation. In a two-dimensional problem, the corresponding two-dimensional Laplace equation is

$$\nabla^2 V(x,y) \equiv \frac{\partial^2 V(x,y)}{\partial x^2} + \frac{\partial^2 V(x,y)}{\partial y^2} = 0 \tag{199}$$

for all real (x,y) in region S of a plane. First, consider a region S that composed of all the points (x,y) so that $0 \le x \le a$ and $0 \le y \le b$. The boundary condition for this problem is chosen to be a Dirichlet boundary condition, e.g., for all the points (x,y) on the boundary $C \equiv \partial S$ of S, one has

$$\begin{cases} V(x,0) = V^0, \\ V(x,b) = V(0,y) = V(a,y) = 0. \end{cases} \tag{200}$$

For simplicity, we choose $a = 1$ and $V^0 = 1$, so the analytic solution as $b \to \infty$ is

$$V(x,y) = \frac{2}{\pi} \tan^{-1}\left(\frac{\sin \pi x}{\sinh \pi y}\right) \tag{201}$$

Next, we limit the region S so that $1 \le y \le 2$, the solution (201) remains unchanged while

the corresponding boundary condition on C, the boundary of S, is given by

$$\begin{cases} V(x,1) = \frac{2}{\pi}\tan^{-1}\left(\frac{\sin \pi x}{\sinh \pi}\right), \\ V(x,2) = \frac{2}{\pi}\tan^{-1}\left(\frac{\sin \pi x}{\sinh 2\pi}\right), \\ V(0,y) = V(1,y) = 0. \end{cases} \tag{202}$$

The purpose of this project is to numerically solve the equation (199) on the region S so that $0 \leq x \leq 1$ and $1 \leq y \leq 2$ with the Dirichlet boundary condition (202). The numerical solution is then compared to the exact solution (201). An analysis for the error of the numerical solution is also requested.

To solve equation 199 numerically, one first discretes it using the finite difference method. One covers S with an equidistant grid with grid spacing h along both sides of S. In other words, each of the intervals $[0,1]$ (for x) and $[1,2]$ (for y) is divided into N sub-intervals with the length

$$h = \frac{1}{N}. \tag{203}$$

The potential $V(x,y)$ is then calculated on the grid points using the relaxation method. The potential $V(x,y)$ at a grid point with coordinates $x = ih$ and $y = jh$, is referred to as $V_{i,j}$. In the relaxation method as mentioned for one-dimensional problem in the lecture notes, the

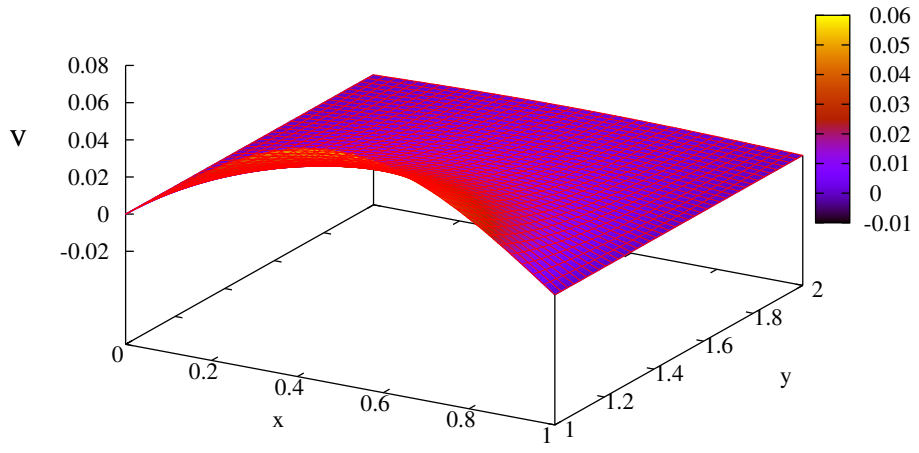


Figure 2: Exact solution of the equation (199) on the region $S : x \in [0,1], y \in [1,2]$

potential at each grid point is obtained from the average $\widetilde{V}_{i,j}$ of the potential values over the corresponding neighbors. To this aim the ordinary Gauss-Seidel relaxation

$$V_{i,j} = \widetilde{V}_{i,j} \tag{204}$$

(or the red-black Gauss-Seidel) is iterated until the potential at each grid point converges to some value. Using more neighbors in the averaging $\widetilde{V}_{i,j}$ procedure, gives more accurate results. In this project we will use three different types of averages. To speedup the

convergence, one can replace the (red-black) Gauss-Seidel relaxation with the successive over-relaxation

$$V_{i,j} = \omega \widetilde{V}_{i,j} + (1-\omega) V_{i,j} \tag{205}$$

with a relaxation factor of $\omega = 2/(1+\sin \pi h)$ (note that the Gauss-Seidel relaxation (204) corresponds to $\omega = 1$).
Finally, the numerical solution obtained will be compared with the exact solution (201) by analyzing the norm of its error defined by

$$||e|| = \sqrt{\frac{1}{N^2} \sum_{i,j} \left[V_{i,j}^{(\text{calculated})} - V_{i,j}^{(\text{exact})} \right]^2}. \tag{206}$$

Detailed tasks

1. Show, using a Taylor expansion, that we can discretize the Laplace equation (199) on the grid to obtain the approximate potential

   $$V_{i,j} = \widetilde{V}_{i,j} + O(h^4)$$

   where the average is obtained from the 'cross' average

   $$\widetilde{V}_{i,j} = V_{i,j}^{+} = \frac{1}{4} \left(V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1} \right). \tag{207}$$

2. Write your code to do successive over relaxation using the 'cross' average with random initial guess to calculate the potential V inside S. Do the error analysis, i.e., using the formula (206), plot on a log-log scale $||e||$ as a function of h and find the dependence of $||e||$ on h. Note that this *global* error is two orders of magnitude smaller than the *l*ocal error obtained from the Taylor expansion in which the exact values of the function at the adjoining sites are known.

3. Repeat the steps 1 and 2 with the 'corner' average

$$\widetilde{V}_{i,j} = V^{\times}_{i,j} = \frac{1}{4}\left(V_{i+1,j+1} + V_{i-1,j+1} + V_{i+1,j-1} + V_{i-1,j-1}\right) \tag{208}$$

and compare the error $||e||$ of the method using the 'corner' formula (208) with that using the 'cross' formula (207).

4. We now use a certain linear combination of two averages above

$$\widetilde{V}_{i,j} = V^{\diamond}_{i,j} = \frac{4}{5}V^{+}_{i,j} + \frac{1}{5}V^{\times}_{i,j}. \tag{209}$$

Modify your code to implement the combination (209), calculate the error $||e||$, and show (quantitatively) that this combination improves the accuracy of the numerical calculations, both by a Taylor expansion method and by actual numerical calculations.

5. *Effects of the finite difference approximation for the boundary C.* The boundary C used in the above problem can be exactly represented by the 'square' grid chosen. In a problem with a different geometry of C, for example, a circle, there are errors coming from the fact that C can *not* be exactly represented by the 'square' grid so one needs another grid.

   Consider the electric potential V within two concentric conducting cylinders with radii a and b $(a < b)$ and very large lengths. The potential V is kept to be V_1 on the inner cylinder and V_2 on the outer cylinder. Due to the translation symmetry along the axis of the two cylinders, chosen to be the z axis, the problem can be considered as a two-dimensional problem on the xy plane. For $a = 1/2, b = 1$ and $V_1 = 1, V_2 = 0$, the exact solution of the potential is then

$$V(x,y) = -\frac{1}{\ln 2} \ln \sqrt{x^2+y^2}. \tag{210}$$

   For this problem, you are asked to

   - Change the boundary condition to reflect the new geometry of the problem and do the relaxation for the sites in the region $1/2 < \sqrt{x^2+y^2} < 1$. Note that with the symmetry of the geometry, you can relax only one quarter of the coordinate plane with appropriate treatment for $x = 0$ and $y = 0$. Do the error

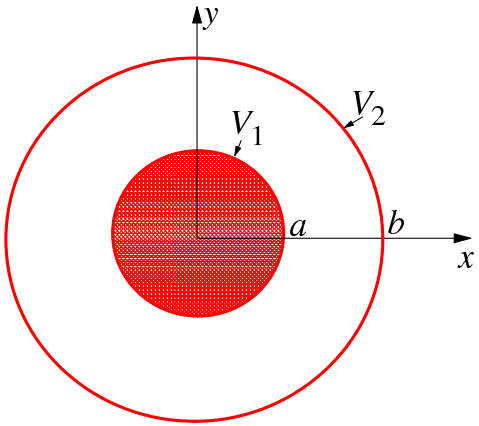


Figure 3: The cross section of the two cylinders, which is represented on the *xy* plane.

analysis for three different averages ('corner', 'cross', and 'combination'). Explain the result for the error obtained.

- Discrete the Laplace equation in *polar coordinates* system and solve the problem numerically in 1D in such a way that the edges are represented exactly. Determine the errors obtained theoretically and numerically.

7.8 Solution of Poisson's equation in spherical coordinates

In spherical coordinates, used typically for electronic structure calculations of atoms, the solution of Poissons equation is easy. If the charge density is given in terms of radial functions times spherical harmonics,

$$\rho(\mathbf{r}') = \sum_{l,m} \rho_{l,m}(r') Y_{l,m}(\hat{\mathbf{r}}') \tag{211}$$

it follows from the addition theorem of spherically harmonics,

$$\frac{1}{|\mathbf{r}-\mathbf{r}'|} = 4\pi \sum_{l,m} \frac{1}{2l+1} \frac{r_<^l}{r_>^{l+1}} Y_{l,m}^*(\hat{\mathbf{r}}') Y_{l,m}(\hat{\mathbf{r}}') \tag{212}$$

that the potential is given by

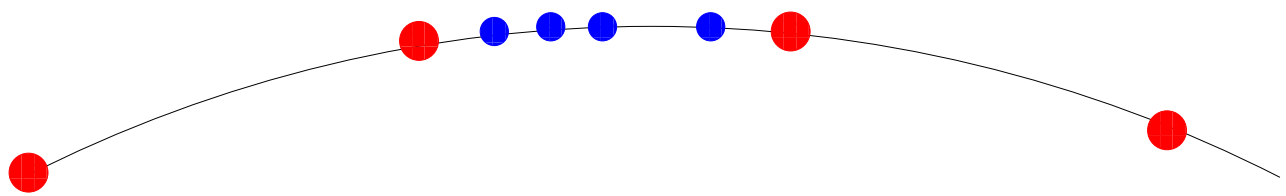
$$V(\mathbf{r}) = \sum_{l,m} \frac{4\pi}{2l+1} Y_{l,m}(\hat{\mathbf{r}}) \left(\frac{1}{r^{l+1}} \int_0^r \rho_{l,m}(r') r'^{l+2} dr' + r^l \int_r^\infty \rho_{l,m}(r') \frac{1}{r'^{l-1}} dr' \right) \tag{213}$$

This means that the potential can be obtained by simple outward (from 0 to r) and inward (from ∞ to r) radial integrations.

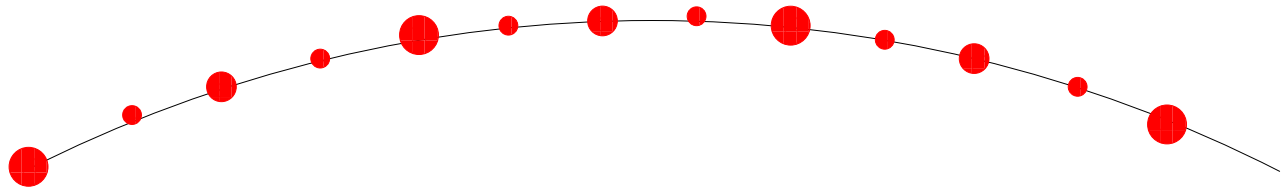
Exercise [1pt]: *Use Gauss's law of electrostatics to verify Eq. 213 for the case $l = 0$.*

7.9 Standard non-recursive and recursive interpolation

Standard Interpolation



Recursive Interpolation



7.10 Solution of Poisson's equation using interpolating scaling functions

The previous method was only applicable if the angular part of the charge density can be represented by a reasonably small number of spherical harmonics. What is still missing is a method that can solve Poisson's equation with free boundary conditions for arbitrary charge densities. Remember that free boundary conditions are obtained if one solves the integral equation Eq. 164.

Let us first discuss the construction of interpolating scaling functions. This construction is closely connected to the question of how to construct a continuous function $f(x)$ if only its values f_i on a finite number of grid points i are known. One way to do this is by recursive interpolation. Given a data set that is defined on all integer points, we first interpolate the functional values on all the midpoints by using for instance the functional values of two integer grid points to the right and of two integer grid points to the left of the midpoint. Four functional values allow us to construct a third order polynomial and we can then evaluate it at the half integer midpoint. In the next step, we take this new data set of functional values at integer and half integer points, which is now twice as large as the original one, as the input for a new midpoint interpolation procedure. This can be done recursively ad infinitum until we have a quasi continuous function.

For linear interpolation the formula for the functional value $f_{1/2}$ in the midlle between

grid points f_0 and f_1 reads

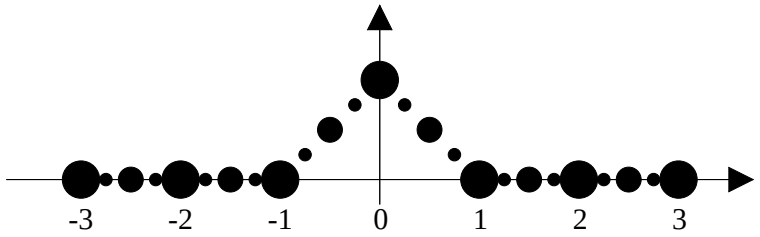
$$f_{1/2} = \frac{1}{2}(f_0 + f_1) \tag{214}$$

and for third order interpolation it reads

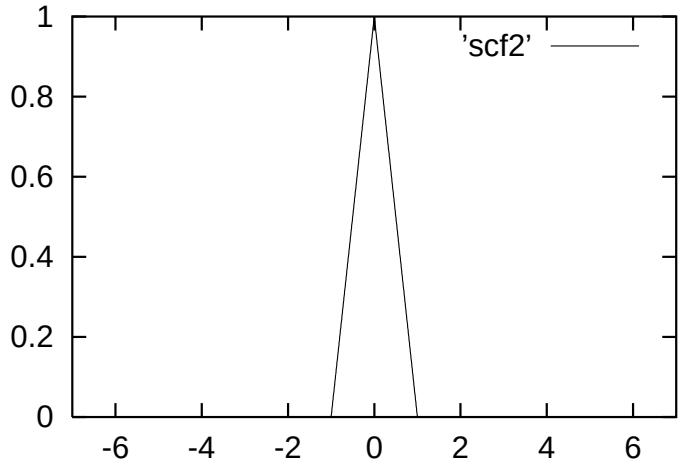
$$f_{1/2} = \frac{1}{16}(-f_{-1} + 9f_0 + 9f_1 - f_2) \tag{215}$$

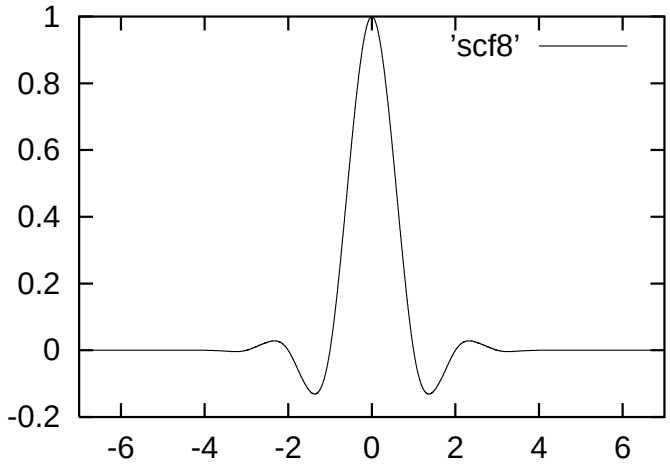
Let us now show, how this interpolation prescription leads to a set of basis functions. Denoting by the Kronecker δ_{i-j} a data set whose elements are all zero with the exception of the element at position j, we can write any initial data set as a linear combination of such Kronecker data sets: $f_i = \sum_j f_j \delta_{i-j}$. Now the whole interpolation procedure is clearly linear, i.e. the sum of two interpolated values of two separate data sets is equal to the interpolated value of the sum of these two data sets. This means that we can instead also take all the Kronecker data sets as the input for separate ad-infinitum interpolation procedures, to obtain a set of functions $\phi(x-j)$. The final interpolated function is then identical to $f(x) = \sum_j f_j \phi(x-j)$. If the initial grid values f_i were the functional values of a polynomial of degree less than four, we obviously will have exactly reconstructed the original function from its values on the grid points. Since any smooth function can locally be well approximated by a polynomial, these scaling functions $\phi(x)$ are good basis functions.

The first construction steps of an interpolating scaling function are shown in Figure below for the case of linear interpolation. The initial Kronecker data set is denoted by the big dots. The additional data points obtained after the first interpolation step are denoted by medium size dots and the additional data points obtained after the second step by small dots.



Continuing this process ad infinitum will then result in the function shown in the left panel of Figure below. If a 7-th order order interpolation scheme is used the function shown in the right panel of Figure below is obtained.





Exercise [1pt]: *Plot the scaling function obtained by third order interpolation (Eq. 215).*

By construction it is clear, that $\phi(x)$ has compact support. If an $(m-1)$-th order interpolation scheme is used, the support interval of the scaling function is $[-(m-1);(m-1)]$. The important property of the interpolating scaling functions is that they vanish at all integer arguments except at the origin, i.e for all integers i

$$\phi(i) = \delta_i \tag{216}$$

If we have now the values of a charge density $\rho i1, i2, i3$ on a 3-dimensional grid with grid spacing h we can very easily construct a continuous charge density $\rho(\mathbf{r})$

$$\rho(\mathbf{r}) = \sum_{i1,i2,i3} \rho_{i1,i2,i3}\, \phi(x/h - i1)\, \phi(y/h - i2)\, \phi(z/h - i3) \tag{217}$$

For simplicity we will in the following set the grid spacing equal to 1.
Exercise [2pt]: *Show that the discrete and continous monopoles and dipoles are identical, i.e*

$$\int_{-\infty}^{\infty} dx \int_{-\infty}^{\infty} dy \int_{-\infty}^{\infty} dz\, \rho(\mathbf{r}) = \sum_{i1,i2,i3} \rho_{i1,i2,i3} \tag{218}$$

$$\int_{-\infty}^{\infty} dx \int_{-\infty}^{\infty} dy \int_{-\infty}^{\infty} dz\, z\, \rho(\mathbf{r}) = \sum_{i1,i2,i3} i3\, \rho_{i1,i2,i3} \tag{219}$$

if the relation between $\rho(\mathbf{r})$ *and* $\rho_{i1,i2,i3}$ *is given by Eq. 217. Hint: Derive first* $\int \phi(x)dx = 1$ *from the fact that a constant function can be represented exactly by any scaling function basis set.*

The potential on the grid point $j1, j2, j3$ of same grid that was used for the charge density is then given by

$$\begin{aligned} V_{j1,j2,j3} &= \sum_{i1,i2,i3} \rho_{i1,i2,i3} \int_{-\infty}^{\infty} dx \int_{-\infty}^{\infty} dy \int_{-\infty}^{\infty} dz \frac{\phi(x-i1)\,\phi(y-i2)\,\phi(z-i3)}{\sqrt{(x-j1)^2+(y-j2)^2+(z-j3)^2}} \\ &= \sum_{i1,i2,i3} \rho_{i1,i2,i3} F_{i1-j1,i2-j2,i3-j3} \end{aligned} \tag{220}$$

Exercise [1pt]: *Using the above definition of the filter* F *as an integral show that it indeed depends only on the difference between two indices, e.g. only on* $i1 - j1$ *and not on* $i1$ *and* $j1$ *separately.*

Since the above expression for the potential $V_{j1,j2,j3}$ is a convolution it can be calculated with FFT techniques at the cost of $N^3 \log(N^3)$ operations where N^3 is the number of grid points. It remains to calculate the values of the filter $F_{i1-j1,i2-j2,i3-j3}$.

Calculating each of the N^3 filter elements as a 3-dimensional numerical integral would be too costly. The calculation becomes however feasible if the $1/r$ kernel is made separable. This can be achieved by representing it as a sum of Gaussians. The representation is best

based on the identity

$$\frac{1}{r} = \frac{2}{\sqrt{\pi}} \int_{-\infty}^{\infty} e^{-r^2 \exp(2s)+s} ds \tag{221}$$

Discretizing this integral we obtain

$$\frac{1}{r} = \sum_l w_l e^{-\gamma_l r^2} \tag{222}$$

With 89 well optimized values for w_l and γ_l it turns out that $1/r$ can be represented in the interval from 10^{-9} to 1 with an relative error of 10^{-8}. The 3-dimensional integral in Eq. 220 becomes then a sum of 89 products of 1-dimensional integrals.

$$\int dx \int dy \int dz \frac{\phi(x-i1)\,\phi(y-i2)\,\phi(z-i3)}{\sqrt{(x-j1)^2+(y-j2)^2+(z-j3)^2}} =$$

$$\sum_l w_l \int dx \int dy \int dz\, \phi(x-i1)\,\phi(y-i2)\,\phi(z-i3)\, e^{-\gamma_l\left((x-j1)^2+(y-j2)^2+(z-j3)^2\right)} =$$

$$\sum_l w_l \left(\int dx\, \phi(x-i1) e^{-\gamma_l (x-j1)^2}\right) \left(\int dy\, \phi(y-i2) e^{-\gamma_l (y-j2)^2}\right) \left(\int dz\, \phi(z-i3) e^{-\gamma_l (z-j3)^2}\right)$$

Using 89 terms in Eq. 222 we have thus to solve just $89N$ one-dimensional integrals which can be done extremely rapidly on a modern computer. The main cost are thus the FFT's required to calculate the convolution with the kernel $F_{i1-j1,i2-j2,i3-j3}$.

How to do convolutions

We consider a real input data set X_i of N items where $i = -N/2, ..., N/2-1$ and a real filter F_i where $i = -M/2, ..., M/2-1$ We want to calculate the one-dimensional convolution

$$Y_j = \sum_{l=-M/2}^{M/2-1} F_l^* X_{j+l} \tag{223}$$

If the filter is of short length, i.e. if M is small compared to N then it is most efficient to directly evaluate the sum in equation 223 for each value of j. The numerical effort is then obviously $N * M$. If M is however equal to N the scaling becomes N^2 and the FFT techniques explained below are more efficient for large N since the scaling becomes $N\log(N)$. It is clear from Eq. 223 that the data set Y is larger than the data set X. If both the data and the filter have data in the interval $i = -N/2, ..., N/2-1$, then the output data are in the range $i = -N+1, ..., N-2$. For simplicity we double the interval length and consider the interval $i = -N, ..., N-1$.
To apply the Fourier method we have to calculate the discrete Fourier coefficients x_k and f_q of the data sets X and F. The Fourier coefficients are obtained from the doubled data sets where all the coefficients in the interval $i = -N+1, ..., -N/2-1$ and in the interval

$N/2,\ldots,N-1$ are set to zero. Denoting $2N$ by $\tilde{N}$ we obtain

$$x_k = \frac{1}{\tilde{N}} \sum_{\nu} \exp(\frac{-2\pi I}{\tilde{N}} k\nu) X_\nu \tag{224}$$

$$f_k = \frac{1}{\tilde{N}} \sum_{\nu} \exp(\frac{-2\pi I}{\tilde{N}} k\nu) F_\nu \tag{225}$$

The original data can then be written in terms of these Fourier coefficients as

$$X_i = \sum_{k} \exp(\frac{2\pi I}{\tilde{N}} ki) x_k \tag{226}$$

$$F_i = \sum_{q} \exp(\frac{2\pi I}{\tilde{N}} qi) f_q \tag{227}$$

Exercise [1pt]: *Verify that plugging in the definition of the Fourier coefficients x_k into the above formula for X_i gives back the original data set X_i*
Inserting these expressions into Eq. 223 gives

$$Y_j = \sum_{k=-\tilde{N}}^{\tilde{N}-1} \sum_{q=-\tilde{N}}^{\tilde{N}-1} \sum_{l=-\tilde{N}}^{\tilde{N}-1} \exp(\frac{-2\pi I}{\tilde{N}} ql) f_q \exp(\frac{2\pi I}{\tilde{N}} k(j+l)) x_k \tag{228}$$

$$= \sum_k \exp(\frac{2\pi I}{\tilde{N}} kj) \sum_q \sum_l \exp(\frac{-2\pi I}{\tilde{N}} (q-k)l) f_q x_k \quad (229)$$

$$= \tilde{N} \sum_k \exp(\frac{2\pi I}{\tilde{N}} kj) \sum_q \delta_{q,k} f_q x_k \quad (230)$$

$$= \tilde{N} \sum_k \exp(\frac{2\pi I}{\tilde{N}} kj) f_k x_k \quad (231)$$

The previous line shows that the Fourier coefficients of the output data set Y are the product of the Fourier coefficients of X and F. So we have to form these products and then to do another Fourier transform to obtain the data set Y. All the Fourier transformations can be done using the Fast Fourier transformation (FFT) algorithm at a cost of $\tilde{N}\log(\tilde{N})$ operations.

In the mathematical literature sums in Fourier transformation formulas typically run from $-N$ to N or $N-1$. In all numerical FFTs indices run from 0 to $N-1$. For all the real data this just implies a shift whereas for data in Fourier space it means that the negative frequencies are in the second half of the data set as shown below for the case of N=4:

$$x_0, x_1, x_2, x_3, x_4, x_{-3}, x_{-2}, x_{-1}$$

8 Integration methods

8.1 1-dim Integration Methods

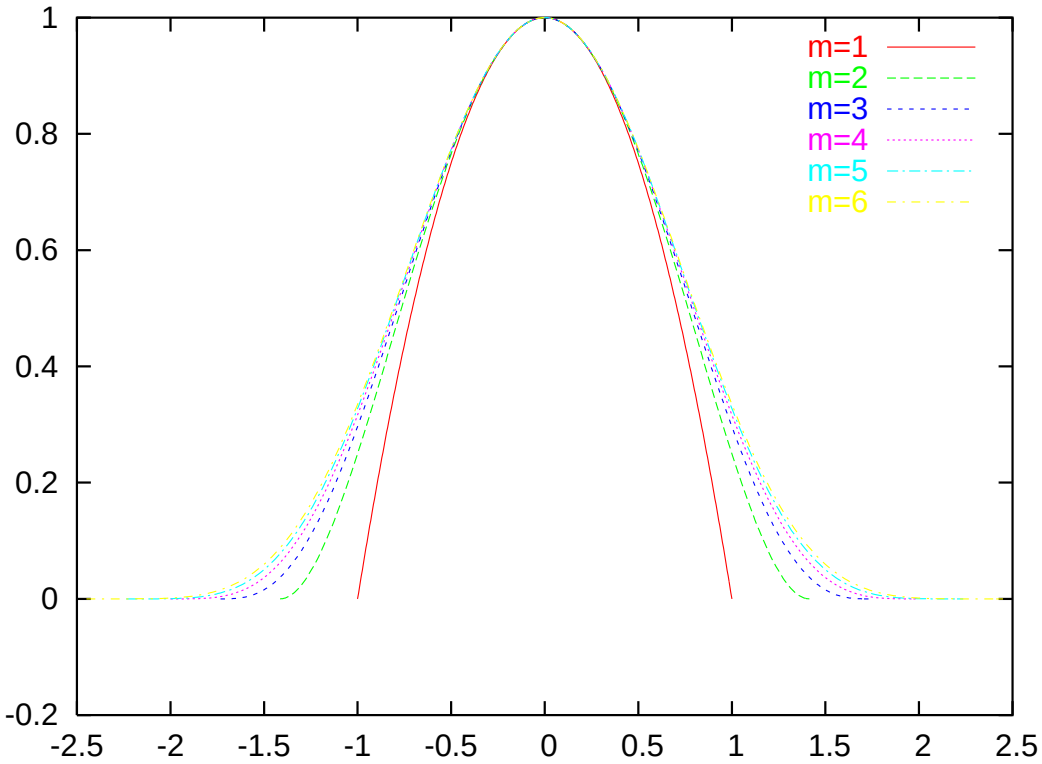
Numerical integration approximates an integral $I = \int_a^b f(x)dx$ by a finite sum $S = \sum_i f(x_i)w_i$, where w_i are the integration weights and x_i the integration points. For best efficiency one tries to get the highest accuracy with the smallest number of integration points. The efficiency that can be obtained depends on two factors

- The smoothness of the function to be integrated, A function is called smooth if many continuous derivatives exist.
  - the smoothness in the interior of the integration interval
  - the behavior of the function at the boundaries of the integration interval, i.e whether the function and the lowest derivatives vanish. If we artificially extend the integration interval to $[-\infty : \infty]$ by putting the function outside the original integration interval $[a : b]$ to 0 this behavior is again described by the smoothness properties of the function.
- The choice of the integration grid points x_i and their weights w_i

As an illustration let us consider the following family of functions shown below

$$f_m(x) = (1 - \frac{x^2}{m})^m \tag{232}$$

that we want to integrate between its two zeros at $-\sqrt{m}$ and $\sqrt{m}$. At the zeros $m-1$ derivatives vanish as well.

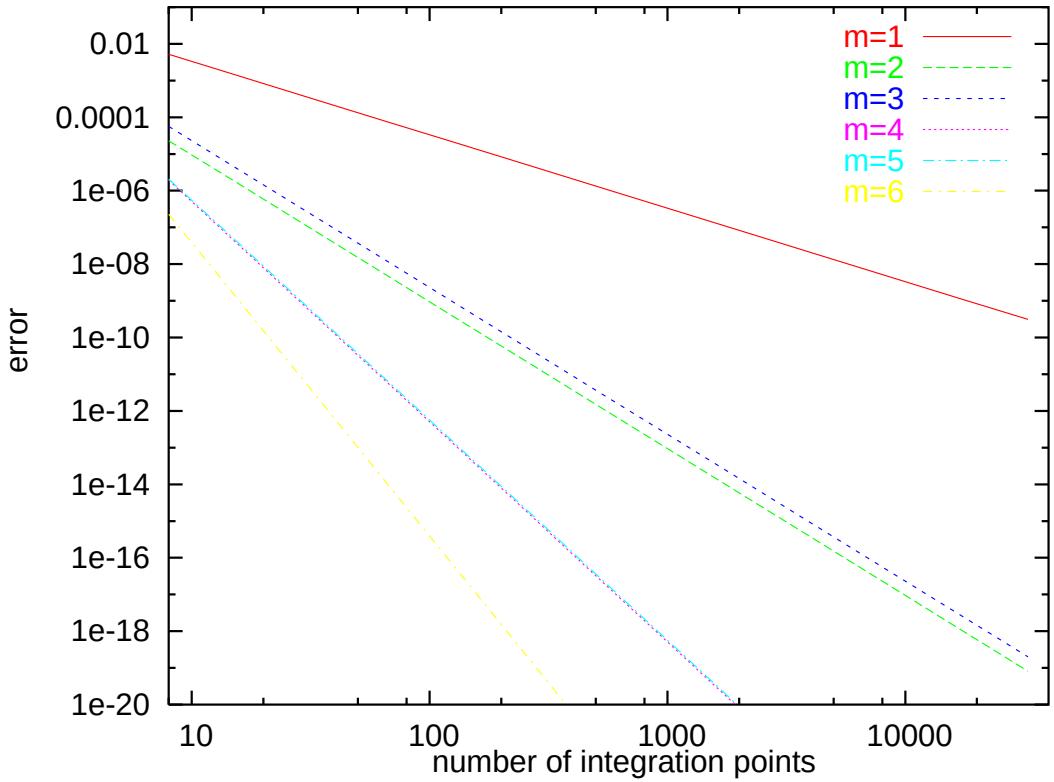


Integrating the above function between $-\sqrt{m}$ and $\sqrt{m}$ is equivalent to integrating over the

whole real axis the function

$$f_m(x) = \begin{cases} (1-\frac{x^2}{m})^m & \text{if } |x| < \sqrt{m} \\ 0 & \text{else} \end{cases} \tag{233}$$

The above family of functions becomes smoother with increasing m since $m-1$ derivatives are continuous everywhere.
The errors with the simplest integration scheme, namely an equally spaced grid with $w_i = 1$ are shown below. It is clearly seen, the smoother the function, the faster this simple integration scheme converges.
Exercise [1pt]: *Guess which function can be integrated from $-\infty$ to ∞ with the smallest number of equally spaced integration points and $w_i = 1$?*



A simple equally spaced grid is optimal for the integration over the entire space ($-\infty$ to ∞) of an analytic function, or for the integration over the periodicity volume of an analytic periodic function. Optimality in this context means that exponential convergence can be obtained, i.e a convergence that is faster than any power with respect to the number of integration points. The exponential convergence follows from the Paley-Wiener theorem.

If we consider the Fourier development of a function that is periodic in the interval $[0:1]$

$$f(x) = \sum_{k=-\infty}^{\infty} c_k e^{I2\pi kx} \tag{234}$$

the Paley-Wiener theorem tells us that the Fourier coefficients c_k decay exponentially for large $|k|$ if the function is analytic. Hence we have

$$\frac{1}{N}\sum_{j=0}^{N-1} f(j/N) = \sum_{k=-\infty}^{\infty} c_k \frac{1}{N}\sum_{j=0}^{N-1} e^{I2\pi kj/N} \tag{235}$$

Since

$$\frac{1}{N}\sum_{j=0}^{N-1} e^{I2\pi kj/N} = \begin{cases} 1 & \text{if } k \text{ is a multiple of } N \\ 0 & \text{else} \end{cases} \tag{236}$$

we get

$$\frac{1}{N}\sum_{j=0}^{N-1} f(j/N) = c_0 + \sum_{m=1}^{\infty} (c_{mN} + c_{-mN}) \tag{237}$$

Since c_0 is the exact value of the integral, the second term is the error term. As asserted by the Paley-Wiener theorem it decays exponentially.

Exercise [2pt]: *Demonstrate numerically exponential convergence for an analytic periodic function. A possible choice is*

$$\int_0^1 \exp((\sin(\pi x))^2)dx = 1.7533876543770903957 \tag{238}$$

where $\pi = 3.1415926535897932385$

For a non-periodic function that is not perfectly smooth the following techniques may be applied:

- Find a transformation that makes it smoother
- Use weights w_i that give better convergence for non-smooth integrands. Typically the worst non-smooth places are at the upper and lower integration limits. So the weights should be modified close to the integration limits.
- Use in addition to optimal weights non equally spaced integration points (Gauss integration)

The first two techniques will be illustrated in the following

Integration of a radial wave-function after a transformation

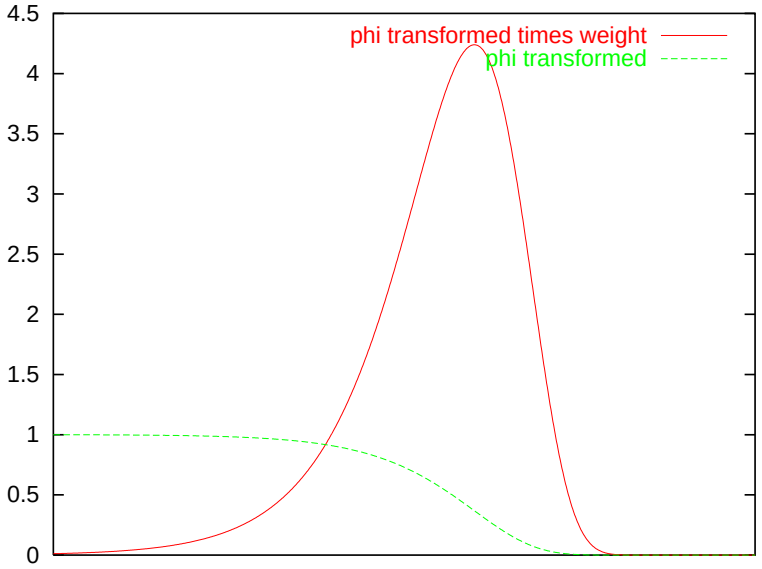
There are two problematic regions for the integration of a radial wave-function such as the 1s hydrogen wave-function e^{-r}. Near the origin the wave-functions have a considerable variation and high derivatives are important. In the tail region we have the problem that the wave-function extends to rather large radii without much variation. Consequently we need a transformation that stretches the wave-function near the origin and that compresses the wave-function in the tail region. The stretching near the origin will introduce small weights near the origin. As a consequence the quantity to be integrated which is the product of the transformed function and the weight will tend to zero. A transformation with this property is

$$r = f(x) = a\exp(\alpha x) - a \tag{239}$$

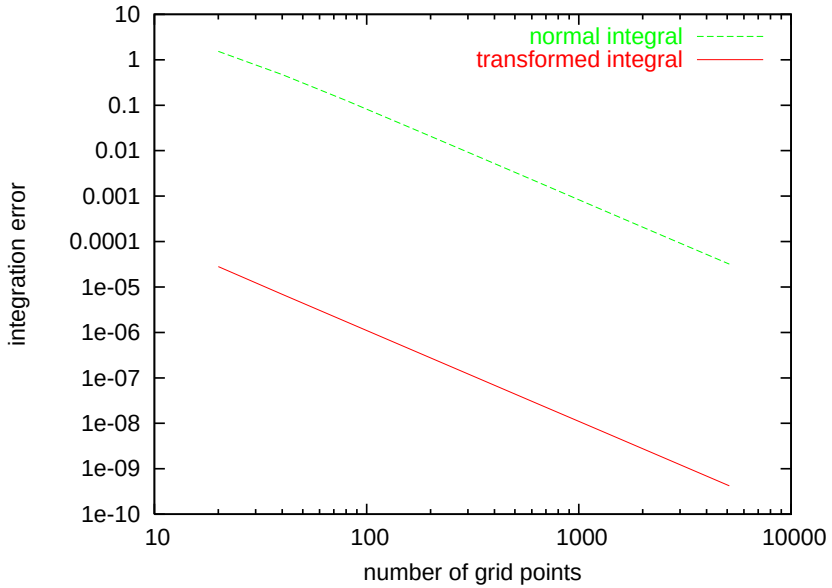
The constant a determines how much the function is stretched near the origin. α determines where the function will start to drop to zero. An integral over a radial function ϕ can then be written as

$$\int_0^\infty \phi(r)dr = \int_0^\infty \phi(f(f^{-1}(r)))dr = \int_0^\infty \tilde{\phi}(f^{-1}(r))dr = \int_0^\infty \tilde{\phi}(x)\frac{dr}{dx}dx = \int_0^\infty \tilde{\phi}(x)\alpha a\exp(\alpha x)dx \tag{240}$$

where we have introduced the transformed function $\tilde{\phi}(x) = \phi(f(x))$ The function $\tilde{\phi}(x)$ resulting from $\phi(r) = \exp(-r)$ by itself and multiplied by the weights is plotted below.



The integration error resulting from a numerical integration of ϕ and $\tilde{\phi}$ is shown below.



Exercise [2pt]: *Find a transformation such that* $\int_{-\infty}^{\infty} \frac{1}{1+x^2} dx = \pi$ *can be calculated with an error of less than* 10^{-10} *with the smallest possible number of integration points.*

Weights for numerical integration of functions on short intervals

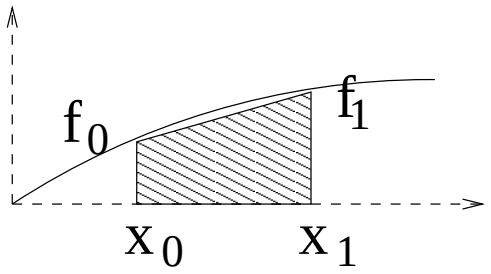
Let us assume that we want to integrate a function $f(x)$ and that we know the functional values $f_i = f(x_i)$ on several equally spaced values $x_i = x_0 + ih$. An integration formula gives us an approximate value for the integral under the form of a weighted sum of the functional values f_i.

$$\int_{x_0}^{x_m} f(x)\,dx \approx h \sum_{i=0}^{m} f_i w_i \tag{241}$$

The basic principle for deriving numerical integration formulas is analogous to the one used for numerical differentiation. First find a polynomial approximation to the function and then integrate the polynomial. The lowest order integration formula, called the trapezoidal rule, is obvious.

$$\int_{x_0}^{x_1} f(x)\,dx = \left(\frac{1}{2}f_0 + \frac{1}{2}f_1\right) h \tag{242}$$

It is the area of a linear function that passes through the two points (x_0, f_0) and (x_1, f_1) as shown below



The integration weights w_i for high order integration formulas are again best calculated by symbolic computation. The following Mathematica program gives the weights for 4 integration points

```
f[x_]:=Evaluate[InterpolatingPolynomial[{{0,y0},{1,yp1},{2,yp2},{3,yp3}},x]]
tt=Simplify[Integrate[f[x],{x,0,3}]]
```

The output is shown below

```
        3 (y0 + 3 yp1 + 3 yp2 + yp3)
Out[1]= ----------------------------
                      8
```

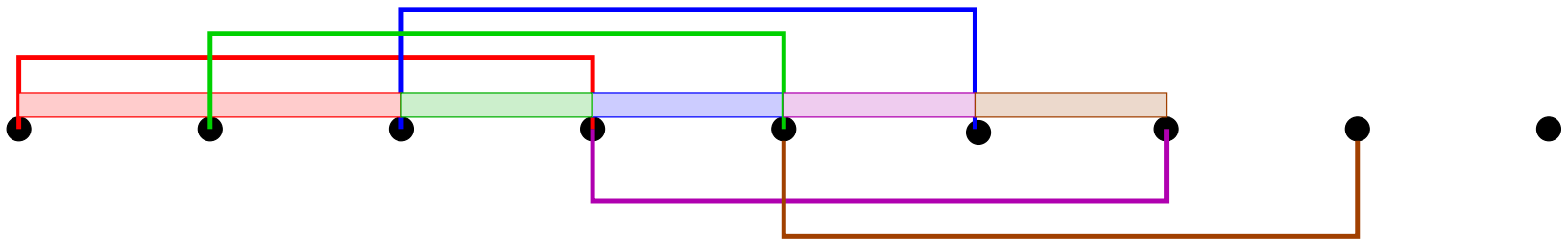
The error analysis is also analogous to the case of the finite difference formulas. One considers the Taylor expansion of the function to be integrated. The formula will integrate exactly the first $m+1$ terms of the Taylor expansion.

Table 2: Integration coefficients w_i (Eq. 241) on short intervals of various order.

O	w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7
h^3	1/2	1/2						
h^5	3/8	9/8	9/8	3/8				
h^7	95/288	125/96	125/144	125/144	125/96	95/288		
h^9	5257/17280	25039/17280	343/640	20923/17280	20923/17280	343/640	25039/17280	5257/17280

Weights for numerical integration of functions on long intervals

The integration formulas for short intervals were motivated by the assumption that the function can be represented over the whole interval by a single Taylor expansion. Such an assumption does generally not hold true for functions found in science and engineering that extend over a longer interval. Such functions have typically a different behavior in different regions of the integration interval. Even though a global Taylor expansion is not adequate, we can assume that Taylor expansions for smaller local subintervals are accurate. One could thus subdivide a large interval into smaller subintervals and use the integration formulas derived for short intervals in each subinterval. This would give the strange result that different points in the middle have different weight, even though all points are 'equal'. This artifact can be avoided if one uses the polynomial constructed over several grid points to integrate only the interval between 2 grid points. This is shown schematically below for the points close to the left integration boundary. To integrate an interval denoted by a certain color, one uses a polynomial that goes trough the points below (or above) the bar with the same color.

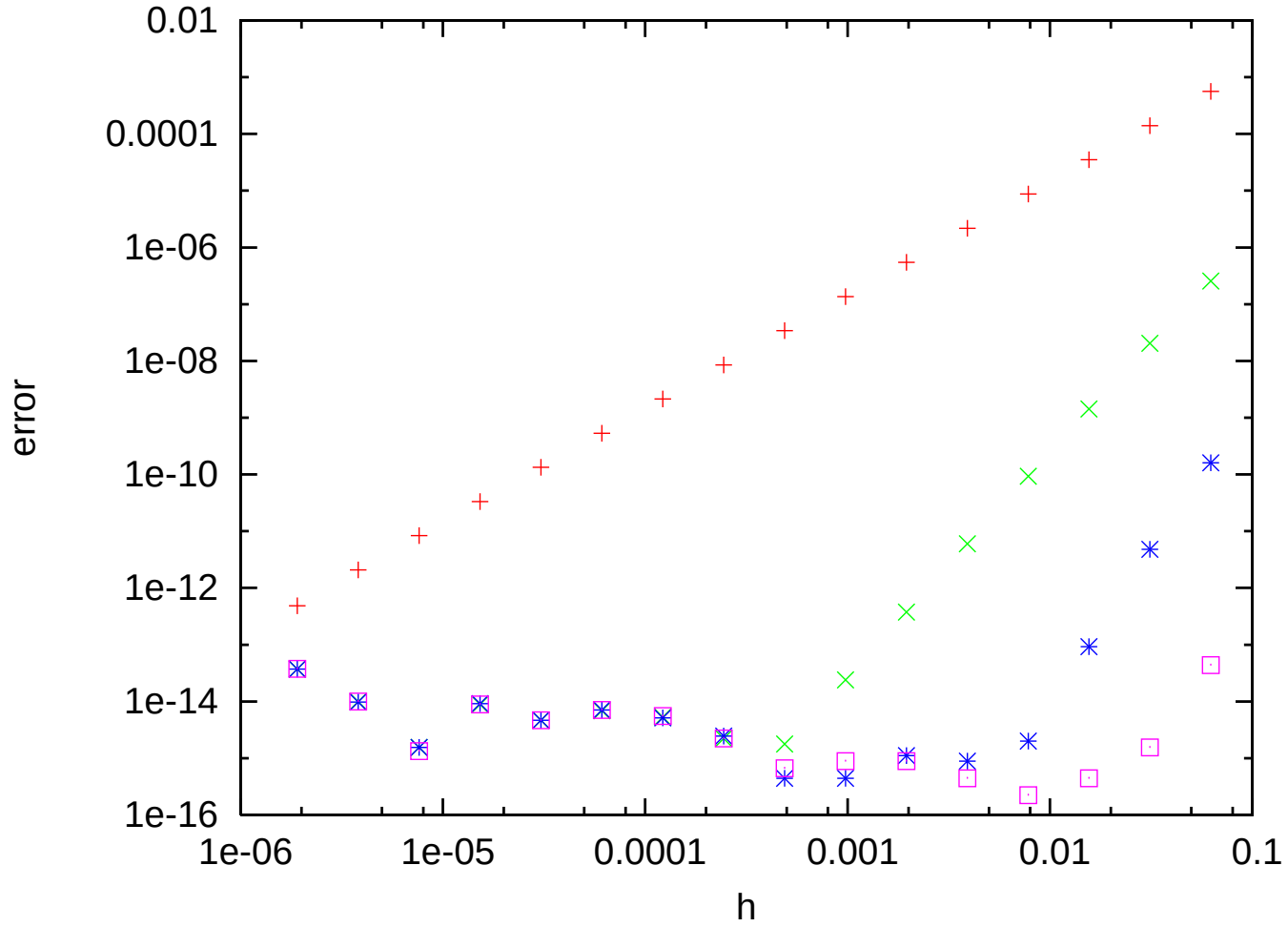


The integration weights obtained in this way are listed below

Table 3: The coefficients w_i for integration formulas on long intervals of various order. Listed are only the coefficients for the left integration boundary, the coefficients for the right boundary are identical, e.g. $w_m = w_0$, $w_{m-1} = w_1$. etc. The integration weights w_i in the middle part are all equal to 1.

O	w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7
h^3	1/2	1	1	1	1	1	1	1
h^5	1/3	31/24	5/6	25/24	1	1	1	1
h^7	14/45	679/480	139/240	58/45	71/80	163/160	1	1
h^9	41/140	6899/4480	3247/15120	226109/120960	1291/3780	159811/120960	2749/3024	24467/24192

The figure below shows the error for all the 4 sets of integration coefficients of Table 3



Exercise [2pt]: *Derive the integration weights of order h^5 in Table 3*
Hint: The following Mathematica program

```
f[x_]:=Evaluate[InterpolatingPolynomial[{{0,y0},{1,yp1},{2,yp2},{3,yp3}},x]]
Simplify[Integrate[f[x],{x,1,2}]]
```

gives this output

```
        -y0 + 13 yp1 + 13 yp2 - yp3
Out[1]= ---------------------------
                    24
```

and the program

```
f[x_]:=Evaluate[InterpolatingPolynomial[{{0,y0},{1,yp1},{2,yp2},{3,yp3}},x]]
Simplify[Integrate[f[x],{x,0,1}]]
```

gives this output

```
        9 y0 + 19 yp1 - 5 yp2 + yp3
Out[2]= ---------------------------
                    24
```

8.2 High Dimensional Integration Methods

The one dimensional product formulas can be generalized to higher dimensions. For an analytic function f we obtain the following integration formulas and error estimates of a regular grid of $n^d = N$ grid points in an d-dim space.

- Simple summation on regular grid

$$\int_0^1 dx_1 \int_0^1 dx_2 \ldots \int_0^1 dx_d \, f(\vec{x}) = \frac{1}{n} \sum_{i_1=0}^{n-1} \frac{1}{n} \sum_{i_2=0}^{n-1} \ldots \frac{1}{n} \sum_{i_d=0}^{n-1} f(\frac{i_1}{n}, \frac{i_2}{n}, \ldots \frac{i_d}{n}) + O(\frac{1}{N^{1/d}}) \tag{243}$$

- Trapezoidal rule

$$\int_0^1 dx_1 \int_0^1 dx_2 \ldots \int_0^1 dx_d \, f(\vec{x}) = \frac{1}{n} \sum_{i_1=0}^{n} w_{i_1} \frac{1}{n} \sum_{i_2=0}^{n} w_{i_2} \ldots \frac{1}{n} \sum_{i_d=0}^{n} w_{i_d} f(\frac{i_1}{n}, \frac{i_2}{n}, \ldots \frac{i_d}{n}) + O(\frac{1}{N^{2/d}}) \tag{244}$$

  where w_i is 1/2 for $i = 0, n$ and 1 for all other values of i.

- Going to ever higher order integration schemes gives an error of $O(\frac{1}{N^{l/d}})$ where l is the order of the one dimensional integration scheme. For large d this error will decrease very slowly.

So the problem is that these product formulas give very poor convergence with respect to the total number of grid points in a high dimensional space. Another disadvantage is that it is very difficult to check convergence. One cannot easily add some more grid points to check the convergence. Unless one throws away the old result one has to double the number of grid point in each direction, which will lead to a increase in the total number of grid points by a factor of 2^d.
For these reasons Monte Carlo Integration is frequently recommended for high dimensional integration. Given a sequence of random numbers $\vec{x}_i$ the integration formula is

$$\int_0^1 dx_1 \int_0^1 dx_2 ... \int_0^1 dx_d f(\vec{x}) = \frac{1}{N} \sum_{i=1}^{N} f(\vec{x}_i) + O(\frac{\sigma}{\sqrt{N}}) , \tag{245}$$

The error term is not a strict error bound, but only an estimate. Its form comes from the assumption that the integral value has a normal distribution. This is fulfilled according to the central limit theorem in the limit of large N. The standard deviation σ is given by

$$\sigma = \sqrt{< f^2 > - < f >^2} \tag{246}$$

where

$$< f >= \frac{1}{N} \sum_{i=1}^{N} f(\vec{x}_i) \qquad\qquad < f^2 >= \frac{1}{N} \sum_{i=1}^{N} f(\vec{x}_i)^2 \tag{247}$$

Exercise [2pt]: *Show numerically that the result of a Monte Carlo integration of* $\int_0^1 x dx$ *has a normal distribution*

The Convergence of the Monte Carlo method is too slow to be useful in practice unless the variance of the function is very small from the beginning. Sometimes it is possible to reduce the variance of the integration problem by what is called importance sampling. For importance sampling we need to find a positive function $p(x)$ that has about the same shape as the function f to be integrated and we normalize it to one ($\int p(x)dx = 1$).

$$\int_a^b f(x)dx = \int_a^b p(x)\frac{f(x)}{p(x)}dx \tag{248}$$

The new function $\frac{f(x)}{p(x)}$ has a lower variance and the above integral can be approximated by

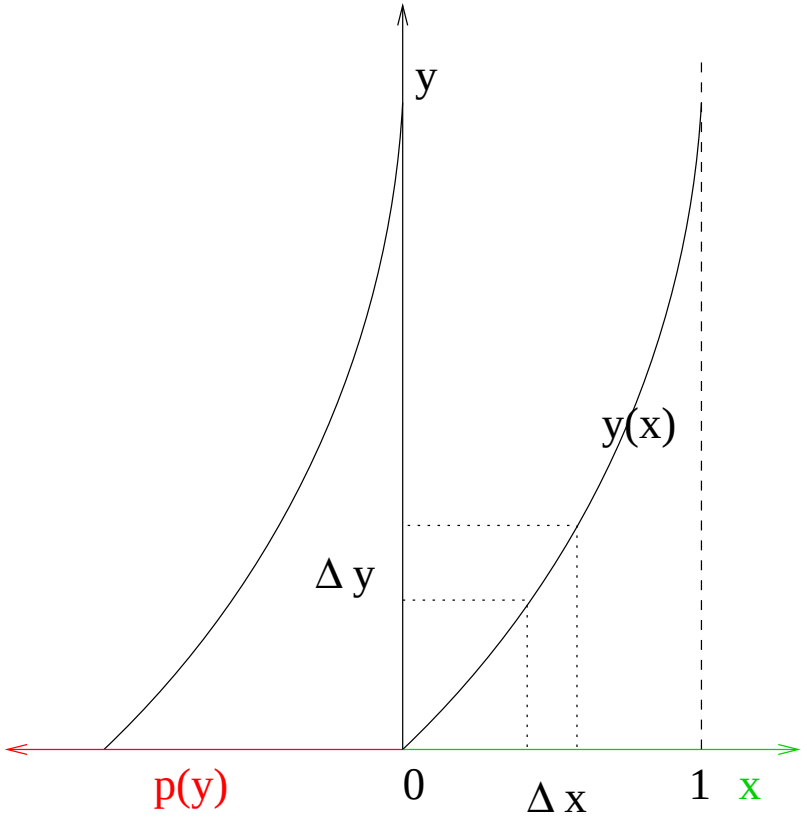
$$\frac{b-a}{n}\sum_{j=1}^{n}\frac{f(y_j)}{p(y_j)} \tag{249}$$

if the random points y_j are now distributed according to the distribution $p(x)$. The standard random number generators generate a sequence of numbers that are uniformly distributed in the interval $]0:1]$, i.e. $p(x) = 1$ in this interval and $p(x) = 0$ outside this interval. A random number sequence that is distributed according to another distribution can frequently be obtained in the 1-dim case by applying a function y on the output of a standard random

sequence x_j.

$$y_j = y(x_j) \tag{250}$$

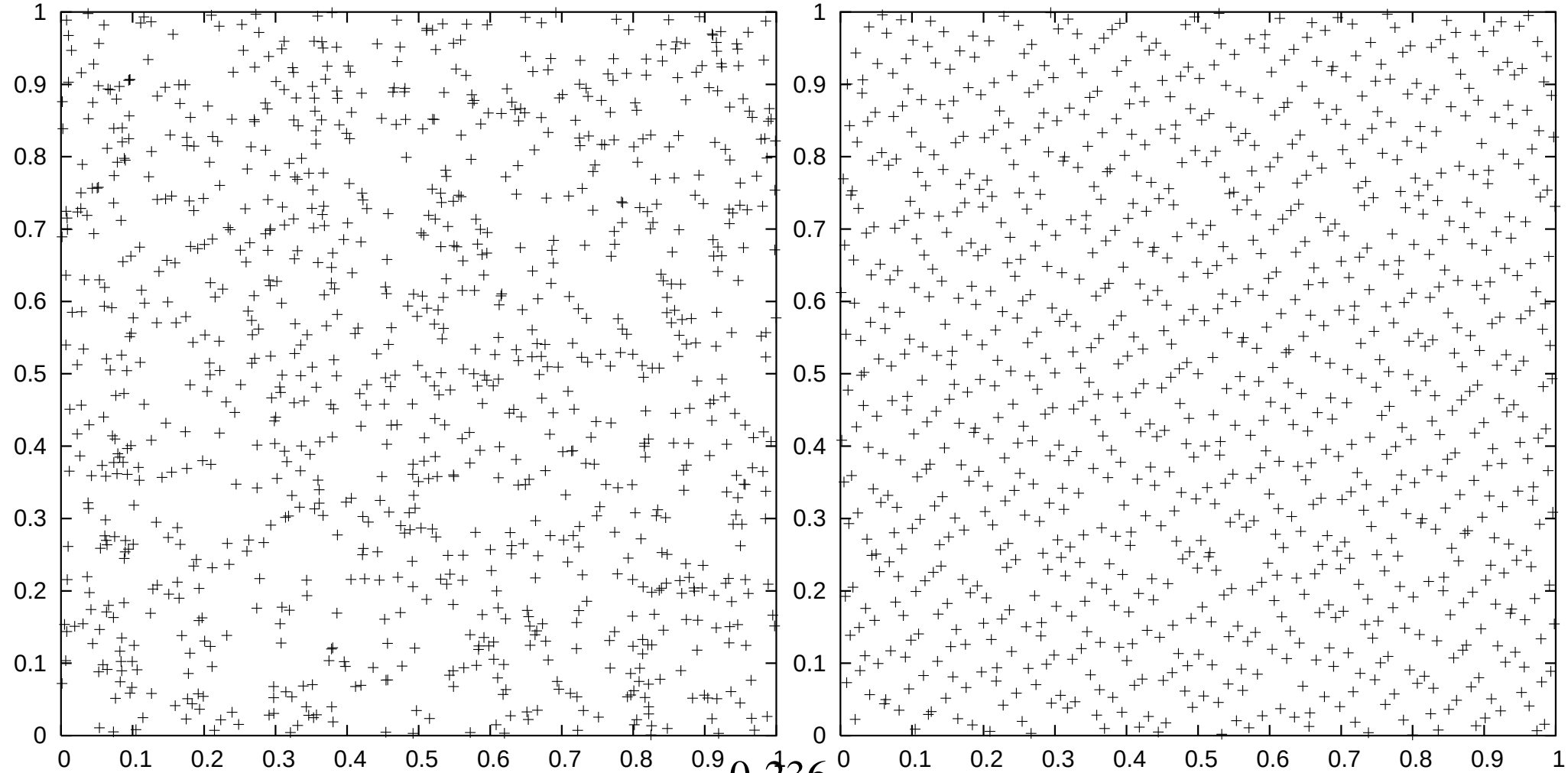
y_j is then distributed according to $1/|y'|$. If one wants a certain distribution one has consequently to find the function y whose reciprocal of the derivative will give this distribution. The situation is illustrated pictorially below.



Exercise [1pt]: *Write a subroutine that gives random numbers that are exponentially distributed in* $[0 : \infty]$

Quasi-random integration points

The slow $1/\sqrt{N}$ convergence of the Monte Carlo integration comes from the fluctuations in the density of the random points used as evaluation points. It would be desirable to have a more uniform coverage of the integration volume. Such sequences exist and are called quasi random numbers, or low discrepancy sequences. Below, the first 1000 points of a random and and of quasi-random sequence are contrasted.



With quasi-random integration points a convergence rate of $\frac{\log(N)^d}{N}$ can be obtained for smooth functions, where d is the dimension of the integration space. Conceptually the simplest low discrepancy sequence is Halton's sequence. We write the counting numbers $0, 1, 2, 3, \ldots$ first in base 2

$$0, 1_2, 10_2, 11_2, 100_2, 101_2, 110_2, \ldots \tag{251}$$

then in base 3

$$0, 1_3, 2_3, 10_3, 11_3, 12_3, 20_3, \ldots \tag{252}$$

then in base 5, and so on through the first d primes. Then each of these representations of the counting number is reversed to obtain a number in $[0 : 1[$. So, for example the sequence coming from the base 2 representation is

$$0, 0.1_2, 0.01_2, 0.11_2, 0.001_2, 0.101_2, 0.011_2, \ldots \tag{253}$$

while the base 3 sequence gives

$$0, 0.1_3, 0.2_3, 0.01_3.0.11_3, 0.21_3, 0.02_3, \ldots \tag{254}$$

From the construction it is obvious that this sequence fills space in a rather uniform way. A very powerful sequence that can be used for integration in spaces of dimension of up to

roughly 30 is Sobols sequence, which is based on very sophisticated mathematics. Even though it has the same asymptotic convergence rate as Haltons sequence, it has smaller prefactors in the error term. The first 1000 points of the Sobol sequence were shown on the previous page. When integrating discontinuous functions, quasi random sequences do not perform much better than random sequences.
The fact that at present there exists no satisfactory scheme to combine quasi-random sequences with importance sampling limits the use of quasi-random sequences in physics.

9 Monte Carlo methods

A wide variety of simulation methods is denoted by Monte Carlo methods. The only thing that all these methods have in common is that they are using random numbers in some way. Random numbers generated by a random number generator on a computer are numbers that are equally distributed in the interval $[0:1)$ and that do not exhibit any recognizable pattern. Since they are generated by a deterministic procedure, there is of course some pattern, but since the pattern is well hidden it should not matter in a simulation and the computer generated random numbers should behave like truly random numbers, i.e. they should represent a completely unpredictable sequence of numbers. The simplest and most widely used type of random number generator is the linear congruential operator which generates a sequence of integers I_j where

$$I_{j+1} = mod(aI_j + c, m) \tag{255}$$

For properly chosen integers a and c the sequence will have a period of m, where m is typically the integer word size of the machine, i.e. $m = 2^{32}$. The corresponding real random number in the interval $[0:1)$ is simply obtained as $I_j/2^{32}$.
The Metropolis algorithm plays a central role in most Monte Carlo methods. We will first discuss its use for the generation of distributions in classical statistical mechanics.

High dimensional integrals in physics

High dimensional integrals have to be evaluated in various contexts to treat many-body systems. One example is the quantum mechanical energy expectation value E of a many-electron wave function $\Psi(\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_N)$

$$E_{QM} = \int \mathbf{dr}_1, \mathbf{dr}_2, ..., \mathbf{dr}_N \Psi(\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_N) H \Psi(\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_N)$$

where H is the many electron Hamiltonian. Another example is the thermodynamic energy expectation value E of a classical many body system

$$< E >= \frac{\int \mathbf{dr}_1, \mathbf{dr}_2, ..., \mathbf{dr}_N e^{-\frac{1}{k_B T} E(\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_N)} E(\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_N)}{\int \mathbf{dr}_1, \mathbf{dr}_2, ..., \mathbf{dr}_N e^{-\frac{1}{k_B T} E(\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_N)}}$$

where $E(\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_N)$ is the potential energy surface, T the temperature and k_B the Boltzmann constant. For large values of N all the previously discussed methods will fail in evaluating these integrals. In both cases the integrand is virtually zero in the entire high dimensional space except in a subvolume that is vanishingly small. It is extremely unlikely that either a random or quasi random point will fall in such a subvolume. Hence a random or quasi random integration will give the wrong value of zero even for a large number of random integration points. The only class of methods that works under such circumstances are the Monte Carlo methods.

9.1 The Metropolis algorithm

The Metropolis algorithm is the standard method to generate random points with a certain distribution in high dimensional spaces. Since a high dimensional random point corresponds generally in physics to the configuration of some system (e.g. all the positions of a many particle system) we will use the word configuration rather than random point and we will denote such a configuration by X. The vast majority of the total configurational space of a many particle system corresponds to unphysical configurations where for instance 2 atoms are very close to each other. This results in extremely high energies that contribute for instance virtually nothing to a Boltzmann distribution at room temperature. Importance sampling, i.e. creating distributions that sample only the low energy part of the configurational space is therefore essential.
The Metropolis algorithm is based on a Markov chain. In a Markov chain the next configuration X' is obtained from the present configuration X by a certain move that is characterized by a transition probability $T(X' \leftarrow X)$. The probability $P_N(X_1, X_2, ..., X_N)$ of finding a certain sequence of configurations is therefore given by

$$P_N(X_1, X_2, ..., X_N) = T(X_N \leftarrow X_{N-1})...T(X_3 \leftarrow X_2)T(X_2 \leftarrow X_1)P_1(X_1) \tag{256}$$

where the transition probabilities are normalized

$$\sum_{X'} T(X' \leftarrow X) = 1 \tag{257}$$

and where $P_1(X)$ is an arbitrary distribution. For a truly random (i.e. non-Markovian) sequence this probability would be given by

$$P_N(X_1,X_2,...,X_N) = P_1(X_1)P_1(X_2)...P_1(X_N) \tag{258}$$

For Markov processes it is convenient to introduce the notion of a walker. Instead of saying that one configuration is obtained from another one says that the walker goes from one configuration to another. The concept of a walker does not in any way modify the mathematics, it just gives a more intuitive description of a Markov process. The simplest example of a Markov process is a random walk on a 2-dim square lattice. Each node represents a configuration. At any step the walker can jump to any of its 4 nearest neighbors, i.e. $T(X' \leftarrow X) = 1/4$, independently of which site he visited before. The walker behaves in this example like a drunken sailor who randomly walks from one intersection to the next in a city.

Let us now introduce an ensemble of walkers together with the function $P(X,t)$ which gives the probability of finding a walker at configuration X at Markov step t. The probability $P(X,t+1)$ is then given by $P(X,t)$ plus the gain $\sum_{X'} T(X \leftarrow X')P(X',t)$ minus the loss $\sum_{X'} T(X' \leftarrow X)P(X,t)$. Once equilibrium has been reached $P(X,t+1) = P(X,t) = P(X)$ and hence

$$\sum_{X'} T(X' \leftarrow X)P(X,t) = \sum_{X'} T(X \leftarrow X')P(X',t) \tag{259}$$

This equation is satisfied under the conditions of a detailed balance where

$$T(X' \leftarrow X)P(X) = T(X \leftarrow X')P(X') \tag{260}$$

Like in molecular dynamics there is an equivalence in Monte Carlo methods between ensemble averages and time averages. Unlike in molecular dynamics there is no continuous time variable, but the discrete 'time' variable t that denotes the t-th Markov step takes over the role of the time variable in molecular dynamics. Our equilibrium distribution $P(X)$ gives the probability of finding a system in configuration X in an ensemble of systems. Numerically we rather take time averages by following the movement of one walker and so $P(X)$ is also the probability for finding a walker at configuration X. Since the distribution of walkers $P(X,t)$ tends to $P(X)$ we have

$$P(X) = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} P(X,t) \tag{261}$$

The transition probability in a Markov process consists of two parts: a trial step probability $\omega_{X',X}$ and an acceptance probability $A_{X',X}$

$$T(X' \leftarrow X) = \omega_{X',X} A_{X',X} \tag{262}$$

Since $\omega_{X,X'}$ is required to be symmetric, i.e. $\omega_{X',X} = \omega_{X,X'}$ the detailed balance condition (Eq. 260) gives

$$\frac{A_{X',X}}{A_{X,X'}} = \frac{P(X')}{P(X)} \tag{263}$$

The Metropolis acceptance prescription accepts a trial step if the probability of the new configuration X' is larger than of the old configuration X and accepts it with a probability $\frac{P(X')}{P(X)}$ in the opposite case:

$$A_{X',X} = \begin{cases} 1 & \text{if } P(X') > P(X) \\ \frac{P(X')}{P(X)} & \text{if } P(X') < P(X) \end{cases} \tag{264}$$

That the above prescription satisfies Eq. 263 can easily be seen by considering the two possible cases. If $P(X') > P(X)$

$$\frac{A_{X',X}}{A_{X,X'}} = \frac{1}{\frac{P(X)}{P(X')}} = \frac{P(X')}{P(X)}$$

If $P(X') < P(X)$

$$\frac{A_{X',X}}{A_{X,X'}} = \frac{\frac{P(X')}{P(X)}}{1} = \frac{P(X')}{P(X)}$$

Accepting with a certain probability is done in the following way numerically. A random number, equally distributed in the interval $[0:1]$, is generated by calling a random number generator. If this random number is less than $\frac{P(X')}{P(X)}$ the step is accepted, otherwise it is rejected.

The Metropolis algorithm is certainly not the only stochastic process that leads to a certain probability distribution, but it is presumably the simplest one. Its simplicity is mainly due to the requirements of the detailed balance and the symmetry of ω. Other, non-Markovian processes, where the probability for a certain sequence depends on all previous configurations, have probability expressions that are more complicated than truly random (Eq. 258) or Markov sequences (Eq. 256). An example of such a sequence is given by the self-avoiding random walk in which the walker is not allowed to visit a site that has been visited in the past.

The trial step probabilities $\omega_{X,X'}$ do not enter into the Metropolis acceptance criterion (Eq. 264). Nevertheless they play an important role. First, they have obviously to be chosen in such a way that any configuration of the system can be reached. Second, their choice determines how fast the equilibrium distribution $P(X)$ is reached. With a bad choice, it may well turn out that the equilibrium distribution can not be reached within the available computer time.

To illustrate the effect of the probabilities $\omega_{X,X'}$ on the equilibration rate, let us consider a simple model system that consists of 11 states i with energies $\varepsilon_i = i$, $i = 0, 1, ..., 8, 9, 10$.

We want to generate the Boltzmann equilibrium distribution at temperature T

$$p_i = P(i,\infty) = \exp(-\beta\varepsilon_i) / \left(\sum_k \exp(-\beta\varepsilon_k) \right) \tag{265}$$

where $\beta = 1/(k_B T)$. We compare the speed with which we converge to this Boltzmann distribution for two different trial step probability matrices. The first matrix $\omega_{i,j}$ connects only states that are neighbors in energy, i.e. state i is connected to state $i+1$ and $i-1$ with some kind of periodic boundary conditions.

```
 0   .5    0    0    0    0    0    0    0    0   .5
.5    0   .5    0    0    0    0    0    0    0    0
 0   .5    0   .5    0    0    0    0    0    0    0
 0    0   .5    0   .5    0    0    0    0    0    0
 0    0    0   .5    0   .5    0    0    0    0    0
 0    0    0    0   .5    0   .5    0    0    0    0
 0    0    0    0    0   .5    0   .5    0    0    0
 0    0    0    0    0    0   .5    0   .5    0    0
 0    0    0    0    0    0    0   .5    0   .5    0
 0    0    0    0    0    0    0    0   .5    0   .5
.5    0    0    0    0    0    0    0    0   .5    0
```

The second matrix $\omega_{i,j}$ allows for transitions between any pair of states:

```
 0   .1   .1   .1   .1   .1   .1   .1   .1   .1   .1
.1    0   .1   .1   .1   .1   .1   .1   .1   .1   .1
.1   .1    0   .1   .1   .1   .1   .1   .1   .1   .1
.1   .1   .1    0   .1   .1   .1   .1   .1   .1   .1
.1   .1   .1   .1    0   .1   .1   .1   .1   .1   .1
.1   .1   .1   .1   .1    0   .1   .1   .1   .1   .1
.1   .1   .1   .1   .1   .1    0   .1   .1   .1   .1
.1   .1   .1   .1   .1   .1   .1    0   .1   .1   .1
.1   .1   .1   .1   .1   .1   .1   .1    0   .1   .1
.1   .1   .1   .1   .1   .1   .1   .1   .1    0   .1
.1   .1   .1   .1   .1   .1   .1   .1   .1   .1    0
```

The entire transition matrix is given by

$$T_{i,j} = \begin{cases} \omega_{i,j} & \text{if } \varepsilon_i < \varepsilon_j \\ \omega_{i,j} \frac{p_i}{p_j} = \omega_{i,j} \exp(-\beta(\varepsilon_i - \varepsilon_j)) & \text{if } \varepsilon_i > \varepsilon_j \\ 1 - \sum_{k \neq j} T_{k,j} & \text{if } i = j \end{cases} \tag{266}$$

where $T_{i,j}$ denotes the probability for a transition from the j-th state to the i-th state. The transition matrix T describes the dynamics of an ensemble of N systems that undergo the above defined Markov process. If all the systems are initially in state j, then after one step there will be $NT_{i,j}$ systems in state i. This relation holds in an average way for finite N and is exact in the limit of large N. Let us now introduce the vector $\mathbf{P}(l)$ which is obtained by applying the transition matrix T l times on the initial state vector $\mathbf{P}(0)$.

$$\mathbf{P}(l) = T^l \mathbf{P}(0) \tag{267}$$

This vector is a discrete version of the function $P(X,t)$ used previously and it gives the distribution of our systems after l Markov steps, i.e there will be $NP(i,l)$ systems in state i in the sense defined above.
It can be shown that any transition matrix T has the following mathematical properties. All its eigenvalues λ_i are real and smaller in magnitude than 1 with the exception of one eigenvalue, λ_0, that is exactly 1. This last property is actually easy to see. By construction

all the columns of T sum to 1, $\sum_j T_{j,i} = 1$, Together with the detailed balance condition we obtain

$$\sum_j T_{i,j} p_j = \sum_j T_{j,i} p_i = p_i \tag{268}$$

Hence $\mathbf{p}$ is an eigenvector of T with eigenvalue 1.
Our initial state $\mathbf{P}(0)$ can be written as a linear combination of the eigenvectors $\mathbf{v}_k$ of T

$$\mathbf{P}(0) = \sum_k c_k \mathbf{v}_k \tag{269}$$

In terms of the $\mathbf{v}_k$'s the vector $\mathbf{P}(l)$ is given by

$$\mathbf{P}(l) = \sum_k c_k \lambda_k^l \mathbf{v}_k \tag{270}$$

Since all eigenvalues are smaller in magnitude than 1 except λ_0 we obtain

$$\lim_{l\to\infty} \mathbf{P}(l) = c_0 \mathbf{v}_0 \tag{271}$$

The rate of convergence to $\mathbf{v}_0$ depends however on the eigenvalue that is second largest in magnitude. If we call this eigenvalue λ_1 we have the condition that

$$\lambda_1^l \leq \varepsilon \tag{272}$$

in order to get the equilibrium distribution with an error of less than ε. As a consequence the convergence to the final equilibrium distribution will be slow if there is an eigenvalue λ_1 close to 1.

Let us now come back to our example with the two trial matrices ω. The eigenvector $\mathbf{v}_0$ of the two transition matrices constructed according to Eq. 266 with the two trial matrices is of course identical as it should be and represents the Boltzmann distribution. However λ_1 is different in the two cases, approximately .9 for the first trial matrix and .8 for the second in the case of $\beta = 1$. Hence equilibrium is obtained roughly two times faster with the second transition than with the first one. This is not surprising. The second matrix has a much higher connectivity that allows transitions between all states. This should speed up the equilibration process.

Exercise [3pt]: *Perform a Markov process with the two transition matrices (Eq. 266 together with the two trial step probabilities from the previous pages) and verify that you get faster convergence for the trial matrix with the high connectivity. First verify by performing a large number of time steps about (10000) that you obtain in both cases the Boltzman distribution at temperature 1. Then start 10000 different (i.e with different random numbers) walkers from the same initial state* $i = 10$ *and perform only 20 Metropolis steps (Eq. 264) for both trial matrices. Plot the histogram of the distribution for the 2 cases and compare with the Boltzmann distribution.*

The Metropolis algorithm plays a central role in simulations in statistical mechanics, where one has to calculate quantities such as the energy in a canonical ensemble

$$< E >_{canonical} = \frac{\sum_Y e^{-\beta E(Y)} E(Y)}{\sum_Y e^{-\beta E(Y)}} \tag{273}$$

For the case of a continuous system the sum is actually a high-dimensional integral. If we have configurations X that are distributed according to the Boltzmann distribution the internal energy is simply given by the sum (or integral) over the N configurations.

$$< E >_{canonical} = \frac{1}{N} \sum_X E(X) \tag{274}$$

In a simulation, we calculate the above ensemble average again as a 'time' average

$$< E >_{canonical} = \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} E(X(t)) \tag{275}$$

where $X(t)$ is the 'trajectory' of a walker. In the above sum all the configurations X for which the escape steps fail once or m times because the new configurations X' are rejected, are included twice or $m+1$ times in the sum. This slows down the equilibration process. To avoid summing over the same configuration many times, the trial escape steps should be chosen such that they have a reasonable chance of success.

Error estimates for expectation values obtained by the Metropolis algorithm

The error in a stochastic process decreases like $\frac{\sigma}{\sqrt{N}}$. So one might expect that the error in a quantity that is calculated with a random distribution generated by a Markov process is given by the same expression. For instance one might think that the average error of Eq. 275 is given by

$$\frac{\sigma}{\sqrt{T}} \tag{276}$$

where

$$\sigma = \sqrt{< E^2_{canonical} > - < E_{canonical} >^2} \tag{277}$$

and

$$< E >_{canonical} = \frac{1}{T} \sum_{t=1}^{T} E(X(t)) \tag{278}$$

$$< E^2 >_{canonical} = \frac{1}{T} \sum_{t=1}^{T} E(X(t))^2 \tag{279}$$

This error estimate is much too optimistic. This can easily be seen by considering the case where the Monte Carlo simulation is trapped in a configuration. In such a case exactly the

same energy is included many times in the expectation values, but the error clearly does not decrease as predicted by the above formula. The reason why the above formulas are not valid is because a Markov process contains correlations that are not present in a truely random process. The correlation length can be obtained from the auto-correlation function $C(k)$

$$C(k) = \frac{< E(X(t))E(X(t+k)) > - < E(X(t)) >^2}{E(X(t))^2 > - < E(X(t)) >^2} \tag{280}$$

For a truely random process the expectation value $< E(X(t))E(X(t+k)) >$ is zero unless $k = 0$ and hence $C(k) = \delta_k$. For a Markov process $C(k)$ is not a delta function, but it decays exponentially. The correlation time T_c is defined as the time at which $C(k = T_c)$ has decayed close to zero. In calculating the error bound we have therefore to sum in Eqs. 278,279 not every term but only every T_c-th term.

PROJECT: Calculation of the magnetization in the Ising model

The $2d$ Ising model on a periodic square lattice

In the square lattice periodic Ising model we construct a square lattice of points labelled by the integers (i, j) where $i = 1, ..., L$ and $j = 1, ..., L$ giving, in total, $N = L^2$ lattice sites (or *spins*). At each lattice site sits an Ising spin $s_{i,j}$ which takes the value ± 1 (either *spin-up* or *spin-down*). This $L \times L$ lattice is then repeated periodically (see figure below) such that $s_{L+1,j} = s_{1,j}$ and $s_{i,L+1} = s_{i,1}$. The energy of the system (for an arbitrary spin configuration $\mathcal{S}$) $E_{\mathcal{S}}$ is given by

$$E_{\mathcal{S}} = -J \sum_{i}^{L} \sum_{j}^{L} (s_{i,j} s_{i+1,j} + s_{i,j} s_{i,j+1}) \tag{281}$$

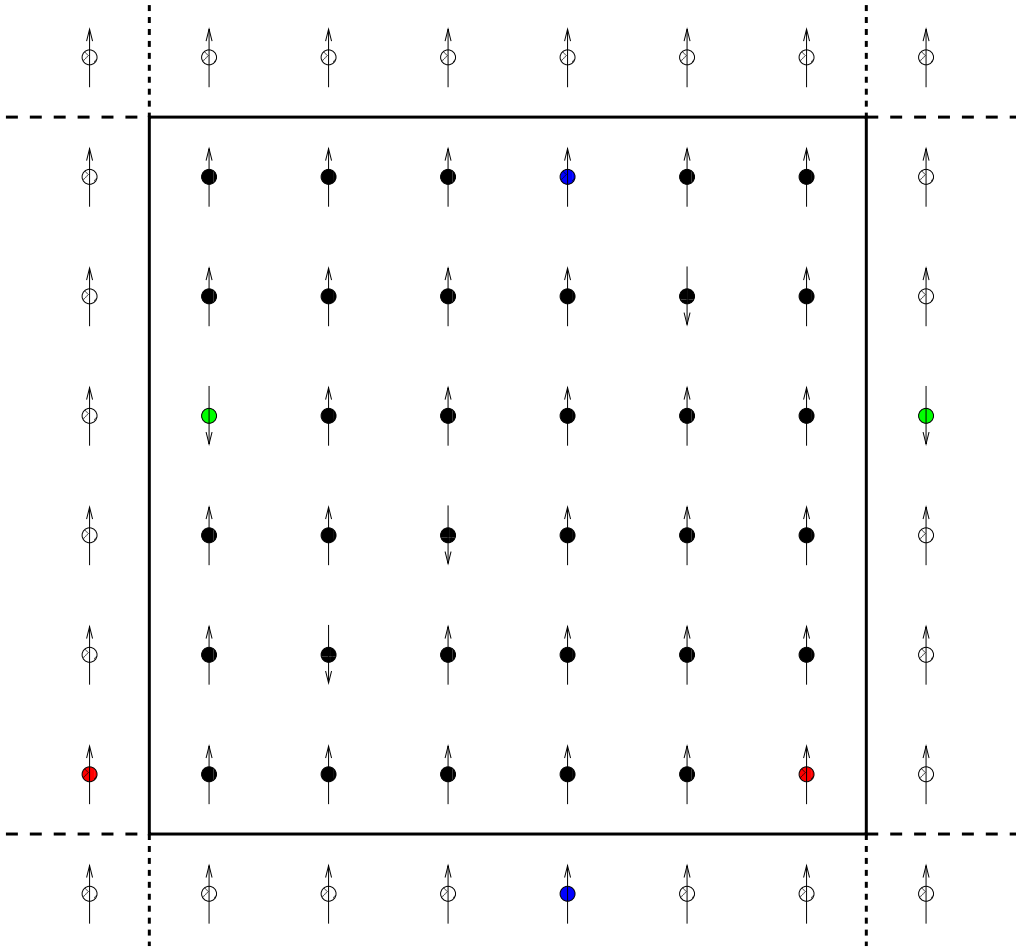
where for our purposes we may set $J = 1$ as only the ratio $E/k_{\mathrm{B}}T$ (which is dimensionless) is relevant in what follows. The average net magnetisation of the system is given by (where the sum over $\mathcal{S}$ means sum over all possible 2^N spin configurations)

$$\langle M \rangle = \sum_{\mathcal{S}} \frac{\exp(-E_{\mathcal{S}}/k_{\mathrm{B}}T) M_{\mathcal{S}}}{\mathcal{Z}} \tag{282}$$

where $\mathcal{Z} = \sum_{\mathcal{S}} \exp(-E_{\mathcal{S}}/k_{\mathrm{B}}T)$ and the net magnetisation for a given configuration is

$$M_{\mathcal{S}} = \sum_{ij} s_{i,j}. \tag{283}$$

For large L the sum over all configurations rapidly becomes prohibitive. Below a critical temperature T_c, known as the Curie temperature, the system has a net magnetisation and is termed ferromagnetic. Above T_c there is no net magnetisation and the system is in a paramagnetic phase.



Periodic 6×6 *lattice. The spins with white circles represent the spins in the neighbouring lattices (some of the periodic images are displayed with matching colours).*

Task 1

Given a spin distribution A with energy E_A we may pick a site at random and *flip* the spin at that site changing the energy to E_B. Write down a simple formula for the difference in the energies of system A and B

$$\Delta E_{A\to B} = E_A - E_B. \tag{284}$$

Hint : A maximum of 5 operations are required to compute $\Delta E_{A\to B}$.

The Metropolis Algorithm

If we pick a spin from system A, *at random*, we can work out $\Delta E_{A\to B}$ associated with changing the orientation of the spin to produce system B. According to Boltzman distribution the *probability* that the transition $A \to B$ takes place goes like

$$P \sim \exp(-\Delta E_{A\to B}/k_{\mathrm{B}}T) \tag{285}$$

where k_{B} is Boltzman's constant and T is the temperature.

The Metropolis algorithm, applied to the Ising model described in the previous section, can be summarised as follows

1. Visit all sites (i, j) consecutively. An optimal strategy is to first sweep over sites where $i + j$ is odd and then sites where $i + j$ is even as spins in each of these sets are statistically independent.

2. Calculate ΔE associated with flipping the spin at site (i, j)

3. Calculate the *probability*, at a given temperature T, of the transition taking place using $P = \exp(-\Delta E / k_{\mathrm{B}} T)$.

4. Decide if the transition takes place. This decision is made by generating another random number α:-

    - If P exceeds α, the transition is performed.
    - If P is less than α, the transition is ignored, and a new one is tried instead (return to point (1) and repeat MANY times).

Task 2
In the following a Metropolis 'sweep' will refer to an update of all spins in the system. Allow the algorithm to *equilibrate* (perform around 100000 sweeps) before sampling.

1. Implement the Metropolis algorithm for the periodic square Ising lattice described in the previous section. To test the algorithm you many use many iterations on a small, say $L = 10$ lattice. Start with all of the spins at $+1$. Try running a *low* temperature ($1.3J$) and a high temperature ($5J$). You should see the magnetisation per site ($m = M/N$) oscillate in the ranges ~ 0.8 to 1.0 and ~ -0.4 to 0.4 respectively. **For the remaining tasks use a lattice with L=50.**

2. After equilibration output to a file (this will allow the rest of the tasks to be completed without repeating the Metropolis calculation) M and m after each sweep for temperatures $1.9J$, $2.1J$, $2.4J$ and $2.6J$. (*Hint : The number of sweeps may be very large* > 500000).

    - Plot the net magnetisation per site m as a function of the number of Metropolis sweeps.
    - Calculate the average value of M and m from the data output for each temperature.

3. ¿From these calculations estimate the Curie temperature T_c. *Hint :* $T_c \approx 2.269J$. *Close to this temperature the Metropolis algorithm suffers "critical slowing down" and the number of required sweeps will become VERY large.*

4. We may define the autocorrelation as

$$A(j) = \langle M^k M^{k+j} \rangle - \langle M^k \rangle \langle M^{k+j} \rangle \tag{286}$$

    - Using the output for M (for temperatures $1.9J$, $2.1J$, $2.4J$ and $2.6J$) plot $A(j)$. As $A(j) \propto \exp -j/\tau$ find a value of τ for each of these temperatures.

5. The calculation of exponentials in the Metropolis algorithm is computationally expensive. For the above algorithm devise a way to avoid calculating exponentials in the inner Metropolis loop. How much faster is this improved algorithm?

10 Numerical solution of the single particle Schrödinger equation

The time independent single particle Schrödinger equation is given by

$$\mathcal{H}\phi_i(\mathbf{r}) = \varepsilon_i\phi_i(\mathbf{r}) \tag{287}$$

where

$$\mathcal{H} = -\frac{1}{2}\nabla^2 + V(\mathbf{r}) \tag{288}$$

The eigenvalue ε_i gives the energy of the i-th state
The Hamiltonian in Eq. 288 is expressed in atomic units. These units are formally obtained by setting $\hbar = m_e = e = \kappa_0 = 1$, where $\kappa_0 = 4\pi\varepsilon_0$. In this way important atomic properties have unit values

- charge of an electron = 1 (instead of 1.60 $\times 10^{-19}$ C)
- mass of an electron = 1 (instead of 9.11 $\times 10^{-31}$ kg)
- Angular momentum, $\hbar$ = 1 (instead of 1.05 $\times 10^{-34}$ J s
- Bohr radius of hydrogen atom $a_0 = \frac{\hbar^2}{m_e e^2}$ = 1 (instead of .529 $\times 10^{-10}$ m)

- Ground state energy of hydrogen atom $-\frac{1}{2}\frac{m_e e^4}{\hbar^2} = -\frac{1}{2}$ (instead of $-\frac{1}{2}$ 4.36×10^{-18} J)

Because many other atomic and molecular properties are related to the above quantities, they will also have numerical values that are of the order of unity. For instance

- Bond lengths are of the order of the Bohr radius
- The binding energy of a molecule is typically a fraction of the ground state energy of the hydrogen atom
- The electric dipole moment of a molecule is typically of the order of $ea_0 = 1$ (instead of 8.45×10^{-30} C m)

10.1 Discretization of the single particle Schrödinger equation

A computer can not directly do calculations on continuous functions. A continuous function has therefore to be parametrized by a finite number of discrete parameters. This leads to a discretization of the equations that the functions have to satisfy. To discretize the Schrödinger equation one expresses in most cases the eigen-functions as a linear combination of a set of so-called basis functions $U_k(\mathbf{r})$.

$$\phi_i(\mathbf{r}) = \sum_k u_{k,i} U_k(\mathbf{r}) \tag{289}$$

Popular types of basis functions will soon be discussed. Substituting Eq. 289 into Eq. 287 one obtains

$$\mathcal{H} \sum_k u_{k,i} U_k(\mathbf{r}) = \varepsilon_i \sum_k u_{k,i} U_k(\mathbf{r}) \tag{290}$$

Multiplying from the left by $U_l(\mathbf{r})$ and integrating, one obtains

$$\sum_k u_{k,i} \int U_l(\mathbf{r}) \mathcal{H} U_k(\mathbf{r}) d\mathbf{r} = \varepsilon_i \sum_k u_{k,i} \int U_l(\mathbf{r}) U_k(\mathbf{r}) d\mathbf{r} \tag{291}$$

Introducing the Hamiltonian matrix $H_{l,k}$

$$H_{l,k} = \int U_l(\mathbf{r}) \mathcal{H} U_k(\mathbf{r}) d\mathbf{r} \tag{292}$$

and the overlap matrix $S_{l,k}$

$$S_{l,k} = \int U_l(\mathbf{r}) U_k(\mathbf{r}) d\mathbf{r} \tag{293}$$

we finally get the following eigenvalue problem

$$\sum_k H_{l,k} u_{k,i} = \varepsilon_i \sum_k S_{l,k} u_{k,i} \tag{294}$$

Eq. 294 is the discretized version of the continuous Schrödinger equation Eq. 287. Introducing a vector $\vec{u}_i$ that contains the expansion coefficients of the i-th eigenfunction Eq. 294 can be rewritten in matrix vector notation

$$H\vec{u}_i = \varepsilon_i S\vec{u}_i \tag{295}$$

The matrices H and S are symmetric, S is positive definite. Eq. 295 is a generalized eigenvalue problem. The difference to a standard eigenvalue problem is simply that the matrix S is not the unit matrix.

Exercise [3pt]: *Prove for the 1-dim case Schrödinger equation that S and H (Eq. 293,292) are symmetric. Prove that S is positive definite.*

10.2 The variational principle

The variational principle states that the ground state wave function ϕ_0 is the wave-function ϕ which minimizes the energy expectation value ε

$$\varepsilon = \int \phi(\mathbf{r}) \mathcal{H} \phi(\mathbf{r}) \, d\mathbf{r} \tag{296}$$

under the normalization constraint

$$\int \phi(\mathbf{r}) \phi(\mathbf{r}) \, d\mathbf{r} = 1 \tag{297}$$

We will prove the discretized version of the variational principle which reads: the ground state vector $\vec{u}_0$ is the vector $\vec{u}$ which minimizes the discrete energy expectation value ε

$$\varepsilon = \vec{u}^T \, H \, \vec{u} \tag{298}$$

under the normalization constraint

$$\vec{u}^T \, S \, \vec{u} = 1 \tag{299}$$

Exercise [2pt]: *Show that Eq. 298 and Eq. 299 are obtained from Eq. 296 and Eq. 297 by using Eq. 289.*

Proof : For the generalized symmetric eigenvalue problem with a positive definite S

$$H\vec{v}_i = \varepsilon_i S\vec{v}_i \tag{300}$$

the eigenvectors v_i form a complete set of vectors with the properties that

$$\vec{v}_i^T S\vec{v}_j = \delta_{i,j} \qquad ; \qquad \vec{v}_i^T H\vec{v}_j = \delta_{i,j}\varepsilon_i \tag{301}$$

Hence we can expand our ground state vector in terms of the eigenvectors

$$\vec{u}_0 = \sum_i c_i \vec{v}_i \tag{302}$$

The normalization condition then becomes

$$\sum_i c_i^2 = 1 \tag{303}$$

and the expectation value for the energy is given by

$$\varepsilon = \sum_i \varepsilon_i c_i^2 \tag{304}$$

Since the eigenvalues are ordered by increasing value, the minimum is obtained if $c_0 = 1$ with all other coefficients being zero. Hence $\varepsilon = \varepsilon_0$ and $\vec{u}_0 = \vec{v}_0$

10.3 Numerical utilization of the variational principle

Since the ground state wave-function is the one that minimizes the energy ε_0, the solution of Schrödinger's equation can be considered as a minimization problem of the following expression which includes the normalization constraint:

$$\frac{1}{2}\frac{\vec{u}^T H\vec{u}}{\vec{u}^T S\vec{u}} \tag{305}$$

Its gradient g with respect to $\vec{u}$ is

$$\vec{g} = H\vec{u} - \varepsilon S\vec{u} \tag{306}$$

where $\varepsilon = \frac{\vec{u}^T H\vec{u}}{\vec{u}^T S\vec{u}}$ and u is normalized such that $\vec{u}^T S\vec{u} = 1$.
Exercise [1pt]: *Verify that Eq. 306 is the gradient of Eq. 305*
The condition that the gradient vanishes is thus equivalent to the eigenvalue problem. Alternatively, ε in Eq. 306 can be considered as a Lagrange multiplier that enforces normalization if the unconstrained gradient of $\frac{1}{2}\vec{u}^T H\vec{u}$ is used. The Lagrange multiplier point of view is more general and will be used later on.
It can be shown that the Hessian matrix A for the search of the ground state ε_0, u_0 is diagonal in the basis of the eigenvectors of H and that it has the diagonal elements D_i

$$D_i = \varepsilon_i - \varepsilon_0 \tag{307}$$

Even though the eigenvalues and eigenvectors are unknown when one starts solving Schrödinger's equation, Eq. 307 is useful for the construction of approximate Hessians for preconditioning purposes.

Generalizations of the variational principle

- The variational principle does not only hold if the wave-function is written as a linear combination of basis functions (Eq. 289) but also for any nonlinear parameterization of a wave-function.
- The variational principle can be generalized to excited states. It can be shown that the M-th excited state u minimizes the energy expectation values

$$\varepsilon = \sum_{l,k} u_l \, H_{l,k} \, u_k \tag{308}$$

under the normalization constraint

$$\sum_{l,k} u_l \, S_{l,k} \, u_k = 1 \tag{309}$$

and the additional constraint that u is orthogonal to all $M-1$ lower eigenstates u_i, $i = 1, .., M-1$

$$\sum_{l,k} u_l \, S_{l,k} \, u_{k,i} = 0 \tag{310}$$

10.4 Independent particle methods: Density Functional Theory

The hydrogen atom is the only one-electron system found in nature. All other atoms, molecules and solids are many electron systems and should therefore be described by a many electron Schrödinger equation. The solution of the many electron Schrödinger equation is numerically extremely expensive and for this reason the many electron Schrödinger equation is frequently approximated by so-called independent particle methods. These independent particle schemes give rise to a set of single particle Schrödinger equations with a modified potential. Whereas in the simple single particle Schrödinger equation the potential contains only external potentials (such as the potential of the nuclei $V_{en}(\mathbf{r})$), the potential in an independent particle scheme contains an additional part that describes the influence of the other electrons. The most popular independent particle scheme is density functional theory. Within this theory this additional potential is called the exchange correlation potential. We will not dwell onto the theoretical background of density functional theory, but only present the resulting Kohn-Sham equations that need to be solved.
Even though there exists an existence proof for the exchange correlation potential, its explicit form is unknown. For this reason various approximate forms are used. The most basic one is called the local density approximation (LDA). The functional used in this context depends only locally on the density and it has the property that it is exact for a constant electron density. In a real atom or molecule the density is of course not constant but varies. In spite of this the LDA approximation is surprisingly accurate. Within the

local density approximation of density functional theory, the total energy of a system of N electrons is given by

$$E = -\sum_{i=1}^{N} \frac{1}{2} \int \phi_i^*(\mathbf{r}) \nabla^2 \phi_i(\mathbf{r}) d\mathbf{r} + \int V_{en}(\mathbf{r}) \rho(\mathbf{r}) d\mathbf{r} + \frac{1}{2} \int \frac{\rho(\mathbf{r}) \rho(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}' d\mathbf{r} + E_{xc}[\rho(\mathbf{r})] \tag{311}$$

where the charge density $\rho(\mathbf{r})$ is the sum over the square of all the occupied Kohn-Sham orbitals ϕ_i

$$\rho(\mathbf{r}) = \sum_{i=1}^{N} \phi_i^*(\mathbf{r}) \phi_i(\mathbf{r}) \tag{312}$$

The first term in Eq. 311 is the kinetic energy of N independent electrons, the second the interaction of the electrons with the nuclei and potentially other external potentials, the third the classical electron-electron repulsion and the last the exchange correlation energy. The Kohn-Sham equations are obtained by minimizing the total energy expression Eq. 311 (multiplied for convenience by $1/2$) under the constraint that the orbitals ϕ_i are orthonormal. Using the Euler Lagrange formalism, it can be shown that the unconstrained gradient $d_i(\mathbf{r})$ is

$$d_i(\mathbf{r}) = -\frac{1}{2} \nabla^2 \phi_i(\mathbf{r}) + V_{en}(\mathbf{r})\, \phi_i(\mathbf{r}) + \int \frac{\rho(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}'\, \phi_i(\mathbf{r}) + v_{xc}(\rho(\mathbf{r}))\, \phi_i(\mathbf{r}) \tag{313}$$

where the exchange correlation potential is defined as

$$v_{xc}(\rho(\mathbf{r})) = \frac{\delta E_{xc}(\rho(\mathbf{r}))}{\delta \rho(\mathbf{r})} \tag{314}$$

Defining a LDA Kohn-Sham Hamiltonian as

$$\mathcal{H}_{KS} = -\frac{1}{2}\nabla^2 + V_{en}(\mathbf{r}) + V_H(\mathbf{r}) + v_{xc}(\rho(\mathbf{r})) \tag{315}$$

where $V_H = \int \frac{\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}'$ is the Hartree potential, the unconstrained gradient of Eq. 313 can be re-expressed as

$$d_i(\mathbf{r}) = \mathcal{H}_{KS}\,\phi_i(\mathbf{r}) \tag{316}$$

It can also be shown that the orthogonality constraints can be imposed by Lagrange multipliers and the condition that the constrained gradient vanishes becomes

$$d_i(\mathbf{r}) - \sum_{j=1}^{N} \Lambda_{i,j}\phi_j(\mathbf{r}) = 0 \tag{317}$$

where

$$\Lambda_{i,j} = \int \phi_j^*(\mathbf{r}) d_i(\mathbf{r}) d\mathbf{r} = \int \phi_i(\mathbf{r}) d_j^*(\mathbf{r}) d\mathbf{r} \tag{318}$$

and ϕ_i is a set of orthonormal wave functions, i.e $\int \phi_i^*(\mathbf{r})\phi_j(\mathbf{r})d\mathbf{r} = \delta_{i,j}$. Using Eq. 316, the condition that the constrained gradient vanishes (Eq.317) can then be written as

$$\mathcal{H}_{KS}\phi_i(\mathbf{r}) - \sum_{j=1}^{N} \Lambda_{i,j}\phi_j(\mathbf{r}) = 0 \tag{319}$$

Exercise [1pt]: *Verify that Λ is a symmetric matrix, i.e. that Eq. 318 holds true*
The set of orbitals ϕ_i that satisfies Eq. 319 is not unique. This comes from the fact that the LDA energy expression (Eq. 311) is invariant under unitary transformations. This means that if we have one set of orbitals ϕ_i, any other set $\tilde{\phi}_i$ where

$$\tilde{\phi}_i = \sum_{j=1}^{N} U_{i,j}\phi_j \tag{320}$$

will give the same energy if U is an unitary matrix. The demonstration of the correctness of this statement follows from the fact that the energy in Eq. 311 depends only on the charge density ρ and the kinetic energy. Both are invariant under unitary transformations of the orbitals. Let's demonstrate this explicitly for the charge density, the demonstration for the kinetic energy is analogous. Lets call the charge density obtained from the new set

of orbitals $\tilde{\phi}_i$ $\tilde{\rho}$. Then we have

$$\begin{aligned}
\tilde{\rho}(\mathbf{r}) &= \sum_{l=1}^{N} \tilde{\phi}_l^*(\mathbf{r})\tilde{\phi}_l(\mathbf{r}) && (321)\\
&= \sum_{i,j} \phi_i^*(\mathbf{r})\phi_j(\mathbf{r}) \sum_l U_{l,i}^* U_{l,j} && (322)\\
&= \sum_{i,j} \phi_i^*(\mathbf{r})\phi_j(\mathbf{r}) \sum_l U_{i,l}^H U_{l,j} && (323)\\
&= \sum_{i,j} \phi_i^*(\mathbf{r})\phi_j(\mathbf{r})\delta_{i,j} && (324)\\
&= \sum_{i} \phi_i^*(\mathbf{r})\phi_i(\mathbf{r}) = \rho(\mathbf{r}) && (325)
\end{aligned}$$

Because of this invariance under unitary transformations of the orbitals we may choose so-called canonical orbitals which give rise to a diagonal matrix Λ. Denoting the diagonal elements of Λ by ε_i Eq. 319 becomes

$$\mathcal{H}_{KS}\phi_i(\mathbf{r}) - \varepsilon_i\phi_i(\mathbf{r}) = 0 \qquad (326)$$

This equation resembles very much the single particle Schrödinger's equation. As already pointed out the difference is that the potential of the Kohn-Sham Hamiltonian consists of

the external potential V_{en} plus the so-called Hartree potential $V_H(\mathbf{r}) = \int \frac{\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}'$ and the exchange correlation potential v_{xc}. The Hartree potential describes the classical repulsion of charged particles whereas v_{xc} gives all the quantum mechanical corrections to this classical repulsion. Even though Eq. 319 looks like an eigenvalue problem, it is not a standard eigenvalue problem. The reason for this is that the Kohn-Sham Hamiltonian depends on the eigen-orbitals ϕ_i, i.e on the solution of the apparent eigenvalue problem. For this reason the energy (Eq. 311) is also not equal to the sum of the Kohn-Sham eigenvalues ε_i as it would be for non-interacting electrons.

Since independent particle schemes do not lead to a standard eigenvalue problem, but to a more complicated self-consistent eigenvalue problem, we will consider the solution of the Kohn-Sham equations (Eq. 326) as a minimization problem.

Eq. 326 can be discretized in the same way as we did with Eq. 287. In analogy to Eq. 306 we obtain

$$H_{KS}\vec{u}_i - \varepsilon_i S\vec{u}_i = 0 \tag{327}$$

where H_{KS} is the Kohn-Sham matrix and $\vec{u}_i$ is the vector that contains the expansion coefficients of the i-th orbital (Eq. 289). Since the Hartree and exchange correlation potential in H_{KS} depend on the charge density, H_{KS} has to be recalculated in any step of a minimization algorithm.

Up to now we have neglected spin effects. Because of the Pauli principle, the orbitals of different electrons have to be orthogonal. The orthogonality is automatically assured if

the electrons have different spins. In the case of the helium atom we have for instance 2 electrons, both of which have the same spatial Kohn-Sham orbital, but which differ in their spin. This is the simplest example of a closed shell system. A closed shell system consists of an even number of electrons, and the electrons pairwise share spatial orbitals. The majority of stable molecules and solids belongs to this type. For a closed shell system the charge density of the spin up and spin down electrons is the same and the total charge density can consequently be obtained by summing only over all the different spatial orbitals and multiplying by 2:

$$\rho(\mathbf{r}) = 2\sum_{i=1}^{N/2} \phi_i^*(\mathbf{r})\phi_i(\mathbf{r}) \tag{328}$$

In the same way the kinetic energy is given by

$$-2\sum_{i=1}^{N/2} \frac{1}{2}\int \phi_i^*(\mathbf{r})\nabla^2\phi_i(\mathbf{r})d\mathbf{r} \tag{329}$$

In addition there are however also so-called spin polarized systems for which the spatial part of the spin up orbitals $\phi_i^{\uparrow}$ is not equal to the spatial part of the spin down orbitals $\phi_i^{\downarrow}$. This is necessarily the case if the number of electrons is odd. One obtains then a up-spin

and down-spin charge density

$$\rho^{\uparrow}(\mathbf{r}) = \sum_{i=1}^{N^{\uparrow}} \phi_i^{\uparrow *}(\mathbf{r})\phi_i^{\uparrow}(\mathbf{r}) \qquad ; \qquad \rho^{\downarrow}(\mathbf{r}) = \sum_{i=1}^{N^{\downarrow}} \phi_i^{\downarrow *}(\mathbf{r})\phi_i^{\downarrow}(\mathbf{r}) \tag{330}$$

where $N^{\uparrow}$ and $N^{\downarrow}$ is the number of spin-up and down-electrons. The total charge density ρ is obviously the sum of both, i.e.

$$\rho(\mathbf{r}) = \rho^{\uparrow}(\mathbf{r}) + \rho^{\downarrow}(\mathbf{r}) \tag{331}$$

For such a system one has then to use in Eq. 311 a spin polarized version of the exchange correlation energy

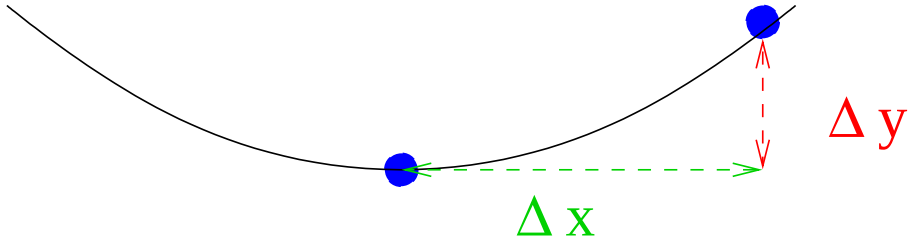
$$E_{xc} = E_{xc}(\rho^{\uparrow}(\mathbf{r}), \rho^{\downarrow}(\mathbf{r})) \tag{332}$$

and instead of Eq. 326 one obtains two sets of Kohn-Sham equations for both the spin up and spin down orbitals. On the spin-up electrons the spin up exchange correlation potential $v_{xc}^{\uparrow}(\rho(\mathbf{r}))$ is acting whereas on the spin-down electrons the spin down counterpart $v_{xc}^{\downarrow}(\rho(\mathbf{r}))$ is acting. The Hartree energy and external potential energy depend also in the case of a spin polarized system only on the total charge density $\rho(\mathbf{r})$

Accuracy of the discretized solution of Schrödinger's equation

The exact solution of the discretized Schrödinger's equation Eq. 294 is always an approximate solution of the exact continuous equation Eq. 288. The difference in the eigenvalues of the discrete and continuous equations is called the discretization error. The discretization error decreases with an increasing number of basis functions and tends to zero in the limit of an infinitely large systematic basis set. A systematic basis set is by definition just a basis set with this property , namely that the continuous eigenvalue ε can be approximated with any desired accuracy. All orthogonal basis sets, i.e basis sets for which $S_{i,j} = \delta_{i,j}$ are systematic basis sets. Because of the variational principle the discrete eigenvalue is always bigger than the continuous eigenvalue. This can be easily seen by the following argument. Let us compare the eigenvalue $\varepsilon(m)$ obtained by using a basis set of m basis functions and of another one, $\varepsilon(n)$, obtained by using a second set of n functions. We assume that $n > m$ and that the first m functions in the basis set containing n functions are identical to the m basis function of the smaller set. Hence any solution that can be represented by the smaller basis set can also be represented by the larger basis set. Since one has more degrees of freedom one can better minimize the wave function represented by the larger basis set and consequently $\varepsilon(n) < \varepsilon(m)$. Since $\varepsilon(n)$ tends to the continuous eigenvalue ε if n is infinitely large we have as well that $\varepsilon < \varepsilon(m)$.

Let us now assume that we have solved the discretized Schrödinger equation in a finite basis set. Because of the variational property, the error in the eigenvalue (Δy) is smaller than the error (Δx) of the wave-function. This can be seen from the figure below



$$\Delta y \propto (\Delta x)^2 \tag{333}$$

In other words, it is possible to get fairly good energies with a basis set that is too small to represent the wave-function with high accuracy.

10.5 The hydrogen atom

We will consider the slightly more general problem of a single electron orbiting around a nucleus of charge Z. Obviously the H atom has $Z = 1$. This is the only atomic or molecular system found in nature for which an analytic solution is known. Since it allows us to check any numerical solution against the analytic solution it will serve as a starting point for numerical work. In addition the fact of having radial symmetry and no electron electron interactions leads to considerable simplifications. The Hamiltonian is

$$\mathcal{H} = -\frac{1}{2}\nabla^2 - \frac{Z}{|\mathbf{r}|} \tag{334}$$

As is well known from elementary quantum mechanics, the eigenstates are characterized by the 3 quantum numbers n, l, m and can be written as a product of spherical harmonics $Y_{l,m}(\hat{\mathbf{r}})$ and radial functions $R_{n,l}(r)$

$$\phi(\mathbf{r})_{n,l,m} = R_{n,l}(r)Y_{l,m}(\hat{\mathbf{r}}) \tag{335}$$

The radial functions satisfy the differential equation

$$\left[-\frac{1}{2}\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d}{dr}\right) + \frac{l(l+1)}{2r^2} - \frac{Z}{r}\right] R_{n,l}(r) = E_{n,l}R_{n,l}(r) \tag{336}$$

Both the angular and radial parts form sets of orthonormal functions

$$\int R_{n,l}(r)R_{n',l}(r)\,r^2 dr = \delta_{n,n'} \quad ; \quad \int Y^*_{l,m}(\hat{\mathbf{r}})Y_{l,m}(\hat{\mathbf{r}})d\Omega = \delta_{l,l'}\delta_{m,m'} \tag{337}$$

where $d\Omega$ indicates integration over the surface of the unit sphere. The first radial functions are listed below.

$$R_{1,0} = 2Z^{\frac{3}{2}}\,e^{-Zr} \tag{338}$$

$$R_{2,0} = 2\left(\frac{Z}{2}\right)^{\frac{3}{2}}\left(1-\frac{Zr}{2}\right)\,e^{-Zr/2}$$

$$R_{2,1} = \frac{1}{\sqrt{3}}\left(\frac{Z}{2}\right)^{\frac{3}{2}} Zr\,e^{-Zr/2} \tag{339}$$

$$R_{3,0} = 2\left(\frac{Z}{3}\right)^{\frac{3}{2}}\left(1-\frac{Zr}{3}+\frac{2\,(Zr)^2}{27}\right)\,e^{-Zr/3}$$

$$R_{3,1} = \frac{4\sqrt{2}}{3}\left(\frac{Z}{3}\right)^{\frac{3}{2}} Zr\left(1-\frac{Zr}{6}\right)\,e^{-Zr/3}$$

$$R_{3,2} = \frac{2\sqrt{2}}{27\sqrt{5}}\left(\frac{Z}{3}\right)^{\frac{3}{2}} (Zr)^2\,e^{-Zr/3}$$

The first spherical harmonics are given by

$$\begin{aligned} Y_{0,0} &= \sqrt{\frac{1}{4\pi}} \\ Y_{1,0} &= \sqrt{\frac{3}{4\pi}}\cos(\theta) \\ Y_{1,-1} &= -\sqrt{\frac{3}{4\pi}}\sin(\theta)\exp(-I\phi) \\ Y_{1,1} &= \sqrt{\frac{3}{4\pi}}\sin(\theta)\exp(I\phi) \end{aligned}$$

For the radial differential equation 336 there is also a variational quantity. Using the Euler-Lagrange formalism it can be shown, that solving Eq. 336 is equivalent to minimizing the quantity

$$E = \int \left[\frac{1}{2} \left(r \frac{d\,R(r)}{dr} \right)^2 + R(r) \frac{l(l+1)}{2} R(r) - R(r) Z r R(r) \right] dr \tag{340}$$

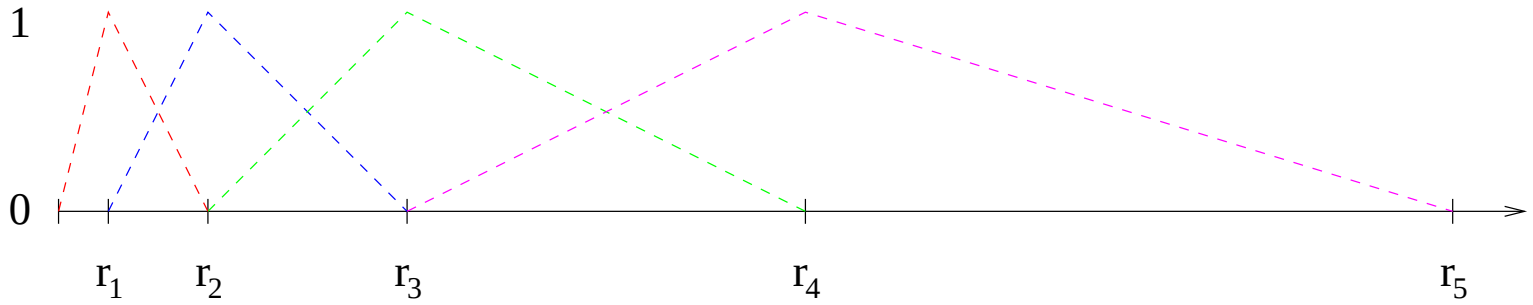
under the appropriate orthogonality constraints.

Finite elements for the radial hydrogen equation

We will use first order finite elements to discretize the radial wave-function $R(r)$.

$$R_i(r) = \sum_k u_i(k) U_k(r) \tag{341}$$

The finite element functions $U_1(r)$, $U_2(r)$, $U_3(r)$, $U_4(r)$ attached to the radial grid r_1, ...,r_4 are shown below



They have the property of being one at the central grid point and tend linearly to zero towards the two neighboring grid points. Beyond the neighboring grid points they are identically zero. Thus the expansion coefficients $u(k)$ in Eq. 341 are identical to the value of the function at the grid point r_k. Consequently the function R in each interval $[r_k : r_{k+1}]$ is given by

$$R(r) = u(k) + \frac{u(k+1) - u(k)}{r_{k+1} - r_k}(r - r_k) \tag{342}$$

Using Eq. 342 rather than Eq. 341 as the basic definition of our radial wave-function is

actually advantageous. With Eq. 342 the radial wave-function can take on a non-zero value $u(0)$ at the origin $r_0 = 0$, whereas Eq. 341 would impose a zero value.
In our finite element basis the energy expression of Eq. 340 becomes

$$\begin{aligned} E &= \frac{1}{2}\sum_{l=0}^{M-1}\int_{r_l}^{r_{l+1}}\left(r\frac{u(l+1)-u(l)}{r_{l+1}-r_l}\right)^2 dr \\ &+ \frac{l(l+1)}{2}\sum_{l=0}^{M-1}\int_{r_l}^{r_{l+1}}\left(u(l)+\frac{u(l+1)-u(l)}{r_{l+1}-r_l}(r-r_l)\right)^2 dr \\ &- Z\sum_{l=0}^{M-1}\int_{r_l}^{r_{l+1}}\left(u(l)+\frac{u(l+1)-u(l)}{r_{l+1}-r_l}(r-r_l)\right)^2 r\,dr \end{aligned} \tag{343}$$

The unconstrained gradient $d(k)$ of the above energy expression is obviously given by $\frac{\partial E}{\partial u(k)}$. The constrained gradient of Eq. 294 requires the radial overlap matrix. A convenient way to obtain the overlap matrix is obtained from the identity.

$$\begin{aligned} S(k,l) &= \frac{1}{2}\frac{\partial}{\partial u(k)}\frac{\partial}{\partial u(l)}\int R(r)R(r)\,r^2\,dr \\ &= \frac{1}{2}\frac{\partial}{\partial u(k)}\frac{\partial}{\partial u(l)}\sum_{m=0}^{M-1}\int_{r_m}^{r_{m+1}}\left(u(m)+\frac{u(m+1)-u(m)}{r_{m+1}-r_m}(r-r_m)\right)^2 r^2\,dr \end{aligned} \tag{344}$$

In particular, this way of calculating S avoids complications that would arise with the standard definition because we would like to add "half" a finite element in the interval $[r_0 : r_1]$.

Exercise [2pt]: *Show that the calculation of the overlap matrix through Eq. 344 is equivalent to the definition given in Eq. 293*

Let us now discuss the choice of the radial grid r_k. We know that the radial functions $R_{n,l}(r)$ vary much faster close to the origin than far away. Therefore the resolution near the origin should be higher, i.e the distance between the grid points smaller. There are many recipes for constructing grids with this property. A widely used grid is the so-called logarithmic grid

$$r_k = a\,\exp(\alpha k) - a \qquad\qquad k = 0,\ ...,\ \mathrm{M} \tag{345}$$

The constant a determines the distance of r_1 form the origin. It should be a small fraction of the extent of the least extended radial orbital. So let's put $a = 1/1000$. The largest radial grid point should be at a distance where the most extended orbital has decayed to a tiny value. Let's put $r_M = A = 100$. How many grid points M we can afford now determines α

$$\alpha = \ln\left(\frac{A+a}{a}\right)/M \tag{346}$$

At the end of computational interval we impose the boundary conditions $R(r_M) = 0$.

The condition number of a a free particle

When one tries to solve Schrödinger's equation by using minimization methods such as steepest descent or conjugate gradient one realizes that the convergence becomes very slow if the number of grid points is large. As we have seen before a slow convergence rate is related to a poor conditioning number. For realistic systems the condition number can not be calculated analytically, but for a free particle this can be done and the deterioration of the condition number can consequently be shown. Let us consider a free 1-dim electron in a periodic box of length L. To represent the Hamiltonian we use first order finite differences on a grid of spacing h. There shall be m grid points in the box, hence $L = mh$. The resulting Hamiltonian is

$$H_{i,j} = \begin{cases} \frac{1}{h^2} & \text{if } j = i \\ -\frac{1}{2h^2} & \text{if } j = i \pm 1 \\ 0 & \text{else} \end{cases} \tag{347}$$

This Hamiltonian is identical to the matrix of Eq. 184. The eigenvalues of this Hamiltonian are

$$\frac{1}{h^2}\left(1 - \cos(2\pi k/m)\right) \tag{348}$$

and the associated eigenvectors

$$u_k(j) = \exp\left(I2\pi\frac{kj}{m}\right) \tag{349}$$

where k runs from $-m/2+1$ to $m/2$

Exercise [1pt]: *Verify that the above equations give the eigenvalues and vectors of the Hamiltonian (347)*

The lowest eigenvalue ε_{min} equals 0, the highest eigenvalue ε_{max} equals $2/h^2$ and the eigenvalue of the first excited state ε_1 equals $(1-\cos(2\pi/m))\frac{1}{h^2}$. If m is large this eigenvalue is approximately given by $(2\pi/m)^2\frac{1}{h^2}$. According to Eq. 307 the smallest eigenvalue of the Hessian of a ground state search is then $\varepsilon_1 - \varepsilon_{min}$ and the largest eigenvalue is $\varepsilon_{max} - \varepsilon_{min}$. Hence the condition number is given by

$$\kappa = \frac{\varepsilon_{max} - \varepsilon_{min}}{\varepsilon_1 - \varepsilon_{min}} = \frac{\varepsilon_{max}}{\varepsilon_1} = \frac{m^2}{2\pi^2} \tag{350}$$

As claimed, the condition number deteriorates as the number of grid points increases. Preconditioning is therefore necessary.

Preconditioning the eigenvalue problem

As we have seen solving Schrödinger's equation is equivalent to a minimization problem. For an orthogonal basis set, the (constrained) gradient of Eq. 306 reduces to

$$\vec{g} = H\vec{u} - \varepsilon\vec{u} \tag{351}$$

Since ε is a variational quantity, it converges faster to the true eigenvalue than $\vec{u}$ converges to the true eigenvector (see Eq. 333). In our mathematical analysis we can therefore assume that ε is an exact eigenvalue. Let us assume that we have at a certain stage of our iteration the approximative eigenvector $\vec{u}$ and that the true eigenvector is given by $\vec{u}+\vec{p}$. This gives the equation

$$H(\vec{u}+\vec{p}) - \varepsilon(\vec{u}+\vec{p}) \approx 0 \tag{352}$$

Solving the above equation we get a linear system of equations for $\vec{p}$

$$(H-\varepsilon)\vec{p} = -(H-\varepsilon)\vec{u} = -\vec{g} \tag{353}$$

Eq. 353 is in principle the equation to be solved for preconditioning purposes. It gives us the preconditioned gradient $\vec{p}$ from the ordinary gradient $\vec{g}$. As it will turn out, we should however better slightly modify Eq. 353 in the numerical context. To analyze the problem, let us introduce the exact eigenvalues ε_i and eigenvectors $\vec{u}_i$, satisfying the ordinary eigenvalue problem

$$H\vec{u}_i - \varepsilon_i\vec{u}_i = 0 \tag{354}$$

The gradient $\vec{g}$ as well as the preconditioned gradient $\vec{p}$ can be written as a linear combination of these eigenvectors

$$\vec{g} = \sum_i \alpha_i \vec{u}_i \qquad ; \qquad \vec{p} = \sum_i \beta_i \vec{u}_i \tag{355}$$

Plugging these two expansions into Eq. 353 allows us to solve for the components of $\vec{p}$

$$\beta_i = -\alpha_i \frac{1}{\varepsilon_i - \varepsilon} \tag{356}$$

We see that β_i can explode whenever ε is very close to an exact eigenvalue. This can cause numerical problems if this condition is by chance encountered even though the eigenvectors are not yet very well converged. There are several possibilities to eliminate this problem. One possibility is to replace the equation for $\vec{p}$

$$\vec{p} = (H - \varepsilon I)^{-1} \vec{g} \tag{357}$$

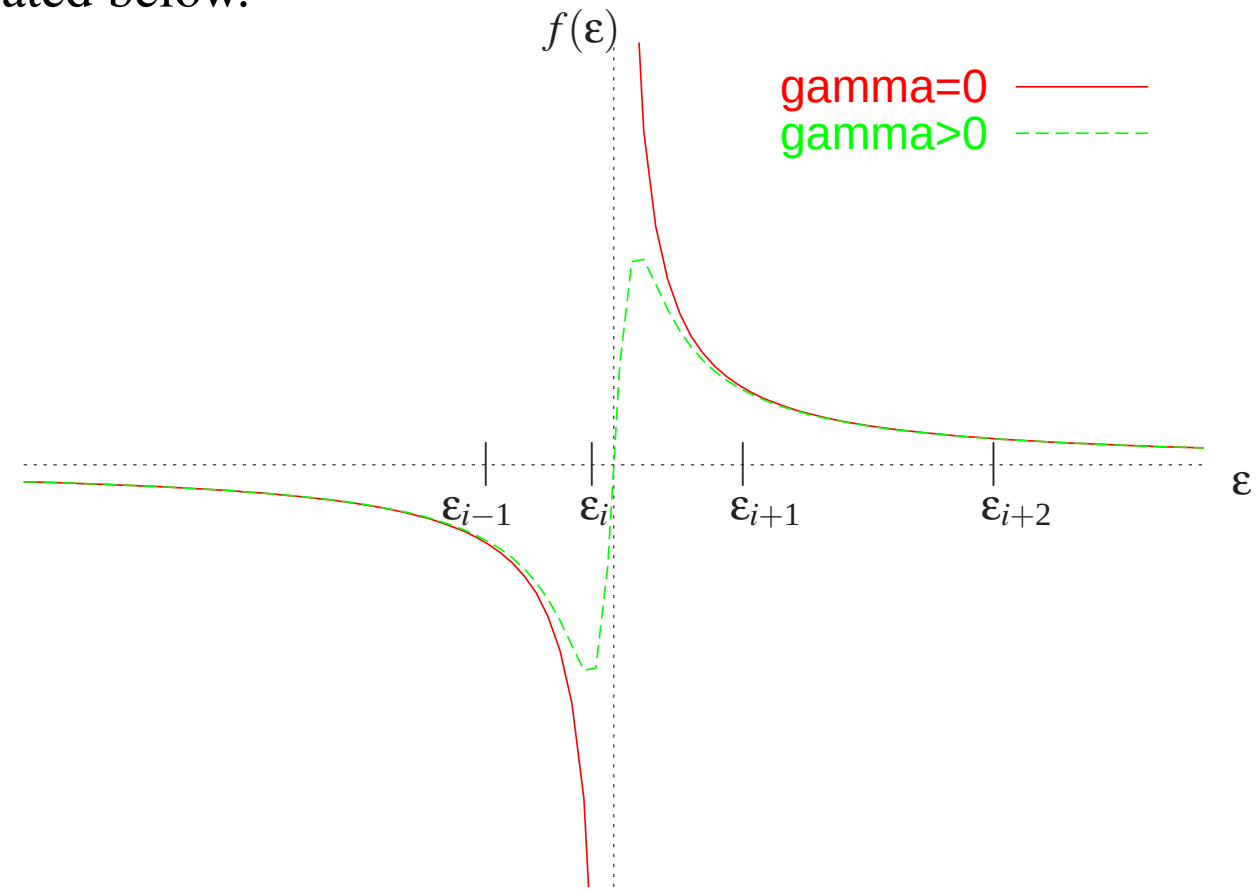
by

$$\vec{p} = Re\left[(H - (\varepsilon + i\gamma)I)^{-1} \vec{g}\right] \tag{358}$$

The introduction of this imaginary shift γ modifies Eq. 356 to

$$\beta_i = \alpha_i \frac{\varepsilon_i - \varepsilon}{(\varepsilon_i - \varepsilon)^2 + \gamma^2} \tag{359}$$

The components that are close in energy to ε can now not any more explode whereas all the components that are far away from ε are treated essentially in the same way. The situation is illustrated below.



Eq. 358 thus gives rise to powerful preconditioned steepest descent iteration

$$\vec{u} = \vec{u} - t\,\vec{p} \tag{360}$$

where t is of the order of 1

PROJECT: Calculation of the ionization potential of Helium

Physical background:
The ionization potential is the energy that is required to take away an electron from an atom or molecule. The ionization potential of helium is consequently the difference in energy between the neutral helium atom and an He^+ atom. The missing electron is supposed to be very far away with negligible kinetic energy. Hence the missing electron gives no contribution to the energy. The wave-functions of both the He and He^+ atom have pure *s* character and in both cases only one spatial orbital has to be calculated. In the case of He^+ we have a spin-polarized system, where only one electron (let's say with spin up) is occupying this orbital and in the case of He we have a closed shell where a pair of spin-up and spin-down electrons is occupying the spatial orbital.

Tasks
In this project we will first develop a program that calculates the 1*s* orbital of H. By adding exchange correlation potentials and energies we will then pass from the single particle Schrödinger equation to the independent particle Kohn-Sham equations. This will allow us then to treat the 2-electron He atom.

Part I: The hydrogen atom

- The file http://www.unibas.ch/comphys/comphys/TEACH/WS04/ATOM/subs.f90 contains all the subroutines that are provided for this project. It contains the subroutine "energr" that calculates the energy of Eq. 343 and the gradient $\frac{\partial E}{\partial u(k)}$ for a given input vector u. The routine works for general angular components l and since we are only interested in s states "lang" has to be set to 0. The routine "overlap" calculates the overlap expectation value $O = \vec{u}^T S\vec{u}$ and its derivative $\frac{\partial O}{\partial u(k)}$. These two subroutines where generated quasi automatically by Maple. The routine radgrid generates the radial grid of Eq. 345.

- Use the exact $1s$ radial wave-function of Eq. 338 and check whether the gradient of Eq. 306 is small. It is actually not zero, since the analytical wave-function is not the exact solution for the case where the wave-function is represented by finite elements. The gradient should however tend to zero in the limit where we use more and more grid points (and consequently more and more finite elements). Check whether this is fulfilled. The energy should actually improve by a factor of 4 whenever the number of grid points is doubled (why?).

- Try now whether you can find the excact numerical finite element solution. Use as an input guess for the wave-function the function $exp(-\frac{1}{2}r^2)$ (do not forget to

normalize it numerically!). Use the steepest descent method with energy feedback. You will find that you need a very small step size α to prevent the energy from going up. Verify that the convergence becomes slower and slower as you add more grid points.

- Add now preconditioning to your steepest descent minimization. Eq. 358 which was derived for an orthogonal basis set becomes

$$\vec{p} = Re\left[(H - (\varepsilon + I\gamma)S)^{-1}\vec{g}\right] \tag{361}$$

in the case where the overlap matrix S is not the unit matrix I. In order to solve the resulting system of equations you need now explicitly the tridiagonal Hamiltonian and overlap matrices. They are calculated by the routines "crthhp" and "crtssp". The complex system of equations is solved by the routine "ctridag". .1 is a good value for γ. With preconditioning, the number of iterations should be nearly independent of the number of grid points and the optimal stepsize t in Eq. 360 should be close to 1. For the solution of the complex system of equations you need to transform a real array into an complex array and the real part of a complex array into a real array. The first operation can be done in Fortran by

```
complex(16), dimension(n) :: ac
real(8), dimension(n) :: ar
...
do i=1,n
ac(i) = cmplx(ar(i),0.d0)
enddo
```

and the second by

```
do i=1,n
ar(i) = real(ac(i))
enddo
```

Part II: The density functional atomic program

- Calculate the electronic density ρ for the LDA hydrogen wavefunction found previously. Check that it has the correct normalization. Numerically, the normalization

integral $\int R(r)^2 r^2 dr$ has to be replaced by a sum. Use the trapezoidal rule which approximates this integral in the following way

$$\sum_{i=0}^{m} R(r_i)^2 r_i^2 w_i \tag{362}$$

where

$$w_i = \begin{cases} \frac{r_1 - r_0}{2} & \text{if } i = 0 \\ \frac{r_m - r_{m-1}}{2} & \text{if } i = m \\ \frac{r_{i+1} - r_{i-1}}{2} & \text{else} \end{cases} \tag{363}$$

r_m is the largest radial grid point and r_0 equals zero in our context.

- Write routines to calculate the Hartree potential V_H using Eq. 213 and the electrostatic energy $E_H = \frac{1}{2}\int V_H(\mathbf{r})\rho(\mathbf{r})d\mathbf{r}$. To do the integrals for V_H and E_H numerically use again the trapezoidal rule with the above defined weights w_i. For the systems considered in this project the charge density is spherically symmetric and we need only the components associated with $Y_{0,0}$. Note that $\rho_{0,0}$ of Eq. 213 is given by $\frac{1}{\sqrt{4\pi}} R(r)^2$. Plot the calculated electrostatic potential. For large distances, it should coincide with a $1/r$ potential.

- Add a routine that calculates the exchange correlation energy. The routine "LSD_PADE" calculates the exchange correlation energy density $\varepsilon_{xc}(\mathbf{r})$ at any point

$\mathbf{r}$ for which the charge density $\rho(\mathbf{r})$ was given as the input. The exchange correlation energy is obtained by integrating over the exchange correlation energy density times the density.

$$E_{xc} = \int \varepsilon_{xc}(\rho(\mathbf{r}))\,\rho(\mathbf{r})d\mathbf{r} \tag{364}$$

In our case $\rho(\mathbf{r}) = \rho(r) = \frac{1}{4\pi}R(r)^2$. Numerically this integration is obviously replaced by a sum. Do the integration again with the trapezoidal rule:

$$E_{xc} = \sum_i \varepsilon_{xc}(i)u(i)^2 r_i^2 w_i \tag{365}$$

- Calculate next the electrostatic and exchange correlation energy for the $1s$ state of hydrogen. If LDA density functional theory was exact, the sum of both would be zero. Unfortunately it is not zero, but it is rather small and the energy according to Eq. 311 is close to the correct value of -.5. Remember that the hydrogen is a spin polarized system and so the input value "ETA" to the subroutine "LSD_PADE" should be 1.d0.

- Because LDA is not exact for the hydrogen atom, the hydrogen $1s$ wave-function is not any more the solution of the LDA Kohn-Sham equations. Obviously this wave-function should not be very different from the true hydrogen wave-function.

We will next find the wave-function which is the solution of the Kohn-Sham equations by a preconditioned steepest descent method. This requires that we add to the gradient $\vec{d}^{hyd} = H^{hyd}\vec{u}$ of the hydrogen atom the contributions from the Hartree and exchange correlation potential. The additional gradient $\vec{d}^{a}$ is the partial derivative of the Hartree and exchange correlation energy with respect to the u_i's. When calculating this gradient one must take into account that both the Hartree and exchange correlation energy depend on the u_i's and one gets the result

$$\vec{d}^{a}(i) = (V_H(i) + v_{xc}^{\uparrow}(i))\, r_i^2\, w_i\, u(i) \tag{366}$$

Write a subroutine that adds this term to the gradient $H^{hyd}\vec{u}$ from the hydrogen hamiltonian H^{hyd}. The ε in the expression for the constrained gradient (Eq. 306) is the Kohn-Sham eigenvalue which is not the Kohn Sham total energy of Eq. 311. It follows from Eq. 318 that the Kohn-Sham eigenvalue ε is related to the hydrogen eigenvalue ε^{hyd} by $\varepsilon = \varepsilon^{hyd} + \sum_i \vec{d}^{a}(i)u(i)$. To summarize, the final constrained gradient is given by

$$g(i) = \sum_j H^{hyd}(i,j)\, u(j) + d^{a}(i) - \varepsilon \sum_j S(i,j)\, u(j) \tag{367}$$

We see that the additional gradient contribution could also be obtained by adding to H^{hyd} the diagonal terms $(V_H(i) + v_{xc}^{\uparrow}(i))r_i^2 w_i$. This has actually to be done for the matrix that is used for preconditioning.

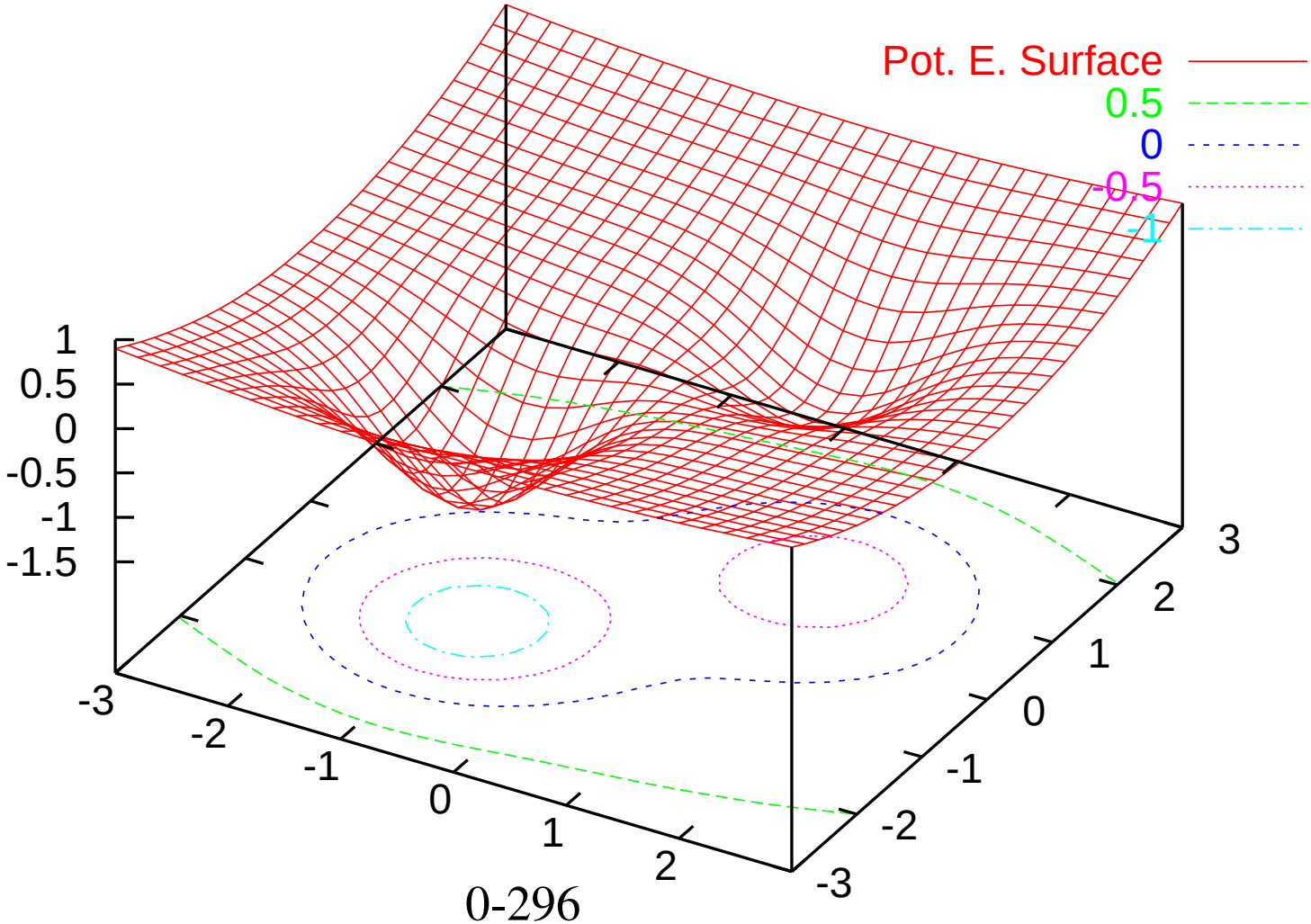
- Calculate next with the same program the total energy of He^+, i.e. of a hydrogenic atom with charge $Z = 2$.

- After having found the LDA energy and wavefunction of the hydrogenic atom we will turn to the helium atom. Not much changes compared to the hydrogenic atom except that we have now 2 electrons in a closed shell system and hence the charge density and kinetic energie have to be calculated according to Eqs. 328 and 329. The spin polarization "ETA" is zero and consequently $\varepsilon_{xc}^{\uparrow}(i) = \varepsilon_{xc}^{\downarrow}(i)$. Calculate the energy of the helium atom by the same preconditioned steepest descent method. The value of ε called "shtr" in the subroutine "ctridag" sould never be much higher in energy than the true eigenvalue. If initial values for ε are very high because the input guess was bad, "shtr" should be constrained to a lower value, that is close to the final eigenvalue.

- Calculate the ionization energy by taking the difference between the LDA total energy of He and He^+.

11 Global geometry optimization

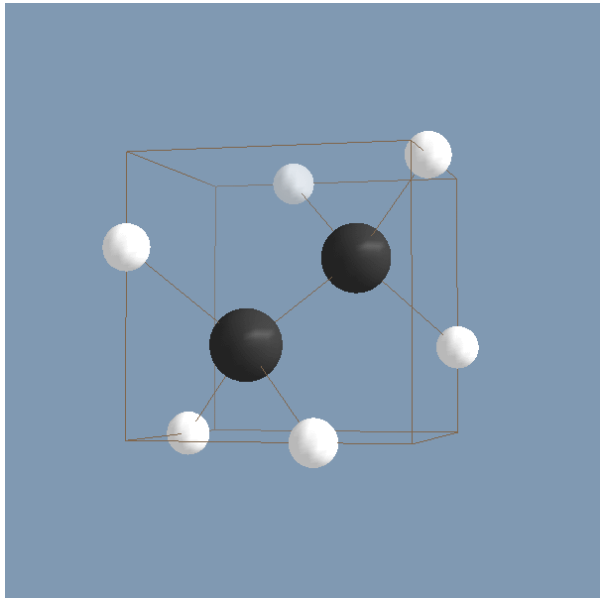
Structure determination is one of the most fundamental endeavors in physics and chemistry. Determining the geometric structure of a solid or molecule requires finding the atomic positions $\mathbf{R}_1, ..., \mathbf{R}_M$ that will give a minimum of the total energy $E(\mathbf{R}_1, ..., \mathbf{R}_M)$. This high dimensional total energy function is also called the Born-Oppenheimer surface. In general this high dimensional function has many minima.

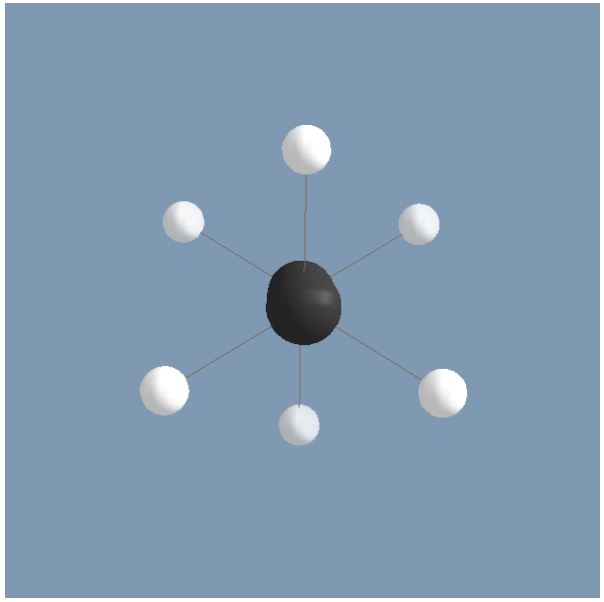


In a local geometry relaxation one finds the local minimum that is closest to a starting point. The minimization methods discussed previously such as the steepest descent or the conjugate gradient method can be used for such a geometry relaxation. Because the condition number of the Hessian is frequently bad, the convergence of these methods can be slow, but nevertheless they will finally find a local minimum. In the case of a molecule there are low frequency torsional and bending modes in addition to high frequency stretching modes. In the case of a bulk-like structure there are long-wavelength elastic modes whose frequencies tends to zero in addition to the high-frequency short-wavelength phonons. As a consequence the condition number is proportional to the largest diameter of the structure.
Exercise [2pt]: *Show that for a linear periodic chain the condition number of the Hessian is proportional to the length of the chain.*

Finding the global minimum of an arbitrary high dimensional function is one of the most difficult mathematical problems. There exists no algorithm that will find such a global minimum with certainty within a computing time that grows less than exponentially with respect to the system size. Systematically exploring the high dimensional space is impossible in practice. Covering it with a grid of m points in each direction would require $m^{3N_{at}}$ grid points because the dimensionality of the Born-Oppenheimer surface of a system of N_{at} atoms is $3N_{at}$. Another complication is that the number of local minima grows exponentially with the number of atoms. This can easily be seen for the alkane family,

C_nH_{2n+2}. As one starts building such a polymer by adding consecutive CH_2 building blocks, one can attach the C atom at any of 3 tetragonal bond directions, while saturating the remaining ones with H. Transforming one configuration into another one requires rotations about $C-C$ bonds, which involves energetic barriers. Hence each configuration is a local minimum and there are of the order of 3^n such local minima. Ethane (C_2H_6) is shown below.





In spite of the mentioned theoretical obstacles there are however algorithms that can find the global minimum for moderately complex systems within acceptable computing time.

11.1 Simulated annealing

Simulated annealing is a classical method to find the global minimum of a high-dimensional function. Even though it is also widely applied outside physics and chemistry we will only consider its application to the structure determination problem.
Simulated annealing is based on thermodynamics. At a sufficiently low temperature the system will be in the ground state, i.e. in the global minimum ε_0 of the potential energy surface since all other minima ε_i have a Boltzmann weight $\exp(-\beta(\varepsilon_i - \varepsilon_0))$ that is vanishingly small. One might hence hope to obtain the ground state by a Markov process that tends towards a low temperature Boltzmann distribution. This will not work in practice. The system will be trapped in some local minimum because at low temperature it can not overcome the barriers that is has to cross to get into other minima. This behavior will be found for more or less any choice of the trial matrix ω. The problem can be alleviated by starting the Markov process at a high temperature and then decreasing the temperature gradually during the simulation until only the ground state remains occupied. In this way the system has hopefully still enough energy to cross barriers before being trapped in the ground state. This gradual decrease of the temperature is the characteristic of simulated annealing and gave rise to its name. There are two essential ingredients of simulated annealing that can be realized in many different ways:

- The type of trial moves that are used which in turn determine the matrix ω.

- The schedule for reducing the temperature

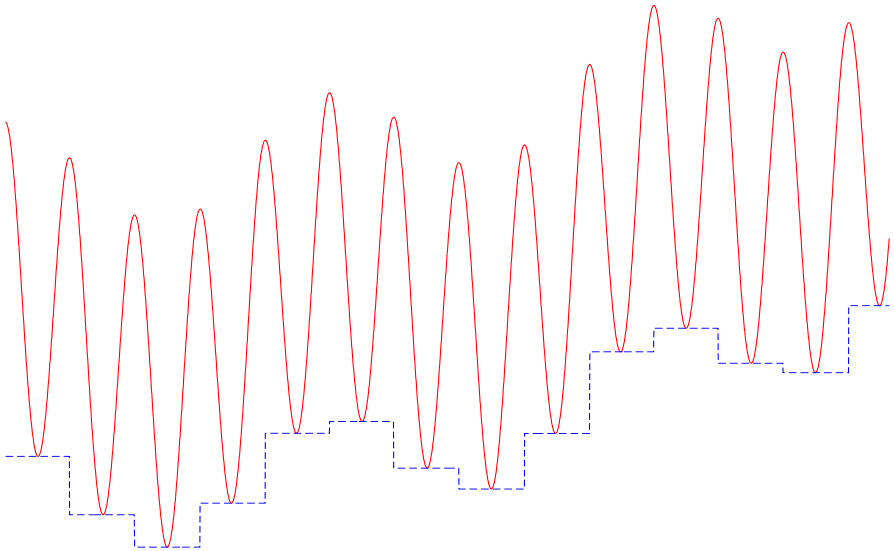
The simplest implementation of simulated annealing is based on molecular dynamics. This has the advantage that one does not have to come up with a prescription for the trial moves. The system is propagated using Newton's equations of motion. Ergodicity ensures that the thermodynamic Boltzmann distribution is finally reached. Molecular dynamics based simulated annealing is thus imitating what is happening in nature during a crystallization process. While the system is slowly cooling down the atoms move according to Newton's law and find finally the global minimum, which is the perfect crystal structure. One is thus only left with setting up a prescription for the cooling rate. The simplest cooling recipe is just to impose an exponential decrease of the temperature. A template program implementing this simplest simulated annealing method is shown below. Some values (4.d0, .9999d0 etc) are just examples and other values may be more appropriate in other contexts. It has to be stressed that there is no guarantee that the global minimum will be obtained at the end of the run. It is always a matter of chance and changing some parameters or the initial atomic positions may well lead to different results.

```
        read initial atomic positions and calculate initial forces

        ref_kin=4.d0
1000    continue
        ref_kin=ref_kin*.99999
        if (ref_kin.le.1.d-3) goto 2000

        DO A VELOCITY VERLET MD STEP AND CALCULATE THE KINETIC ENERGY act_kin

        if (act_kin.gt.ref_kin) then
             reduce velocities by a factor of .99d0
        else
             increase velocities by a factor of 1.01d0
        endif
        goto 1000
2000 continue

        write final atomic positions
```

Exercise [5pt]: *Geometric ground state of sodium clusters*
Use the simulated annealing algorithm of the previous page to find the 3 energetically lowest configurations of a cluster of 13 sodium atoms. Generate by hand some input structures and check whether they will all give the same ground state. Describe the interactions of the sodium atoms with the embedded atom method (Na_v2.eam.fs in MATERIAL folder) used already for the calculation of the auto-correlation functional. A template file is also available in the MATERIAL folder. Use the velocity verlet algorithm (Eq. 98) for the MD part. Check whether energy is conserved (up to some oszillations as shown in the Fig. on page 106). Monitor how many basins of attraction are crossed during the simulated annealing run. Remember that a basin of attraction is associated to a local minimum and that it consists of the ensemble of all points that will lead into this local minimum if they are used as starting points for a steepest descent minimization. Interrupt therefore the MD trajectory every 100 steps and start a steepest descent geometry optimization (Eq. 27) with a small step size from the current configuration along the MD trajectory. Use an energy feedback to adjust the step size. While executing the program write into a file the energy values of the minima of the current basins of attraction found by the geometry optimization. Plot this file with gnuplot or some similar software during program execution to monitor the progress in the search for the global minimum. Save the configurations of the 3 lowest local minima configurations into files.

11.2 Basin hopping

A basin is the 'region of attraction' around a local minimum. All small step size steepest descent minimizations that are started within the basin associated to one local minimum will end up in this local minimum. Basin hopping is a Monte Carlo method on a modified potential energy surface. In an ordinary Monte Carlo simulation the Boltzmann factor $\exp(-\beta(E_{new} - E_{curr}))$ of the Metropolis step contains the energies E_{new} and E_{curr} of the new configuration X_{new} and of the current configuration X_{curr}. A configuration X in our context is determined by all the atomic positions $\mathbf{R}_1, \mathbf{R}_2, \ldots$. In the basin hopping method E_{new} and E_{curr} have different meanings. They are the energies of the local minimum of the basins in which X_{new} and X_{curr} are located. This gives rise to a modified potential that is constant within one basin.



1-dim potential energy toy surface. High barriers separate the basins around each local minimum. In the basin hopping method all the barriers disappear resulting in a piecewise constant potential surface.

The basin hopping has the advantage that barriers separating different minima can be overcome much more easily during the simulation. In an ordinary Monte Carlo simulation the Boltzmann factor $\exp(-\beta(E_{new} - E_{curr}))$ can become very small when one tries to cross into another basin since the energy differences can be much larger than on the transformed potential energy surface of the basin hopping method. In the worst case E_{curr} is the energy of a local minimum and E_{new} is the energy at the top of a barrier between two minima in an ordinary Monte Carlo method. As a consequence crossing from one basin into another is a rather rare event in ordinary Monte Carlo simulations. Hence it can take a very long time until one finally falls into the global minima.
Exactly the same problem is encountered in simulated annealing using MD. At low temperature, the MD trajectory will oscillate back and forth most of the time in the basin around one local minimum and it will only rarely jump into another basin.
In the basin hopping method the calculation of the transformed potential is performed on the fly. For each configuration X one performs a local geometry optimization using a method such as steepest descent. The energy of the local minimum found in this way is then the energy of the configuration X. The trial steps that bring us from one configuration to the next are in the simplest case just random displacements of the atoms. For small random displacements one will remain for a long time in the same basin. Since the energies remain constant, all these moves are accepted in the Metropolis step. On the other hand, if one chooses very large random displacements the algorithm is similar to a random search. Such a random search is generally less efficient because it ignores relationships

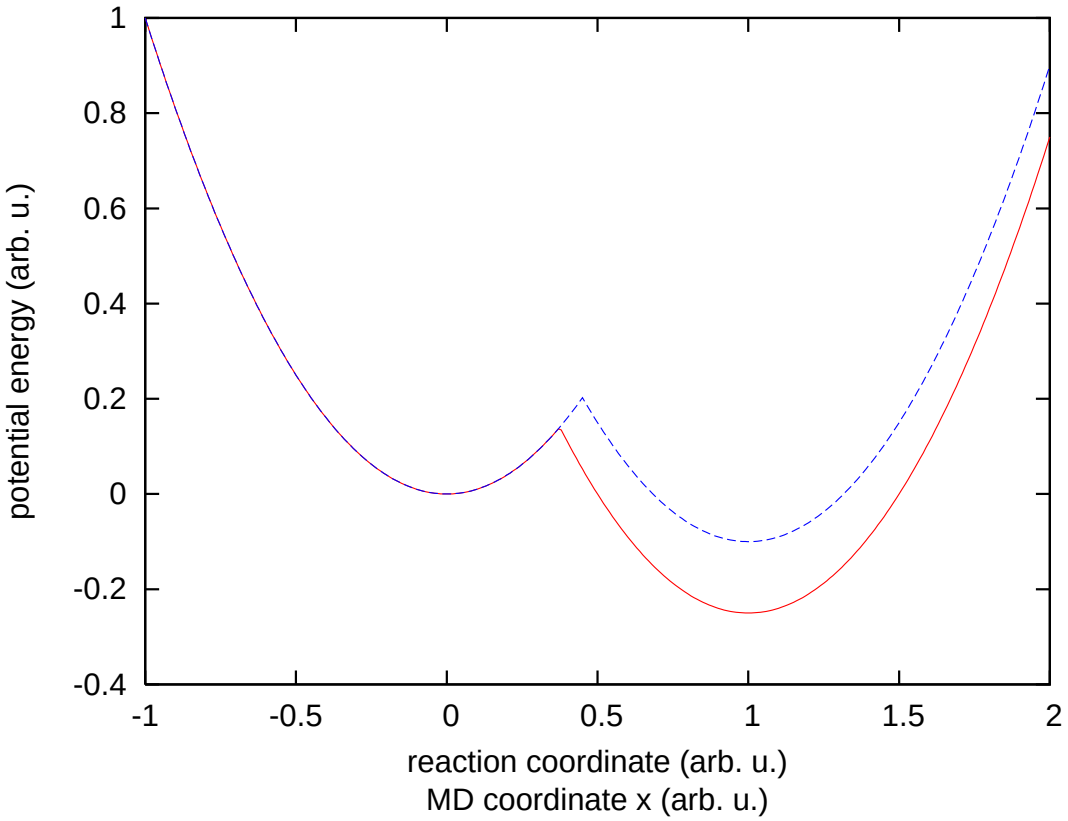
between neighboring local minima. If one has already found a good local minimum, it is likely, that other even better ones are close by. A too large step size gives therefore a low acceptance probability in the Metropolis step. The step size of the random displacement is therefore usually adjusted such that half of all new configurations are accepted. The resulting algorithm is sketched below:

```
    initialize configuration X with energy E ;  initialize stepsize

      do 1000, istep=1,nstep

    generate a new trial configuration:
      Xtrial = X + stepsize*random_vector

      starting from Xtrial do a local minimzation
      to get energy Etrial of the local minimum

    calculate Boltzman factor exp(-beta*(Etrial-E)) for Metropolis step
      If Xtrial is accepted in the Metropolis step then
      X=Xtrial ; E=Etrial
      stepsize=stepsize*1.05
      else
      stepsize=stepsize*.95
      endif

1000  continue
```

There is one free parameter in the basin hopping method, namely the temperature that is hidden in the parameter β. This temperature can be lowered successively during a simulation. Basin hopping can thus be used within a simulated annealing scheme, replacing MD. Thermodynamics guarantees that at sufficiently low temperature only the basin of the global minimum will be populated. But again thermodynamics can not tell us how long it will take until the thermodynamic equilibrium distribution is reached and hence how long it will take to find the global minimum.

11.3 Minima hopping

The next question that arises is how one should search for lower local minima that are close to the current minimum. The answer is the following: One should try to go over low barriers, because neighboring minima that can be reached by crossing low barriers are more likely to be low in energy themselves. This relation between the barrier height and the energy of the local minimum 'behind' the barrier is explained by the Bell-Evans-Polanyi (BEP) principle. It assumes that the entire function is made out of quadratic pieces as shown below. Shifting down the parabola to the right will lower the barrier.



$$E(x) = min\left(x^2, (x-1)^2 - .1\right)$$

$$E(x) = min\left(x^2, (x-1)^2 - .25\right)$$

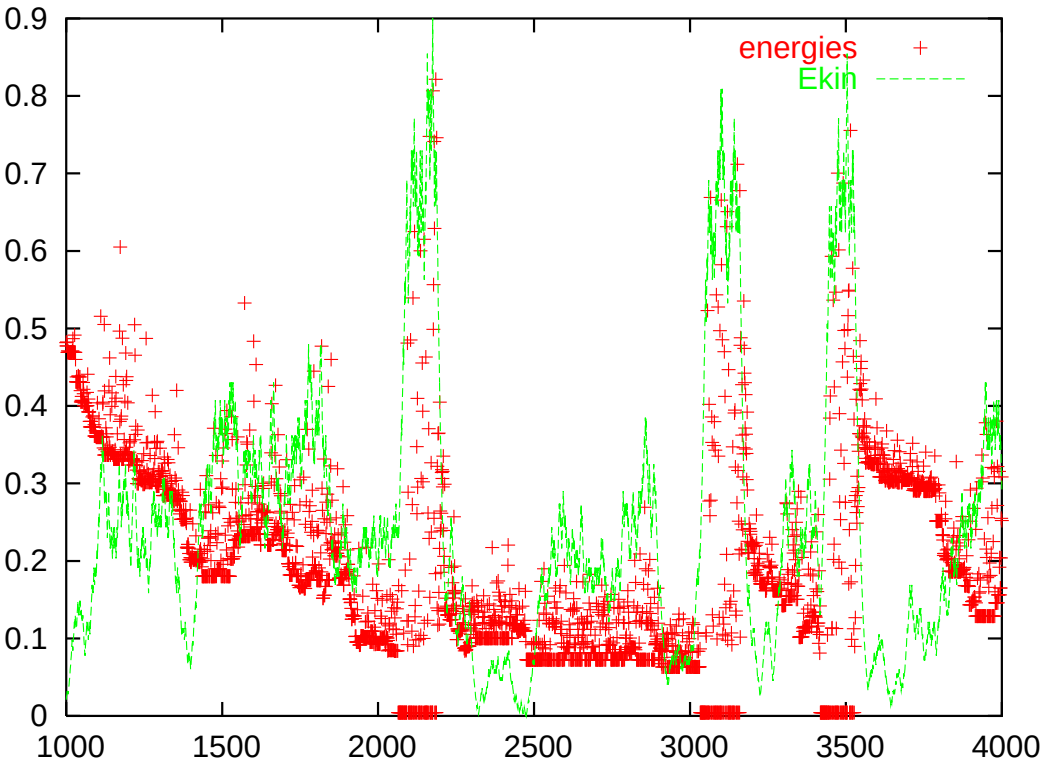
A simple way to find relatively low barriers is to do molecular dynamics, using as the potential the function $f(x_1,...,x_d)$ to be minimized. With molecular dynamics one solves numerically Newtons equation of motion. The forces are given by the negative gradient of $f(x_1,...,x_d)$. Hence the sum of the kinetic and potential energy has to remain constant during such a simulation. If the system has a certain kinetic energy $E_{kinetic}$ then it simply can not cross barriers that are higher than $E_{kinetic}$. The molecular dynamics simulation is started in the current local minimum. Initially the kinetic energy will decrease since the system moves uphill. When it increases again the system has either crossed a barrier or it is oscillating back towards the initial local minimum. At this point the molecular dynamics simulation is stopped and the closet local minimum is found by the standard local minimization techniques such as conjugate gradient. If one is lucky one ends up in a minimum that is different from the current local minima. By repeating this process one can explore many low energy local minima.

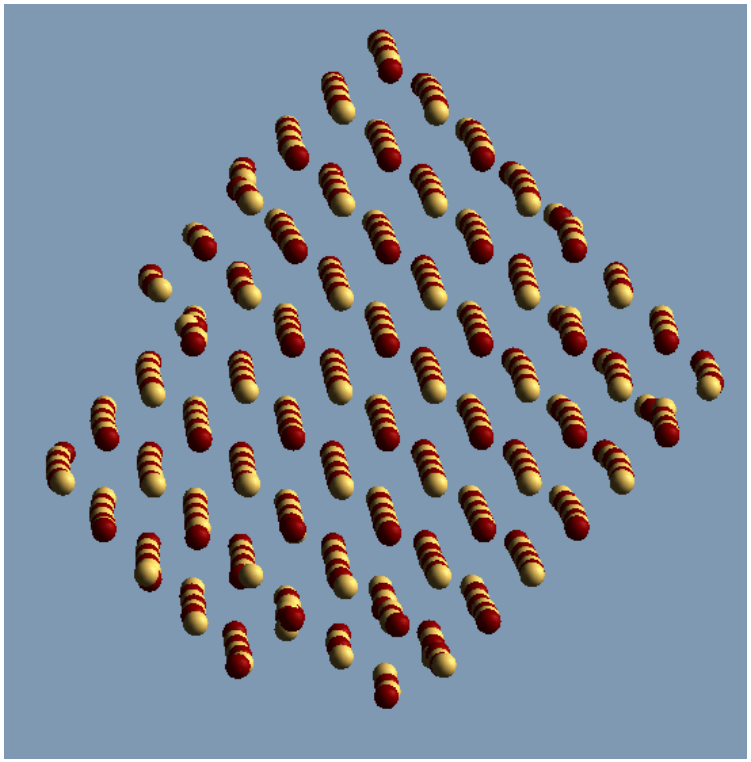
Minima hopping flow chart

```
        initialize a current minimum 'Mcurrent'
MDstart
     ESCAPE  TRIAL PART
       start a MD trajectory with kinetic energy Ekin from the current minimum 'Mcurrent'
       in a soft direction. Once potential energy reaches the mdmin-th minimum along the
       trajectory stop  MD and optimize geometry to find the closest local minimum 'M'
       if ('M' equals 'Mcurrent') then
          Ekin = Ekin*beta_same  (beta_same > 1)
          goto MDstart
       else if ('M' equals a minimum visited previously) then
          Ekin = Ekin*beta_old  (beta_old > 1)
       else if ('M' equals  a new minimum ) then
          Ekin = Ekin*beta_new  (beta_new < 1)
       endif
    DOWNWARD PREFERENCE PART
       if ( energy('M') - energy('Mcurrent') < Ediff ) then
          accept new minimum: 'Mcurrent' = 'M'
          add 'Mcurrent' to history list
          Ediff = Ediff*alpha_acc (alpha_acc < 1)
       else if rejected
          Ediff = Ediff*alpha_rej (alpha_rej > 1)
       endif

       goto MDstart
```

Example: minima hopping for a 512 atom NaCl cluster

Global Minimum: 8 by 8 by 8 cube
wrong funnnel: 7 by 8 by 9 structure





11.4 Genetic algorithms

In contrast to the previous methods for finding the global minimum, genetic algorithms do not have their foundation in thermodynamics. Instead they try to mimic the Darwinistic evolution. The principle is the survival of the 'fittest' and genetic algorithms are for instance using steps that are called mutations and crossovers. The basic quantity is a population of individuals that are represented by their genes. Numerically these genes are binary strings. A mutation consists of a random change of a gene, i.e. of a flip of one or several of the bits in a gene. It is thus similar to a trial step move in a Monte Carlo method. What is really new compared to Monte Carlo methods is the concept of gene crossovers. Given two genes of two individuals, a crossover point is first determined at random and then the genes are combined as shown below to obtain a child.

1 0 0 1 1 1 1 0 0 1 'mother gene'
1 0 1 1 0 0 0 1 1 1 'father gene'
——— ——————-
1 0 0 1 0 0 0 1 1 1 'child gene'

Gene crossing makes only sense if neighboring genes determine common functionalities. This can be easily seen by going back to biology. If for instance in the example above, the first 4 genes encode the functionality of ear and the last 6 the functionality of the eye, then the child has a certain chance having both good ears and a good eyes assuming that the mother had good ears and the father good eyes. If however the first 5 genes determine the

ear and the last five the eye then the above crossover after the fourth bit will very likely result in both ears and eyes that do not work very well.
After performing the operations of mutation and crossovers on a population comes the final survival step. The fitness of each individual i in a population that may consist of parents and children generated by both mutations and crossovers is measured by its fitness f_i which would be in a physical problem for instance the negative of the energy of a configuration. The average fitness of our population $< f >$ of N individuals is given by

$$< f >= \frac{1}{N} \sum_{i=1}^{N} f_i \tag{368}$$

The survival rate of an individual is then proportional to $f_i / < f >$.
Repeating the processes of mutation/crossover and survival gives fitter and fitter populations and the hope is that finally a population might contain a 'perfect' individual, which in the mathematical language would be the global maximum/minimum.
Applying genetic algorithms to structural optimization is problematic for several reasons. First, it is unnatural to represent atomic positions by short binary strings. A continuous problem is in this way mapped onto a discrete problem. Second it is not quite clear how to do the crossover in an efficient way. In order to optimize the structure of clusters, people devised a crossover process that simply combines the parts of two clusters as shown below. It is however questionable whether half of a cluster represents a meaningful functional

unit.