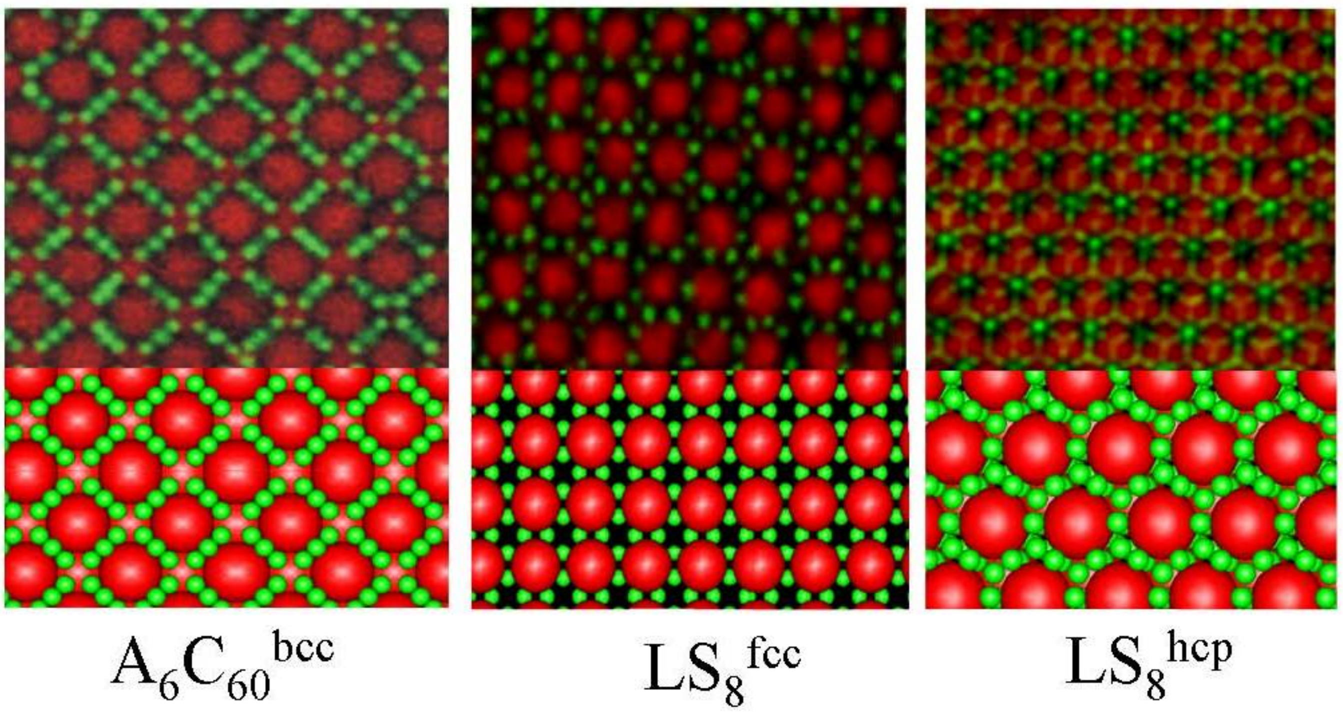
Modelling and Simulation

2026



Marjolein Dijkstra
Soft Condensed Matter & Biophysics
Debye Institute for Nanomaterials Science
Utrecht University, the Netherlands
m.dijkstra@uu.nl
https://colloid.nl/people/marjolein-dijkstra/

Universiteit Utrecht

This is the twelfth edition of the lecture notes of this Master course at Utrecht University written by Marjolein Dijkstra. Parts of the text have been taken from earlier courses which were given together with Thijs Vlugt. The C manual and the exercises were written by Laura Filion, Frank Smallenburg, and Michiel Hermes. We thank all authors for giving permission for using their parts of their texts.

Please report errors, corrections, and other suggestions to M.Dijkstra@uu.nl

Contents

Chapter 1

Introduction to Monte Carlo Methods

1.1 Introduction

In this chapter, we present an introduction to Monte Carlo methods. We describe the basic principles of Monte Carlo simulations and start with a simple Monte Carlo algorithm in the canonical ensemble, i.e. we fix the number of particles N, the volume of the system V and the temperature T. We consider a system that is described by the Hamiltonian:

$$H(\mathbf{r}^N, \mathbf{p}^N) = \sum_{i=1}^{N} \frac{\mathbf{p}_i^2}{2m} + U(\mathbf{r}^N). \tag{1.1}$$

The canonical partition function $Z(N, V, T)$ of such a system is given by

$$Z(N, V, T) = \frac{1}{\Lambda^{3N} N!} \int d\mathbf{r}^N \exp\left[-\beta U(\mathbf{r}^N)\right]. \tag{1.2}$$

The thermal wavelength (also called the De Broglie wavelength) is defined by

$$\Lambda = \frac{h}{\sqrt{2\pi m k_B T}} \tag{1.3}$$

and results directly from the integration over the momenta (see Eq. **??**). The free energy of the system equals

$$F(N, V, T) = -k_B T \ln Z(N, V, T) \tag{1.4}$$

and quantities like the pressure and energy of the system can be computed by employing partial derivatives of $F(N, V, T)$. The canonical ensemble averages of momentum-independent observables, i.e., quantities that only depend on $\mathbf{r}^N$, can be expressed as

$$\langle A \rangle = \frac{\int d\mathbf{r}^N A(\mathbf{r}^N) \exp\left[-\beta U(\mathbf{r}^N)\right]}{\int d\mathbf{r}^N \exp\left[-\beta U(\mathbf{r}^N)\right]}. \tag{1.5}$$

In this chapter, we will show that the partition function $Z(N, V, T)$ cannot be measured in simulations. However, we are able to compute canonical ensemble averages as defined in Eq. 1.5. To illustrate this, we will consider the example of a liquor store (see Fig. 1.1). It is impossible to sample the whole phase space of the liquor store ($\mathbf{r}^N$). After a few samples, you will be too drunk to stand on your feet and you can not sample properly the whole phase space of the liquor store anymore. However, one can measure ensemble averages. For instance, one can take a selection of the beverages and measure the ratio of red wines out of all bottles of wines. In the next section we describe a smart method to calculate ensemble averages in simulations.

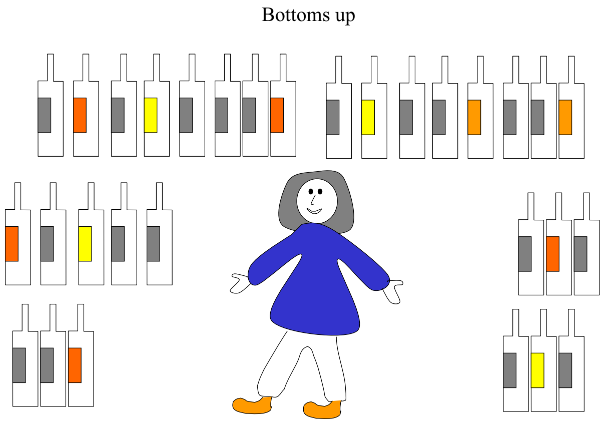


Figure 1.1: One cannot sample the whole phase space of the liquor store, but one can sample ensemble averages, like the ratio of red wine out of all the bottles of wines.

1.2 The Metropolis Method

Naively, one might think that $\langle \mathcal{A} \rangle$ can be evaluated by conventional numerical integration techniques such as numerical quadrature. However, evaluating the integrand on a grid in the high-dimensional phase space is impossible as the number of gridpoints becomes more than astronomically large. For instance, $N = 100$ particles in $D = 3$ dimensions using a very rough grid of only $m = 5$ gridpoints already leads to $m^{DN} = 5^{300}$ gridpoints at which this integrand has to be evaluated. This is impossible within the lifetime of our universe. In addition, suppose that we would somehow be able to perform the integration of Eq. 1.2. Our claim is that the statistical error will then be so large that the result is not meaningful anyway. The reason is that when two particles overlap the potential energy is extremely large and therefore the Boltzmann factor equals zero. In fact, it turns out that for a typical liquid this is the case for almost all configurations $\mathbf{r}^N$ and only an extremely small part of the phase space will have a nonzero contribution to $Z(N, V, T)$.

We therefore have to resort to better numerical techniques. One such a technique is the Monte Carlo method. The basic idea in the Metropolis Monte Carlo method is that phase points are generated in phase space according to the desired probability. In this way, one avoids the numerical evaluation of the integrand on a high-dimensional grid where most of the gridpoints results in configurations with an extremely low Boltzmann weight. Consider for example a sequence of configurations $\mathbf{r}_1^N, \mathbf{r}_2^N, \mathbf{r}_3^N, \cdots, \mathbf{r}_n^N$. If this sequence consists of random configurations, the ensemble average $\langle A \rangle$ is simply

$$\langle A \rangle = \lim_{n\to\infty} \frac{\sum_{i=1}^{n} A(\mathbf{r}_i^N) \exp\left[-\beta U(\mathbf{r}_i^N)\right]}{\sum_{i=1}^{n} \exp\left[-\beta U(\mathbf{r}_i^N)\right]} \tag{1.6}$$

Again, computing $\langle A \rangle$ in this way is often not meaningful as $\exp\left[-\beta U(\mathbf{r}_i^N)\right]$ is nearly always zero. However, suppose that we are able to generate $\mathbf{r}_i^N$ in such a way that the probability of generating $\mathbf{r}_i^N$ is proportional to $\exp\left[-\beta U(\mathbf{r}_i^N)\right]$, then the ensemble average is simply

$$\langle A \rangle = \lim_{n\to\infty} \frac{\sum_{i=1}^{n} A(\mathbf{r}_i^N)}{n} \tag{1.7}$$

This sampling is called Metropolis sampling [1]. As we cannot generate an infinitely long sequence of configurations on the computer, we estimate the ensemble average by taking a large value of n.

To illustrate the difference, we compare two ways of measuring the ratio of red wines out of the bottles of wines of all liquor stores in the Netherlands, by random sampling (Eq. 1.6) and by Metropolis sampling (Eq. 1.7). In the conventional random sampling scheme, the value of the integrand (i.e. the ratio of red wines over all wines) is measured at each gridpoint and many of the gridpoints are located in regions where the Boltzmann factor is vanishingly small (i.e. outside the liquor stores, see Fig. 1.2). In contrast, in the Metropolis sampling, a random walk is constructed through the region of phase space where the integrand is non-negligible, i.e. in all the liquor stores themselves. In the Metropolis sampling trial moves are generated, which are accepted according to its Boltzmann weight. In our example, it means that trial moves that takes you out of one of the liquor stores are rejected, and accepted otherwise, see Fig. 2.3. After each trial move, which can be accepted or rejected, the ratio of red wines is measured.

In principle, the random sampling scheme will give us also a result for the total phase space area (the total number of bottles of red and white wine in the Netherlands), a quantity similar to the total partition function, while this information is not contained in the Metropolis sampling method.

Question 1 (Total phase space area)
By using a nasty trick (importing a bottle of red or white wine from another country and measuring the difference in the red wine/white wine ratio), we are still able to compute the total phase space area using Metropolis sampling. Explain why. Can you think of another nasty trick to do this ?

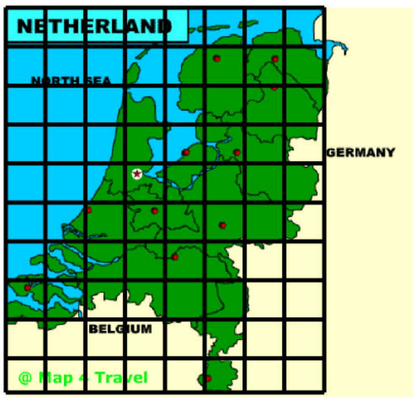


Figure 1.2: Conventional quadrature scheme for measuring the ratio of red wines out of the bottles of wines in all liquor stores in the Netherlands.

We will now present a method to generate $\mathbf{r}_i^N$ with a probability proportional to $\exp\left[-\beta U(\mathbf{r}_i^N)\right]$ (Metropolis sampling). For a system of N particles in volume V, this works as follows: We first generate a configuration of N particles at positions $o = \{\mathbf{r}_o^N\}$ with a non-vanishing Boltzmann weight $\exp[-\beta U(o)]$. Next we generate a random new trial configuration $n = \{\mathbf{r}_n^N\}$, e.g. by picking randomly a particle and by displacing it randomly. The Boltzmann weight of this new trial configuration is $\exp[-\beta U(n)]$. We must now decide whether we accept or reject this trial configuration satisfying the constraint that on average the probability of finding the system in a configuration $\{\mathbf{r}^N\}$ is proportional to the probability distribution $f_c(\{\mathbf{r}^N\}) = \exp[-\beta U(\mathbf{r}^N)]$. If we reject this trial move, the next element in our sequence of configurations (Eq. 1.7) will be $\{\mathbf{r}_o^N\}$, otherwise it will be $\{\mathbf{r}_n^N\}$. The rule to accept or reject $\{\mathbf{r}_n^N\}$ must satisfy three conditions:

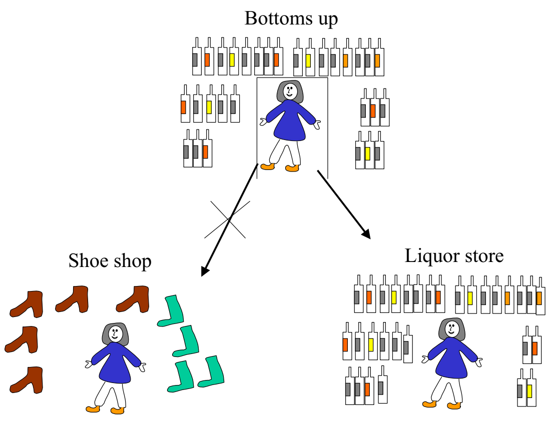


Figure 1.3: Metropolis sampling method: A random walk is made through phase space in the regions where the integrand is non-negligible, i.e. in all liquor stores of the Netherlands.

(1) the number of points in any configuration o should be proportional to $f_c(o)$, *(2)* all configurations can, in principle, be visited (this condition is called ergodicity), *(3)* in equilibrium, the average number of accepted trial moves leaving state o should be equal to the average number of accepted trial moves from all other states to state o. This is called the balance condition [2]. It is however, convenient to impose a much stronger condition (detailed-balance condition); namely that in equilibrium the average number of accepted trial moves leaving state o to state n is equal to the average number of accepted trial moves leaving state n to state o, which essentially means that all "fluxes of configurations" are equal:

$$f_c(o)\pi(o \to n) = f_c(n) \times \pi(n \to o) \tag{1.8}$$

where $\pi(o \to n)$ denotes the transition matrix which can be split in two matrices: *(1)* the probability $\alpha(o \to n)$ to perform a trial move from $o \to n$, and *(2)* the probability $\text{acc}(o \to n)$ of accepting a trial move from $o \to n$:

$$\pi(o \to n) = \alpha(o \to n) \times \text{acc}(o \to n) \tag{1.9}$$

In the original Metropolis scheme [1], $\alpha(o \to n)$ is chosen to be symmetric [*], i.e. $\alpha(o \to n) = \alpha(n \to o)$ and we arrive at

$$\frac{\text{acc}(o \to n)}{\text{acc}(n \to o)} = \frac{f_c(n)}{f_c(o)} = \exp[-\beta(U(n) - U(o))] \tag{1.10}$$

An example of such a symmetric transition is the random displacement of a randomly selected particle. Many choices for $\text{acc}(o \to n)$ satisfy this condition, but the commonly used choice is of Metropolis:

$$\text{acc}(o \to n) = \begin{cases} f_c(n)/f_c(o) & \text{if} \quad f_c(n) < f_c(o) \\ 1 & \text{if} \quad f_c(n) \geq f_c(o) \end{cases} \tag{1.11}$$

More explicitly, to decide whether a trial move will be accepted or rejected we generate a random number, denoted by `ranf()`, from a uniform distribution in the interval [0,1]. If

[*]Note that many Monte Carlo schemes are not symmetric, see for example Ref. [3].

`ranf()`$<$ acc$(o \rightarrow n)$ we accept the trial move and we reject it otherwise. This rule satisfies the Metropolis condition and can be written as

$$\text{acc}(o \rightarrow n) = min(1, \exp[-\beta \times (U(n) - U(o))]) = min(1, \exp[-\beta \Delta U]) \tag{1.12}$$

where $min(a, b) = a$ when $a < b$, and b otherwise. This means that new configurations that lower the energy are always accepted, and new configurations that increase the total energy are accepted with a certain probability that depends on their energy difference and the temperature. It is important to note that it is not trivial to generate a random number on a computer.

Question 2 (Symmetric condition)
Show that the following acceptance rule

$$acc(o \rightarrow n) = \frac{f_c(n)}{f_c(n) + f_c(o)} \tag{1.13}$$

also satisfies detailed balance (Eq. 1.8). Compare this acceptance rule with the conventional Metropolis acceptance rule (Eq. 1.11) for very small random particle displacements.

1.3 Periodic Boundary Conditions

Periodic boundary conditions are often applied in computer simulations of bulk phases to avoid surface effects. Simulations are used to gain information of a macroscopic bulk system. However, the number of particles that can be handled using present-day simulations ranges from a few hundred to a few million. This number is still far from the macroscopic limit (1 mol $\approx 10^{23}$ particles). In fact, if one considers a three-dimensional system then the fraction of the molecules that is at the surface is about $N^{-1/3}$. For instance a crystal consisting of $10 \times 10 \times 10 = 1000$ particles, has about 600 particles at the surface, while a system of a million particles possess still 6% surface particles. To circumvent any surface effects, periodic boundary conditions are employed, see Fig. 1.4. The simulation box is replicated in all directions to form an infinite lattice. During the simulation, all particles in the central box and their periodic images move in the same way. If a particle leaves the central box, one of its images will appear at the opposite face. The number of particles in the central box is fixed. It is not necessary to store the coordinates of all the images in a simulation and we only store the particles in the central box. There are no walls at the boundaries of the central box and, hence, there are no surface effects. In principle, a particle i may interact with all periodic images of particle j. However, we will always apply the nearest image convention which means that a particle i is only allowed to interact with the *nearest image* of particle j.

Question 3 (Nearest image convention)
Consider a system of particles in a rectangular box of dimensions $L \times L \times L$. Show that the nearest image convention implies that distances between particles in any of the three directions are always larger than $-L/2$ and always smaller than $L/2$.

Question 4 (Fraction of surface particles)
Consider a cubic lattice consisting of $n \times n \times n$ particles. Derive that for large n, the fraction of particles at the surface is approximately equal to $6/n$.

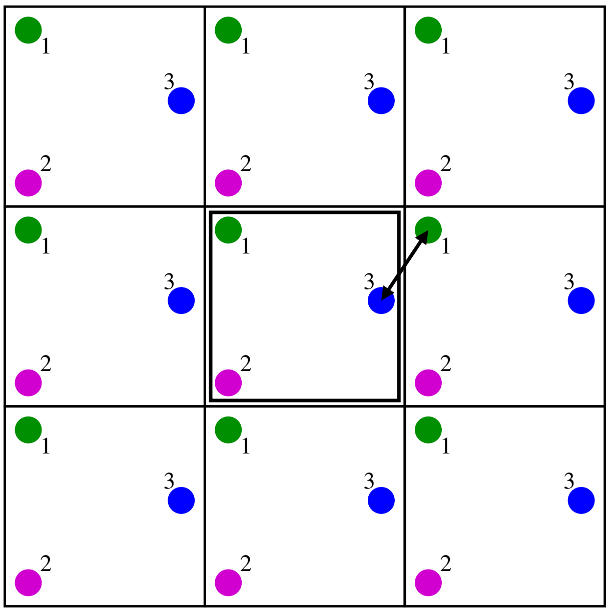


Figure 1.4: Periodic boundary conditions. A central box is surrounded by copies of itself. The arrow shows the shortest distance between particles 1 and 3.

1.4 Lennard-Jones Potential

The Lennard-Jones potential is a simple, idealized pair potential to describe the interactions between atoms. The pair potential, which is commonly used in computer simulations is given by

$$u_{LJ}(r_{ij}) = 4\epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right] \tag{1.14}$$

where σ and ϵ are the so-called Lennard-Jones parameters, see Fig. 1.5, and $r_{ij} = |\vec{r}_i - \vec{r}_j|$ is the distance between two atoms. In this model, two atoms attract each other at large distances as a result of attractive van der Waals forces, but at short distances two atoms repel each other.

Question 5 (Lennard-Jones potential)
Answer the following questions:

- *Two atoms repel each-other at very small distances. Explain why.*
- *Make a plot of the Lennard-Jones potential.*
- *Show that this potential has a minimum for $r_{\text{min}} = 2^{1/6}\sigma$ and that $u(r_{\text{min}}) = -\epsilon$.*

As the Lennard-Jones pair potential describes the interaction energy between two particles, the total energy of a Lennard-Jones fluid should be computed by summing over all possible atom pairs, i.e.

$$U(\mathbf{r}^N) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} u(r_{ij}) = \sum_{i<j} u(r_{ij}) \tag{1.15}$$

For a system consisting of N atoms, there are $N(N-1)/2$ pairs which means that the computational cost of computing the energy of a system is of order N^2. The number of interactions that

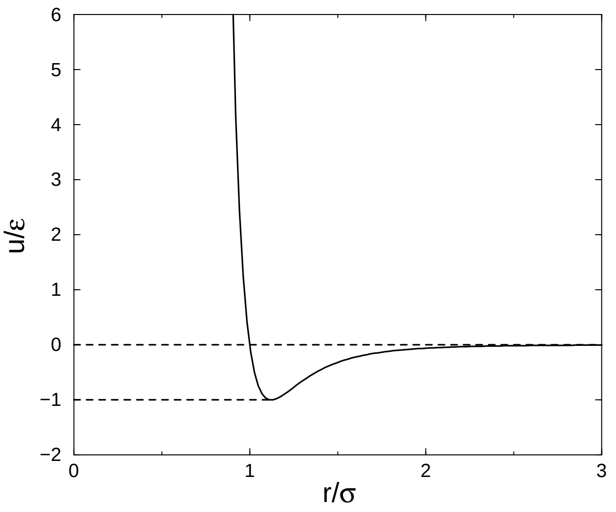


Figure 1.5: The Lennard-Jones potential. u/ϵ is minimal (-1) for $r = 2^{1/6}\sigma$.

need to be computed can be reduced by neglecting all interactions beyond a certain radius r_{cut}, which should not be too small (in practice, $r_{\text{cut}} = 2.5\sigma$ is often used). In any case, r_{cut} should always be smaller than half the boxsize L. Truncating the Lennard-Jones potential at r_{cut} results in the following potential

$$u(r) = \begin{cases} 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] & r \leq r_{\text{cut}} \\ 0 & r > r_{\text{cut}} \end{cases} \tag{1.16}$$

which is a discontinuous function for $r_{ij} = r_{\text{cut}}$. However, in Molecular Dynamics simulations, it is often convenient to remove this discontinuity. This results in the so-called truncated and shifted Lennard-Jones potential:

$$u(r) = \begin{cases} 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] - e_{\text{cut}} & r \leq r_{\text{cut}} \\ 0 & r > r_{\text{cut}} \end{cases} \tag{1.17}$$

in which

$$e_{\text{cut}} = 4\epsilon\left[\left(\frac{\sigma}{r_{\text{cut}}}\right)^{12} - \left(\frac{\sigma}{r_{\text{cut}}}\right)^{6}\right] \tag{1.18}$$

It is important to note that the result of a computer simulation may depend on r_{cut} and whether a truncated or truncated and shifted potential is used.

Question 6 (Tail corrections)
Suppose that we have a system of N Lennard-Jones particles in a volume V using periodic boundary conditions and that the Lennard-Jones potential is truncated at r_{cut} (Eq. 1.16). A simple way to estimate the error of this truncation is the use of a so-called tail correction. Consider a single particle in the system. When we assume that the local number density of particles located at distances beyond r_{cut} equals $\rho = N/V$, the error that we make by truncating the potential for a single central particle equals

$$u_{tail} = \frac{4\pi N}{V}\int_{r_{\text{cut}}}^{\infty} dr r^2 u_{LJ}(r) \tag{1.19}$$

- *Derive this equation.*

- *Show that the total tail correction energy of the whole system equals*

$$U = 8\pi\rho N \left[\frac{1}{9}\left(\frac{\sigma}{r_{\text{cut}}}\right)^9 - \frac{1}{3}\left(\frac{\sigma}{r_{\text{cut}}}\right)^3\right] \tag{1.20}$$

- *Is it possible to use tail corrections for the interaction potential $u(r) \propto r^{-3}$? Explain why.*

- *Discuss the possible difficulties of the use of tail corrections for a system which contains a vapor/liquid interface.*

- *Discus the possible difficulties of the use of tail corrections in a perfect crystal.*

1.5 Reduced Units

In physics and chemistry, it is common practice to used standard SI units (kilogram, meter, second). However, on the scale of a single molecule, this results in very small numbers. For example, the mass of a typical atom is of the order of 10^{-25}kg. Therefore, for molecular systems, it is more natural to use a single molecule as a unit. For example, consider a system consisting of argon atoms that interact via a Lennard-Jones potential. A convenient set of units would be

- unit of energy: ϵ
- unit of length: σ
- unit of mass: m (the mass of a single argon atom)

All other units follow directly from these basic units.

Question 7 (Reduced units)
Show that the resulting units of time, temperature, pressure, and number density are respectively $\sigma\sqrt{m/\epsilon}$, ϵ/k_B (in which k_B is the Boltzmann factor), $\epsilon\sigma^{-3}$, and σ^{-3}

We are now able to express quantities in terms of these reduced units. For example, a reduced distance l^* equals l/σ, in which l is the real distance and σ the basic unit of length. As we will only used reduced units from now on, we will drop the superscript * for reduced units. Using reduced units has several important advantages:

- Very small or very large numbers are avoided. If the outcome of a computer simulation is an extremely large number, it is quite likely that this is because of a programming error. Also, we avoid an underflow and overflow on the computer.

- For different systems characterized by the same functional form of the interaction potential, only a single calculation is required. For example, from the equation of state of a Lennard-Jones potential in reduced units, we are able to calculate the equation of state for argon and neon, provided that their Lennard-Jones parameters are available.

Question 8 (Conversion between simulations)
Suppose that we simulate argon at a temperature of 500 K and a pressure of 1 bar using a Lennard-Jones potential. For which conditions of neon can we use the same simulation data ? The Lennard-Jones parameters are: $\sigma_{Ar} = 3.41\text{Å}$, $\sigma_{Ne} = 2.75\text{Å}$, $\epsilon_{Ar}/k_B = 120K$, $\epsilon_{Ne}/k_B = 36K$.

1.6 Computing the Pressure

It is straightforward to compute the ensemble average of the pressure in a Monte Carlo simulation. Using Eq. 1.4 and the standard equation

$$P = -\left(\frac{\partial F}{\partial V}\right)_{N,T} \tag{1.21}$$

we can write

$$P = \frac{k_B T}{Z}\left(\frac{\partial Z}{\partial V}\right)_{T,N} \tag{1.22}$$

Using scaled coordinates, one finds the virial equation for the pressure

$$\langle P\rangle = \rho k_B T - \frac{1}{3V}\left\langle \sum_{i<j}^{N} r_{ij} \times \left(\frac{du(r_{ij})}{dr_{ij}}\right)\right\rangle \tag{1.23}$$

The structure of a homogeneous fluid can be characterized by the pair correlation function $g(r)$, which describes the probability of finding a pair of particles at a distance r relative to the probability expected for an ideal gas of particles at the same density. The pair correlation function $g(r)$ is defined as

$$g(\mathbf{r}_1, \mathbf{r}_2) = \frac{N(N-1)}{\rho^2 Z(N,V,T)} \int d\mathbf{r}_3 d\mathbf{r}_4 \cdots d\mathbf{r}_N \exp[-\beta U(\mathbf{r}^N)] \tag{1.24}$$

and as the labelling of the particles 1 and 2 is arbitrary, one can rewrite this expression as

$$g(r) = \frac{2V}{N}\left\langle \sum_{i<j} \delta(r - r_{ij})\right\rangle \tag{1.25}$$

in which $\delta(x)$ is the usual δ-function. By definition, $g(r) = 1$ for an ideal gas.

Question 9 (Computing the pressure)
Derive Eq. 1.23 by switching the differentiation of Z and the integration over $\mathbf{r}^N$ and using scaled coordinates ($\mathbf{r}^N = L \times \mathbf{s}^N$ where we assume a cubic box of volume $V = L^3$).

1.7 A Basic Monte Carlo Algorithm

In practice, our Monte Carlo approach to compute ensemble averages of a system of N particles in volume V is as follows:

1. Generate an initial configuration.

2. Start with a configuration o, and calculate its energy $U(o)$.

3. Select a particle at random.

4. Give the selected particle a random displacement $x(n) = x(o) + \Delta$ in which Δ is a uniformly distributed random number from $[-\Delta x, \Delta x]$

5. Calculate the energy $U(n)$ of the new configuration n.

6. Accept the trial move with a probability

$$\text{acc}(o \to n) = min(1, \exp\left[-\beta(U(n) - U(o))\right]) = min(1, \exp\left[-\beta\Delta U\right]) \tag{1.26}$$

7. Update ensemble averages, also after a rejected trial move.

A pseudo computer code of this algorithm is listed in table 1.1. In the next subsections, we would like to make a few remarks on this algorithm.

Question 10 (Update ensemble averages)
Explain why it is necessary to update ensemble averages also after rejected trial moves. Hint: consider a quantum system with 2 *energy levels with energy* 0 *and* ϵ*. What will happen if ensemble averages are only updated after accepted trial moves ?*

1.7.1 Maximum Displacement

In our algorithm, the displacement of a particle is chosen randomly from the interval $[-\Delta x, \Delta x]$, in which Δx is the maximum particle displacement. In principle, our algorithm should work for any value of Δx. However, a very small or very large value of Δx makes our simulation very inefficient. If $\Delta x \approx 0$, particles hardly move and as a consequence $U(n) \approx U(o)$ which means that nearly all moves are accepted. However, these moves are meaningless since the system makes only very small steps in phase space. On the other hand, a very large value of Δx nearly always results in an overlap with another particle, so $\beta\Delta U \gg 1$ and no trial moves are accepted. Therefore, in practice we need to find an optimum and as a rule of thumb we should tune Δx in such a way that on average 50% of all trial moves are accepted. However, the optimum depends on the details of the interaction potential $u(r)$. Note that at low density it is often not possible to tune Δx in such a way that 50% of all trial moves are accepted, as the probability to successfully displace a particle to a random position in the simulation box is larger than 50%. In all cases, Δx should not be larger than half the box size.

Question 11 (Hard-core potential)
A popular potential to model colloidal particles is the so-called hard-core potential

$$u(r) = \begin{cases} \infty & r \leq \sigma \\ 0 & r > \sigma \end{cases} \tag{1.27}$$

For the same density, it is more efficient to use a larger maximum displacement Δx *for this interaction potential than for the Lennard-Jones potential. Explain why.*

Question 12 (Maximum displacement)
For systems at low density, the fraction of accepted displacements is usually very large, irrespective of the maximum displacement Δx*. Explain why.*

1.7.2 Initialization and Length of the Simulation

In our simulation, we have to start with a certain initial configuration $\mathbf{r}^N$. In principle, this could be any configuration. However, if the Boltzmann weight of this configuration is very small, our computed ensemble averages may be strongly influenced by our choice of initial configuration. The reason for this it that we can not average over infinitely long sequences of configurations (Eq. 1.7) so the number of elements in our sequence (n) is a finite number. Therefore, we should avoid preparing our system in a very unfavorable configuration, i.e., a configuration with a

```
  program mc                                basic Monte Carlo algorithm

  do icycle=1,ncycle                        number of MC cycles
     call move                              displace a randomly selected particle
     if(mod(icycle,nsample).eq.0)           each nsample MC cycles
+       call sample                         sample the ensemble averages
  enddo
  end



  subroutine move                           subroutine to calculate the energy

  i=int(ranf()*npart)+1                     select particle i at random
  call energy(x(i),eold,i)                  calculate energy of old configuration
  xnew=x(i)+(2*ranf()-1)*Δx                 random displacement of particle i
  call energy(xnew,enew,i)                  calculate energy of new configuration
  if(ranf().lt.exp(-beta*(enew-eold))       acceptance rule
+    x(i)=xnew                              accept new position of particle i
  return
  end



  subroutine energy(xi,e,i)                 subroutine to calculate the energy

  e=0.0
  do j=1,npart                              loop over all particles
     if(j.ne.i) then                        if particle j is not i
        dx=x(j)-xi                          calculate distance between particle i and j
        dx=dx-box*nint(dx/box)              apply periodic boundary conditions
        eij=4*(1.0/(dx**12)-1.0/(dx**6))    calculate Lennard-Jones potential energy
        e=e+eij
     endif
  enddo
  return
  end
```

Table 1.1: Pseudo-computer code of the Monte Carlo algorithm described in the text. The program consists of three parts. The main program `mc` controls the simulation. The subroutine `move` displaces a randomly selected particle, and the subroutine `energy` computes the energy of a particle. The function `ranf()` generates a uniformly distributed random number between 0 and 1. The function `int` truncates a real number to its integer value, while the function `nint` converts a real number to its nearest integer value (i.e. `int(5.9)=5` and `nint(5.9)=6`).

very low Boltzmann weight as its total energy is high due to for instance particle overlaps. It is therefore essential to equilibrate the system first and throw away the first part of the sequence of configurations (Eq. 1.7).

In practice, we choose a certain number of trial moves and estimate the ensemble average of a certain quantity A ($\langle A\rangle$). Then we repeat the simulation using the final configuration of the previous simulation as our starting configuration. If $\langle A\rangle$ differs from the previous simulation, we repeat this step and increase the number of trial moves. A quick and dirty way to estimate the error in the computed ensemble averages is to perform the same simulation 5 times and to compute the standard deviation.

1.7.3 Checking for errors

There is no straightforward way to debug a Monte Carlo computer code. However, it is usually worthwhile to check whether the energy difference between new and old configurations are calculated correctly. This can be done in the following way:

1. At the start of the simulation, compute the total energy
$$U(\mathrm{start}) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} u(r_{ij}) = \sum_{i<j} u(r_{ij}) \tag{1.28}$$
2. During the simulation, K trial moves are performed, each with energy change ΔU_i, $i = 1, 2, \cdots K$.
3. At the end of the simulation, compute total energy again $U(\mathrm{end})$
4. Within the accuracy of our computer, we should check that $U(\mathrm{end}) = U(\mathrm{start}) + \sum_{i=1}^{K} \Delta U_i$

Chapter 2

Monte Carlo in the NPT and μVT Ensemble

2.1 Introduction

In the previous chapter, we have described the Metropolis Monte Carlo algorithm in the canonical (NVT) ensemble. However, this is not the only ensemble used in simulations. Often, we would like to specify the pressure P instead of the volume V. The reason is that it is usually much easier to specify a pressure than to specify a density. For example, most experiments are performed at a constant pressure of 1 bar rather than at constant volume.

Another important ensemble is the so-called grand-canonical (μVT) ensemble in which the number of particles N is fluctuating and the chemical potential μ is specified. This corresponds for example to the experimental situation where a gas is in contact with a porous medium. The pressure of the gas is directly related to the chemical potential, which controls the amount of gas adsorbed in the pores of a porous medium.

Question 13 (μPT ensemble)
Is the μPT ensemble useful ? Explain your answer.

To derive the Monte Carlo schemes for NPT and μVT ensembles, let us consider a system (volume V, number of particles N, temperature T) that is coupled to a reservoir at the same temperature. This reservoir consists of $M - N$ particles and has a volume $V_0 - V$. See also Fig. 2.1. We will consider the following cases in which the total number of particles ($N+M-N = M$) and the total volume ($V + V_0 - V = V_0$) are fixed:

- The system and the reservoir can exchange volume, but no particles. This corresponds to the NPT ensemble and is discussed in section 2.2.
- The system and the reservoir can exchange particles, but no volume. This corresponds to the μVT ensemble and is discussed in section 2.3.
- The system and the reservoir can exchange both particles and volume. This corresponds to the so-called Gibbs ensemble and this ensemble will be discussed in chapter 3.

2.2 Isobaric-Isothermal (NPT) Ensemble

Below we derive the acceptance rules of constant-pressure Monte Carlo simulations using the classical ensemble theory. The canonical partition function for a system consisting of N particles,

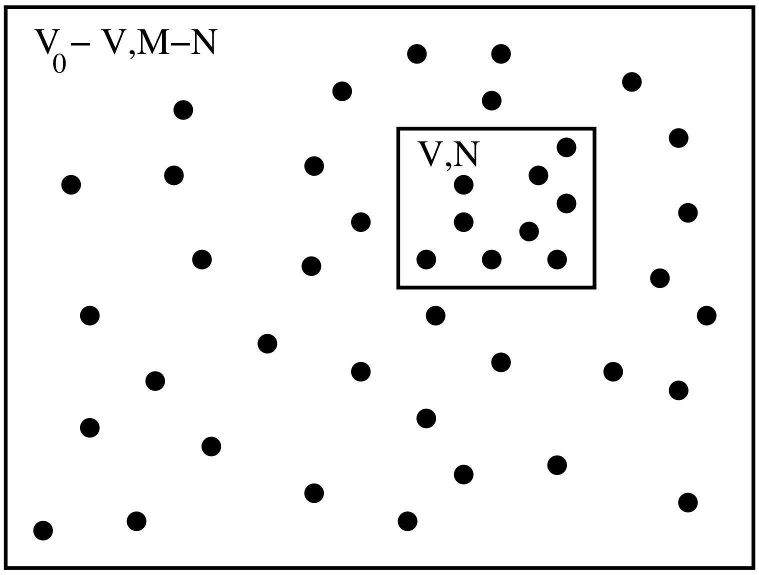


Figure 2.1: A system of N particles with volume V in contact with a reservoir of $M-N$ particles and volume $V_0 - V$.

in a volume V, and at a temperature T is given by

$$Z(N,V,T) = \frac{1}{N!\Lambda^{3N}} \int d\mathbf{r}^N \exp[-\beta U(\mathbf{r}^N)], \tag{2.1}$$

It is convenient to use scaled coordinates $\mathbf{s}^N$ defined by

$$\mathbf{r}_i = L\mathbf{s}_i \quad i = 1, \dots N \tag{2.2}$$

where we assume a cubic box of volume $V = L^3$. The canonical partition function in scaled coordinates equals

$$Z(N,V,T) = \frac{V^N}{N!\Lambda^{3N}} \int_0^1 d\mathbf{s}^N \exp[-\beta U(\mathbf{s}^N; L)] \tag{2.3}$$

The next step is to couple our system of N particles and volume V to the reservoir (Fig. 2.1). Assume that the two systems are separated by a piston from each other and thus, we allow for fluctuations in the volume V (the total volume V_0 is constant) at constant N. In addition, we impose that the particles in the reservoir do not interact and therefore the reservoir is an ideal gas. The total volume of the system and the reservoir is V_0, while the total number of particles is M. The total partition function of the coupled system is the product of the partition functions of the two subsystems.

$$\begin{aligned} Z(N,M,V,V_0,T) &= \frac{V^N (V_0 - V)^{M-N}}{N!(M-N)!\Lambda^{3M}} \int d\mathbf{s}^{M-N} \int d\mathbf{s}^N \exp[-\beta U(\mathbf{s}^N; L)] \\ &= \frac{V^N (V_0 - V)^{M-N}}{N!(M-N)!\Lambda^{3M}} \int d\mathbf{s}^N \exp[-\beta U(\mathbf{s}^N; L)] \end{aligned} \tag{2.4}$$

where we performed the integral over the $\mathbf{s}^{M-N}$ scaled coordinates of the ideal gas reservoir. The probability density $W(V)$ that our system of N particles has a volume V is given by

$$W(V) = \frac{V^N (V_0 - V)^{M-N} \int d\mathbf{s}^N \exp[-\beta U(\mathbf{s}^N; L)]}{\int_0^{V_0} dV' V'^N (V_0 - V')^{M-N} \int d\mathbf{s}^N \exp[-\beta U(\mathbf{s}^N; L')]} \tag{2.5}$$

We now consider the limit that the reservoir tends to infinity, i.e. $V_0 \to \infty, M \to \infty$, while $(M-N)/(V_0-V) \to M/V_0 = \rho$. In that limit, a small volume change of the small system does not change the pressure P of the reservoir and the reservoir acts as a manostat for our system of interest.

Question 14 (Infinite reservoir)
Show that in that limit, we can write

$$(V_0 - V)^{M-N} \approx V_0^{M-N} \exp[-\rho V] = V_0^{M-N} \exp[-\beta PV] \tag{2.6}$$

This results in the following expression for $W(V)$:

$$W(V) = \frac{V^N \exp[-\beta PV] \int d\mathbf{s}^N \exp[-\beta U(\mathbf{s}^N; L)]}{\int_0^{V_0} dV' V'^N \exp[-\beta PV'] \int d\mathbf{s}^N \exp[-\beta U(\mathbf{s}^N; L')]} \tag{2.7}$$

where we used the ideal gas law $\rho = \beta P$. The probability density to find our system in a configuration $\{\mathbf{s}^N\}$ and with a volume V is:

$$\begin{aligned} W(V; \mathbf{s}^N) &\propto V^N \exp[-\beta PV] \exp[-\beta U(\mathbf{s}^N; L)] \\ &\propto \exp[-\beta(U(\mathbf{s}^N; L) + PV - N\beta^{-1} \ln V)] \end{aligned} \tag{2.8}$$

In constant-pressure Monte Carlo simulations, a Metropolis sampling is performed on the reduced coordinates $\{\mathbf{s}^N\}$ and on the volume V. The volume V is treated as an additional coordinate for which we should perform trial moves which will be accepted according to the same rules as trial moves in $\{\mathbf{s}^N\}$. To be more specific, we perform trial moves that consist of an attempt to change the volume from V to $V' = V + \Delta V$, where ΔV is a random number uniformly distributed over the interval $[-\Delta V_{max}, \Delta V_{max}]$. This trial move in the volume will be accepted with a probability ratio

$$\begin{aligned} \frac{\text{acc}(o \to n)}{\text{acc}(n \to o)} &= \frac{\exp[-\beta(U(\mathbf{s}^N; L') + PV' - N\beta^{-1} \ln V')]}{\exp[-\beta(U(\mathbf{s}^N; L) + PV - N\beta^{-1} \ln V)]} \\ &= \exp[-\beta(U(\mathbf{s}^N; L') - U(\mathbf{s}^N; L) + P(V' - V) - N\beta^{-1} \ln(V'/V))] \end{aligned} \tag{2.9}$$

It is important to note that in this trial move, the reduced coordinates of the particles do not change. A schematic overview of the algorithm is presented in table 2.1. It can be shown that the applied pressure P is identical to the average pressure computed using Eq. 1.23.

Question 15 (Random walk in $\ln V$)
In the algorithm described in table 2.1, a random walk is performed in volume V. However, it is also possible to perform a random walk in the logarithm of the volume by using

$$V_{\text{new}} = \exp\left[\ln(V_{\text{old}}) + (2 * \texttt{ranf}() - 1) \times \Delta V\right] \tag{2.10}$$

Derive the correct acceptance rule for this trial move.

Question 16 (Scaling)
Suppose that the cut-off radius r_{cut} is always exactly equal to half the box length. Show that for particles with an interaction potential of the form $u(r) \propto r^{-d}$, the total energy of the system after a change in volume equals

$$U_{\text{new}} = U_{\text{old}} \times \left(\frac{V_{\text{new}}}{V_{\text{old}}}\right)^{-n/3} \tag{2.11}$$

Question 17 (NPT simulation of an ideal gas)
Show using Eq. 2.7 that the ensemble average of the volume $\langle V \rangle$ in an NPT simulation of an ideal gas equals $n/(\beta P)$. Hint: see Eq. 9.33.

```
 program npt-mc                            basic NPT Monte Carlo algorithm

 do icycle=1,ncycle                        number of MC cycles
    if(ranf().lt.pvol) then                select trial move at random
       call volumechange                   perform a volume change
    else
       call move                           move a particle
    endif
    if(mod(icycle,nsample).eq.0)           sample the ensemble averages
+      call sample
 enddo
 end


 subroutine volumechange                   perform a volume change

 call energy(eold)                         compute the old energy
 vold=box**3                               compute old volume
 vnew=vold+(2*ranf()-1)*$\Delta V$         compute new volume
 if(vnew.lt.0) return                      reject negative volume
 boxn=vnew**(1/3)                          compute new boxsize
 do i=1,npart                              scale all coordinates
   x(i)=x(i)*boxn/box
   y(i)=y(i)*boxn/box
   z(i)=z(i)*boxn/box
 enddo
 call energy(enew)                         compute new energy
 arg=-beta*(enew-eold +                    acceptance rule
+  p*(vnew-vold)-N×ln(vnew/vold)/beta
 if(ranf().gt.exp(arg)) then               accept or reject ?
   do i=1,npart                            reject, restore coordinates
     x(i)=x(i)*box/boxn
     y(i)=y(i)*box/boxn
     z(i)=z(i)*box/boxn
   enddo
 else
   box=boxn                                accept, update boxsize
 endif
 return
 end
```

Table 2.1: Pseudo computer code of Metropolis Monte Carlo in the NPT ensemble. The probability that an attempt is made to change the volume equals `pvol` while 1-`pvol` is the probability that an attempt is performed to displace a particle. The subroutine `energy` computes the total energy of the system. `a**b` means a^b. The subroutine `move` is identical to the one in table 1.1.

2.3 Grand-Canonical Ensemble

Below we derive the acceptance rules of Grand-Canonical Monte Carlo simulations, i.e. the chemical potential μ, the volume V, and the temperature T are fixed. To this end, we couple our system of N particles and volume V to an ideal gas reservoir (Fig. 2.1). The two systems can only exchange particles and thus, we allow for particle number fluctuations. The volume of the reservoir ($V_0 - V$) and our system (V) are fixed, while the total number of particles is M ($M - N$ in the reservoir). The total partition function of the coupled system is the product of the partition functions of the two subsystems:

$$\begin{aligned} Z(M,V,V_0,T) &= \sum_{N=0}^{M} \frac{V^N (V_0 - V)^{M-N}}{N!(M-N)!\Lambda^{3M}} \int d\mathbf{s}^{M-N} \int d\mathbf{s}^N \exp[-\beta U(\mathbf{s}^N;L)] \\ &= \sum_{N=0}^{M} \frac{V^N (V_0 - V)^{M-N}}{N!(M-N)!\Lambda^{3M}} \int d\mathbf{s}^N \exp[-\beta U(\mathbf{s}^N;L)] \end{aligned} \tag{2.12}$$

where we performed the integral over the $\mathbf{s}^{M-N}$ scaled coordinates of the ideal gas reservoir. The probability density $W(\mathbf{s}^M;N)$ that our system has N particles at coordinates $\mathbf{s}^N$ in volume V and that the reservoir consists of $M-N$ particles at coordinates $\mathbf{s}^{M-N}$ in volume $V_0 - V$ is given by

$$W(\mathbf{s}^M;N) = \frac{V^N (V_0 - V)^{M-N} \exp[-\beta U(\mathbf{s}^N;L)]}{Z(M,V,V_0,T)\Lambda^{3M} N!(M-N)!} \tag{2.13}$$

The probability of acceptance of a trial move in which a particle is transferred from the ideal reservoir to our system is given by the ratio of the probability densities

$$\begin{aligned} \frac{\text{acc}(N \to N+1)}{\text{acc}(N+1 \to N)} &= \frac{V^{N+1}(V_0 - V)^{M-N-1} \exp[-\beta U(\mathbf{s}^{N+1};L)]}{(N+1)!(M-N-1)!} \\ &\quad \times \frac{(N)!(M-N)!}{V^N (V_0 - V)^{M-N} \exp[-\beta U(\mathbf{s}^N;L)]} \\ &= \frac{V(M-N)}{(V_0 - V)(N+1)} \exp[-\beta(U(\mathbf{s}^{N+1};L) - U(\mathbf{s}^N;L))] \\ &= \frac{V}{\Lambda^3 (N+1)} \exp[\beta(\mu - U(\mathbf{s}^{N+1};L) + U(\mathbf{s}^N;L))] \end{aligned} \tag{2.14}$$

where we have used that the ideal gas reservoir is much larger than our system of interest, i.e. $M \gg N$, $V_0 \gg V$, and $(M-N)/(V_0 - V) \approx M/V_0 = \rho$. In addition, as the reservoir is an ideal gas we can write $\rho = \exp[\beta\mu]/\Lambda^3$ in which μ is the chemical potential of the reservoir. Similarly, we can derive that the removal of a particle is given by the ratio of the probability densities

$$\begin{aligned} \frac{\text{acc}(N \to N-1)}{\text{acc}(N-1 \to N)} &= \frac{V^{N-1}(V_0 - V)^{M-N+1} \exp[-\beta U(\mathbf{s}^{N-1};L)]}{(N-1)!(M-N+1)!} \\ &\quad \times \frac{(N)!(M-N)!}{V^N (V_0 - V)^{M-N} \exp[-\beta U(\mathbf{s}^N;L)]} \\ &= \frac{(V_0 - V)N}{V(M-N+1)} \exp[-\beta(U(\mathbf{s}^{N-1};L) - U(\mathbf{s}^N;L))] \\ &= \frac{N\Lambda^3}{V} \exp[-\beta(\mu + U(\mathbf{s}^{N-1};L) - U(\mathbf{s}^N;L))] \end{aligned} \tag{2.15}$$

```
 program gcmc                                basic μVT Monte Carlo algorithm

 do icycle=1,ncycle                          number of MC cycles
    if(ranf().lt.pexch) then                 select trial move at random
       call exchange                         perform a particle exchange
    else
       call move                             move a particle
    endif
    if(mod(icycle,nsample).eq.0)             sample the ensemble averages
+      call sample
 enddo
 end


 subroutine exchange                         exchange particle
                                             with reservoir
 if(ranf().lt.0.5) then
   if(npart.eq.0) return                     remove a particle
   i=1+int(npart*ranf())                     select random particle
   call ener(x(i),eold,i)                    compute the energy
   arg=npart*exp(beta*eold)/(chem*vol)       acceptance rule
   if(ranf().lt.arg) then                    accept or reject
     x(i)=x(npart)                           accept
     npart=npart-1                           remove the particle
   endif
 else                                        insert new particle
   xnew=ranf()*box                           at random position
   call ener(xn,enew,0)                      compute energy
   arg=chem*vol*exp(-beta*enew)/(npart+1)    acceptance rule
   if(ranf().lt.exp(arg)) then               accept or reject
     npart=npart+1                           accept
     x(npart)=xnew                           add the new particle
   endif
 endif
 return
 end
```

Table 2.2: Pseudo computer code of Metropolis Monte Carlo in the grand-canonical ensemble. We define $chem = \exp[\beta\mu]/\Lambda^3$. The subroutine `move` is defined in table 1.1. The probability that an attempt is made to exchange a particle with the reservoir equals `pexch`. The probability that an attempt is made to add a particle is equal to the probability that an attempt is made to remove a particle. If no particle is present, a particle removal is always rejected.

Question 18 (Grand-canonical ensemble)
Answer the following questions:

- *Does the grand-canonical ensemble work well for solids ?*
- *In principle, one could omit particle displacement in the μVT ensemble. Explain why.*
- *Derive the correct acceptance rules for particle exchange if the probability to select a particle removal is twice as large as selecting a particle addition.*

2.4 Nonideal Gases and Virial Coefficients

Using Monte Carlo simulations in different ensembles (NVT, NPT, μVT) we are able to compute properties of systems of interacting particles. Experimentally, we know that the pressure of a real gas does not satisfy the ideal gas law

$$P = k_B T \rho, \tag{2.16}$$

where $\rho = N/V$ denotes the number density. Rather, we find deviations from the ideal gas law upon increasing ρ. One way of describing the deviations from ideal-gas behaviour is the so-called virial expansion for the pressure

$$P(\rho, T) = k_B T(\rho + +B_2(T)\rho^2 + B_3(T)\rho^3 + \cdots) \tag{2.17}$$

where B_2, B_3 etc. are the second, third, etc. *virial coefficients*. The temperature-dependent virial coefficients can either be obtained by fitting to the observed deviations from ideal-gas behaviour, or one can derive explicit expressions for the virial coefficients in terms of the interaction potential between the particles. For instance, the second virial coefficient is equal to

$$B_2 = \frac{1}{2}\int dr 4\pi r^2 \left(1 - \exp[-\beta\phi(r)]\right) \tag{2.18}$$

Hence, a measurement of the second virial coefficient provides information about the intermolecular interactions. If we assume that molecules are hard spheres with a diameter σ, we find for $r > \sigma$, $\phi(r) = 0$ and hence $\exp[-\beta\phi(r)] = 1$ and for $r < \sigma$, $\phi(r) = \infty$ and hence $\exp[-\beta\phi(r)] = 0$, see Fig. 2.2. Therefore, the second virial coefficient for hard spheres reads

$$B_2^{HS} = \frac{1}{2}\int_0^\sigma 4\pi r^2 dr = \frac{2\pi\sigma^3}{3} \tag{2.19}$$

Question 19 (Square-Well Potential)
Unlike hard spheres, real molecules do not only repel each other at short distances, they also attract at larger distances. A very simple model potential that exhibits both features is the so-called square-well potential. *The square-well potential is equal to the hard-sphere potential for $r < \sigma$. But for $\sigma < r < \lambda\sigma$ (with $\lambda > 1$), the square well potential is attractive:*

$$\phi(r) = \begin{cases} \infty & r \leq \sigma \\ -\epsilon & \sigma < r < \lambda\sigma \\ 0 & r > \lambda\sigma \end{cases} \tag{2.20}$$

see Fig. 2.2.

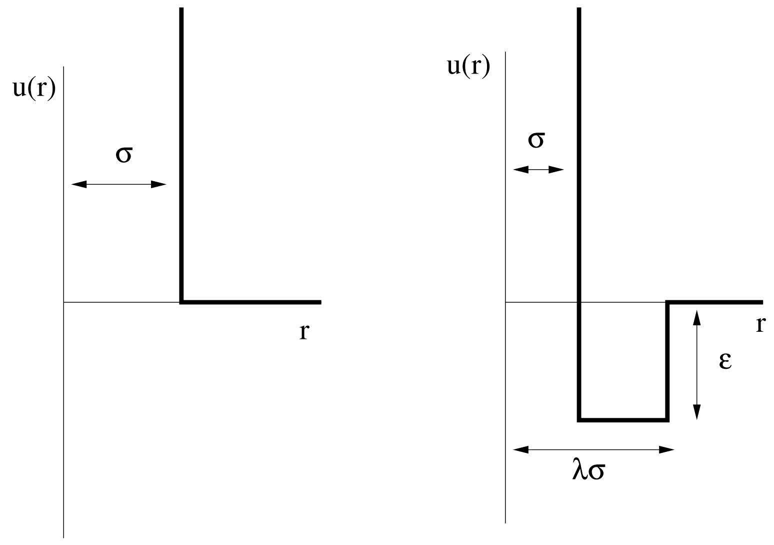


Figure 2.2: Left: the hard-sphere potential. Right: the square-well potential.

1. *Derive that the second virial coefficient of this interaction can be written as*

$$B_2^{SW} = \frac{2\pi\sigma^3}{3}\left(1 + (1 - \exp[\beta\epsilon]) \times (\lambda^3 - 1)\right) \tag{2.21}$$

2. *Show that at very high temperatures ($\beta \to 0$), B_2^{SW} is equal to the hard-sphere second virial coefficient.*

At low temperatures, the term with $\exp[\beta\epsilon]$ *dominates, and B_2 becomes large and negative. The point where B_2 changes sign is called the Boyle temperature. At that temperature B_2 vanishes and the gas behaves almost ideal.*

3. *Show that the Boyle temperature follows from*

$$\frac{k_BT}{\epsilon} = \frac{1}{\ln[\lambda^3/(\lambda^3 - 1)]} \tag{2.22}$$

Question 20 (Boyle Temperature of the Lennard-Jones Potential)
Calculate the second virial coefficient of the Lennard-Jones potential in reduced units ($\epsilon = 1, \sigma = 1$) as a function of temperature and estimate the Boyle temperature. Hint: in general, the integral of the function $f(x)$ can be approximated using

$$\int_a^b f(x)dx \approx h\left[\frac{1}{2}f(x_0) + f(x_1) + f(x_2) + \cdots + f(x_{n-1}) + \frac{1}{2}f(x_n)\right] \tag{2.23}$$

in which $h = (b - a)/n$ and $x_i = a + h \times i$. Note that n should be chosen sufficiently large.

2.5 Equation of State

In the previous section, we described deviations from the ideal gas law by a virial expansion. For sufficiently low densities, only the second virial coefficient $B_2(T)$ is needed to describe the pressure as a function of the density. At higher densities, more terms are needed. An alternative approach which requires less parameters is the use of empirical *equations of state* such as the van der Waals equation. Van der Waals proposed two corrections to the ideal gas law. He argued that the actual volume available to a molecule is smaller than the volume of the system because the finite diameter of each molecule excludes volume, say b, to all the others. Secondly, he argued that the attractions between the molecules reduce the pressure P by an amount $-a\rho^2$, where $a > 0$ is a measure of the strength of the attractions. So Van der Waals proposed

$$P(\rho, T) = \frac{\rho k_B T}{1 - \rho b} - a\rho^2 \tag{2.24}$$

Chapter 3

The Gibbs Ensemble

3.1 Phase Behavior

In simulations, we are often interested in predictions for the phase behavior of a substance. Naively, one can study the phase behavior by performing a simulation at a given statepoint and investigate what happens when one changes the statepoint. For instance, one can perform canonical Monte Carlo simulations, i.e. fix N, V, and T, and measure P or one can perform Isothermal-Isobaric Monte Carlo simulations, i.e. fix N, P, T, and measure V. To be more specific, one can study a system consisting of hard sphere colloids and compress the system until it freezes. However, this method is very inaccurate for the precise location of the transition as large hysteresis will be found when one compresses or expands the system. The hysteresis found for first-order phase transitions is a result of the presence of a large free energy barrier that separates the two coexisting phases. The height of the free energy barrier is determined by the interfacial free energy and increases when the area of the interface between the two coexisting phases increases. Usually, the number of particles in a typical simulation is about 1000. If we like to simulate two coexisting phases and half of the particles belong to phase α and the other half to phase β, than the interface consists of about $2 \times 10 \times 10 = 200$ particles. Thus a considerable fraction of the particles (20%) belongs to the interface. It is therefore, difficult to simulate two coexisting phases simultaneously in a single box as the interfacial free energy is much too high. There are several solutions to this problem, which will be described in the next section.

3.2 Gibbs Ensemble Monte Carlo Simulation

One way to determine phase coexistence is the Gibbs ensemble Monte Carlo method, which was proposed by Panagiotopoulos in 1987 [4,5]. He proposed a method in which the two coexisting phases are simulated simultaneously in two separate boxes and hence, it avoids the interface between the two phases. The thermodynamic conditions for phase equilibria are equal chemical potential (μ), equal pressure (P), and equal temperature (T), i.e. chemical, mechanical, and thermal equilibrium. One might think that the μPT ensemble would be the ideal ensemble to study phase coexistence. However, as already mentioned in Question 13, such an ensemble does not exist, as μ, P, and T are intensive variables, and the extensive variables are unbounded in this ensemble. We have to fix at least one extensive variable to get a well-defined ensemble. It may come therefore as a surprise that we can determine phase coexistence in the Gibbs ensemble method. The reason why this method works is that we fix the difference in chemical potentials between the two coexisting phases, i.e., $\Delta\mu = 0$, and the difference in pressure $\Delta P = 0$, while the absolute values are still undetermined. More precisely, the temperature, the total number of particles in the two boxes, and the total volume of the two boxes are kept fixed. In addition, the two systems can exchange particles *and* volume to ensure equal chemical potential and pressure between the two phases. A Gibbs ensemble Monte Carlo simulation consists of three different trial moves (see table 3.1 and Fig. 3.1): *(1)* a trial move to displace a randomly selected particle in one of the boxes, *(2)* a trial move to exchange volume in such a way that the total volume of the two boxes remains fixed, *(3)* a trial move to transfer a particle from one box to the other. The derivation of the acceptance rules for the trial moves is similar to the ones described in chapter 2 for the various ensembles and will be described below.

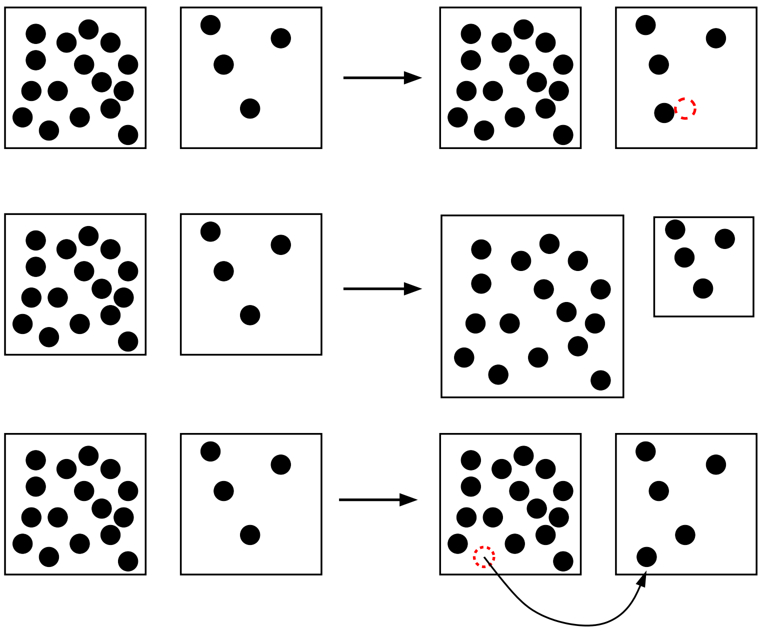


Figure 3.1: A Gibbs ensemble simulation consists of the following trial moves: *(1)* particle displacement, *(2)* exchange of volume and *(3)* exchange of particles.

3.3 The Partition Function

Consider a system of N particles distributed over two volumes V_1 (with N_1 particles) and $V_2 = V - V_1$ (with $N_2 = N - N_1$ particles), where the particles interact with each other in both volumes with the same intermolecular interactions. The volumes V_1 and V_2 can change in such a way that the total volume $V = V_1 + V_2$ remains fixed. The partition function for the total system reads:

$$\begin{aligned} Z(N,V,T) &= \sum_{N_1=0}^{N} \frac{1}{V\Lambda^{3N} N_1!(N-N_1)!} \int_0^V dV_1 V_1^{N_1} (V-V_1)^{N-N_1} \\ &\times \int d\mathbf{s}_1^{N_1} \exp\left[-\beta U(\mathbf{s}_1^{N_1})\right] \int d\mathbf{s}_2^{N-N_1} \exp\left[-\beta U(\mathbf{s}_2^{N-N_1})\right] \end{aligned} \tag{3.1}$$

The probability of finding a configuration with N_1 particles in box 1 with volume V_1 and positions $\mathbf{s}_1^{N_1}$ and $N - N_1$ particles in box 2 with volume $V - V_1$ and positions $\mathbf{s}_2^{N-N_1}$ is given by

$$W(N_1, V_1, \mathbf{s}_1^{N_1}, \mathbf{s}_2^{N-N_1}, T) \propto \frac{V_1^{N_1}(V-V_1)^{N-N_1}}{N_1!(N-N_1)!} \exp\left[-\beta U(\mathbf{s}_1^{N_1}) + U(\mathbf{s}_2^{N-N_1})\right] \tag{3.2}$$

We can now derive the acceptance rules for the trial moves in the Gibbs ensemble Monte Carlo method.

3.3.1 Particle Displacement

One of the trial moves in Gibbs ensemble Monte Carlo simulations is particle displacement. To be more specific, a new configuration is generated by selecting randomly one of the particles in box $i = 1, 2$ and by displacing it randomly. The probability of acceptance of a particle displacement is given by the ratio of the statistical weights of the new and old configurations

$$\frac{W(n)}{W(o)} = \frac{\exp\left[-\beta U(\mathbf{s}_i^{N_i}(n))\right]}{\exp\left[-\beta U(\mathbf{s}_i^{N_i}(o))\right]} \tag{3.3}$$

This acceptance rule is identical to the one used in a conventional NVT ensemble simulations (Eq. 1.26).

Question 21 (Particle displacements)
Consider the following ways to select a particle displacement in the Gibbs ensemble:

- *Pick a particle at random, irrespective of which box it is in.*
- *Pick a box at random, and then pick a particle at random in this box.*

Are the acceptance rules the same ?

3.3.2 Volume Exchange

In addition, we perform trial moves that consist of an attempted change of the old volume $V_1(o)$ of box 1 to a new volume $V_1(n) = V_1(o) + \Delta V$, while the volume of box 2 changes from $V_2(o)$ to $V_2(n) = V_2(o) - \Delta V$. ΔV is a random number uniformly distributed over the interval $[-\Delta V_{max}, \Delta V_{max}]$. This trial move will be accepted with a probability ratio equal to the ratio of the statistical weights of the new and old configuration

$$\frac{W(n)}{W(o)} = \frac{(V_1(n))^{N_1}(V - V_1(n))^{N-N_1}}{(V_1(o))^{N_1}(V - V_1(o))^{N-N_1}} \exp\left[-\beta\left(U(\mathbf{s}^N(n)) - U(\mathbf{s}^N(o))\right)\right] \tag{3.4}$$

resulting in

$$\mathrm{acc}(o \to n) = min\left(1, \left(\frac{V_1(n)}{V_1(o)}\right)^{N_1}\left(\frac{V_2(n)}{V_2(o)}\right)^{N_2} \exp\left[-\beta(U(\mathbf{s}^N(n)) - U(\mathbf{s}^N(o)))\right]\right) \tag{3.5}$$

A schematic overview of the algorithm is presented in table 3.2.

Question 22 (Random walk in $ln(V_1/V_2)$)
It is also possible to perform a random walk in $\ln[V_1/V_2]$ *instead of in* V_1. *Derive the correct acceptance rule for this trial move.*

```
 program Gibbs                      basic Gibbs ensemble Monte Carlo algorithm

 do icycle=1,ncycle                 number of MC cycles
    ran=ranf()*(npart+nvol+nswap)   select trial move at random
    if(ran.le.npart) then
       call move                    move a particle
    elseif(ran.le.(npart+nvol))
       call volumechange            perform a volume change
    else
       call swap                    swap a particle
    endif
    if(mod(icycle,nsample).eq.0)    sample the ensemble averages
+      call sample
 enddo
 end
```

Table 3.1: Pseudo computer code of a Gibbs ensemble Monte Carlo simulation. Each cycle consists of on average `npart` attempts to displace a particle, `nvol` attempts to change the volume and `nswap` attempts to transfer a particle from one to the other box. The subroutine `move` is identical to the one in table 1.1. See also tables 3.3 and 3.2.

3.3.3 Particle Exchange

The third trial move that is used in a Gibbs ensemble Monte Carlo simulation is the exchange of particles. A new configuration is generated from the old configuration by removing a particle from box 1 and inserting this particle in box 2. The ratio of statistical weights of the new and old configuration is given by

$$\frac{W(n)}{W(o)} = \frac{N_1!(N-N_1)!V_1^{N_1-1}(V-V_1)^{N-(N_1-1)}}{(N_1-1)!(N-(N_1-1))!V_1^{N_1}(V-V_1)^{N-N_1}} \exp\left[-\beta(U(\mathbf{s}^N(n)) - U(\mathbf{s}^N(o))\right] \tag{3.6}$$

The acceptance rule is therefore

$$\mathrm{acc}(o \to n) = min\left(1, \frac{N_1 V_2}{(N_2+1)V_1} \exp\left[-\beta(U(\mathbf{s}^N(n)) - U(\mathbf{s}^N(o)))\right]\right) \tag{3.7}$$

A schematic overview of the algorithm is presented in table 3.3.

Question 23 (Gibbs ensemble)
When one of the boxes in the Gibbs ensemble is infinitely large, such that the molecules in this box can be regarded as ideal, the acceptance rules for particle swaps become identical to the acceptance rules for particle swaps in a Grand-canonical ensemble (Eqs. 2.14 and 2.15). Derive this result.

Question 24 (Gibbs ensemble for solid/liquid coexistence)
Is the Gibbs ensemble method described in tables 3.1, 3.2, and 3.3 suitable to study solid/liquid coexistence ?

```
 subroutine volumechange                  perform a volume change

 call energy_tot(box(1),e1old)            compute the old energy of box 1
 call energy_tot(box(2),e2old)            compute the old energy of box 2
 vo1=box(1)**3                            compute old volume box 1
 vo2=box(2)**3                            compute old volume box 2
 vn1=vo1+(2*ranf()-1)*$\Delta V$          compute new volume box 1
 if(vn1.lt.0) return                      reject negative volume
 vn2=v-vn1                                compute new volume box 2
 boxn(1)=vn1**(1/3)                       compute new boxsize box 1
 boxn(2)=vn2**(1/3)                       compute new boxsize box 2
 do i=1,npart                             scale all coordinates
    if(ibox(i).eq.1) then                 check in which box particle i is
       fact=boxn(1)/box(1)
    else
       fact=boxn(2)/box(2)
    endif
   x(i)=x(i)*fact
   y(i)=y(i)*fact
   z(i)=z(i)*fact
 enddo
 call energy_tot(boxn(1),e1new)           compute new energy box 1
 call energy_tot(boxn(2),e2new)           compute new energy box 2
 arg=-beta*(e1new+e2new-e1old-e2old -     acceptance rule
+  (npbox(1)*ln(vn1/vo1)+
+   npbox(2)*ln(vn2/vo2))/beta)
 if(ranf().gt.exp(arg)) then              accept or reject ?
   do i=1,npart                           reject, restore coordinates
      if(ibox(i).eq.1) then               check in which box particle i is
         fact=box(1)/boxn(1)
      else
         fact=box(2)/boxn(2)
      endif
      x(i)=x(i)*fact
      y(i)=y(i)*fact
      z(i)=z(i)*fact
   enddo
 else
   box(1)=boxn(1)                         accept, update boxsize
   box(2)=boxn(2)
 endif
 return
 end
```

Table 3.2: Pseudo computer code for a volume change in a Gibbs ensemble Monte Carlo simulation. The subroutine `energy_tot` computes the total energy of the system. `a**b` means a^b.

```
  subroutine swap                    attempt to swap a particle

  if (ranf().lt.0.5) then            select boxes at random
     in=1                            transfer from box out to box in
     out=2
  else
     in=2
     out=1
  endif
  xn=box(in)*ranf()                  add a particle to box in
  yn=box(in)*ranf()                  at a random position
  zn=box(in)*ranf()
  call ener(xn,yn,zn,enn,in)         calculate energy of new particle in box in
  if(npbox(out).eq.0) return         delete particle from box out, if box empty return
  ido=0                              find a particle to be removed
  do while (ido.ne.out)
     ipart=int(npart*ranf())+1
     ido=ibox(ipart)
  enddo
  xo=z(ipart)                        old configuration
  yo=y(ipart)
  zo=z(ipart)
  call ener(xo,yo,zo,eno,out)        calculate energy particle o in box out
  arg=exp(-beta*(enn-eno +
+ log(vol(out)*npbox(in)+1)/         acceptance rule
+ (vol(in)*npbox(out)))/beta))
  if(ranf().lt.arg) then
    x(ipart)=xn
    y(ipart)=yn                      add new particle to box in
    z(ipart)=zn
    ibox(ipart)=in
    npbox(out)=npbox(out)-1
    npbox(in)=npbox(in)+1
  endif
  return
  end
```

Table 3.3: Pseudo computer code for an attempt to swap a particle between the two boxes in a Gibbs ensemble Monte Carlo simulation. The subroutine `ener` computes the energy of a particle at a given position in a certain box.

3.4 Analyzing the Results

In a Gibbs ensemble simulation, the densities of the two coexisting phases can be obtained simply by sampling the densities during the simulation. Of course, in estimating the standard deviations of the results, one should be careful since the two coexisting densities are not independently. Close to the critical point (see Fig. 5.2), it is possible that the boxes change identity during a simulation and the measured density will tend to be the overall density N/V. In this case, it is more convenient to determine the probability density $P(\rho)$, which is equal to the probability to observe a density ρ. The two maxima of $P(\rho)$ correspond to the coexisting densities. However, close to the critical point the Gibbs ensemble method will not work well as the free energy associated with creating a phase separation will become very small. Instead, we can estimate the location of the critical point by using the following scaling relations [3]

$$\frac{\rho_g + \rho_l}{2} = \rho_c + A \times (T_c - T) \tag{3.8}$$

$$\rho_l - \rho_g = B \times (T_c - T)^{\alpha} \tag{3.9}$$

in which $\alpha \approx 0.32$ for three-dimensional systems, A and B are fit parameters and the subscripts g, l, and c are used to denote the gas, liquid, and critical phases respectively.

Chapter 4

Free-Energy Calculations

Computing the free energy of a system as a function of temperature, volume, and number of particles is essential for locating phase equilibria. In principle, $F(N,V,T)$ can be computed by integrating over the positions of all N particles in the system

$$F(N,V,T) = -k_B T \ln Z(N,V,T) = -k_B T \ln \left(\frac{\int d\mathbf{r}^N \exp\left[-\beta U\left(\mathbf{r}^N\right)\right]}{N!\Lambda^{3N}} \right) \tag{4.1}$$

in which $\beta = 1/k_B T$ and $\mathbf{r}^N$ is a $3N$ dimensional vector containing the positions of all particles. In chapter 1 we have shown that for most systems we cannot compute this integral directly, except for a few trivial cases such as the ideal gas and the Einstein crystal (see chapter 5). This suggests that we are not able to compute the free energy of a system of interacting particles. However, it turns out that we are able to compute free energy differences between systems by thermodynamic integration.

Directly related to the free energy is the chemical potential of component i, which is defined as the partial derivative of the free energy with respect to N_i while keeping V, T, and N_j $(j \neq i)$ constant

$$\mu_i = \left(\frac{\partial F}{\partial N_i} \right)_{T,V,N_j(j\neq i)} \tag{4.2}$$

If two phases are in equilibrium, the temperatures, pressures, and chemical potentials of the coexisting phases must be identical. For a single component system (only one type of particles), there is a direct relation between the free energy F and the chemical potential μ:

$$F = -PV + \mu N \tag{4.3}$$

in which V is the volume of the system and N is the number of particles. Alternatively, we can examine the Gibbs free energy G

$$G = E - TS + PV = F + PV \tag{4.4}$$

which for a single component is directly related to the chemical potential μ

$$G = \mu N \tag{4.5}$$

As the chemical potentials of the coexisting phases must be identical, the Gibbs free energy per particle

$$\bar{G} = \frac{G}{N} = \mu \tag{4.6}$$

must be equal for both phases. The chemical potential can either be computed from the free energy (Eq. 4.3), or by Widom test particle method (see section 4.3).

4.1 Derivatives of the Free Energy

We can compute differences in free energy by using the usual thermodynamic relations. For example, the free energy change when the volume or temperature of a system is changed equals

$$\left(\frac{\partial F}{\partial V}\right)_{N,T} = -P \tag{4.7}$$

$$\left(\frac{\partial (F/T)}{\partial T}\right)_{N,V} = -\frac{E}{T^2} \tag{4.8}$$

in which P is the pressure and E is the total energy of the system. By integration of these equations we can compute the free energy at another temperature or volume. For example, to compute the free energy at temperature T_2 when the free energy at temperature T_1 is known, we can use:

$$\frac{F\left(V,T_2\right)}{T_2} - \frac{F\left(V,T_1\right)}{T_1} = \int_{T_1}^{T_2} dT \left(\frac{\partial (F/T)}{\partial T}\right)_{N,V} = -\int_{T_1}^{T_2} dT \frac{E\left(V,T\right)}{T^2} \tag{4.9}$$

Question 25 (Free energy of an ideal gas)
Show that the free energy of an ideal gas equals (N particles in volume V, $\rho = N/V$)

$$\frac{F_{IG}}{Nk_BT} = \ln \Lambda^3 \rho - 1 + \frac{\ln \sqrt{2\pi N}}{N} \tag{4.10}$$

See also Eq. 9.41.

Question 26 (Maxwell relation)
Show that

$$\left(\frac{\partial F/N}{\partial \rho}\right)_{T} = \frac{P(\rho)}{\rho^2} \tag{4.11}$$

Question 27 (Excess free energy)
An important property is the so-called excess free energy F_{ex} of a system, which equals the free energy of the system F minus the free energy if the system would be an ideal gas F_{IG}. Show that at constant temperature T we can write

$$\frac{F_{ex}(\rho)}{Nk_BT} = \frac{F(\rho)-F_{IG}(\rho)}{Nk_BT} = \frac{1}{k_BT}\int_0^{\rho} d\rho' \frac{P(\rho') - \rho' k_BT}{\rho'^2} \tag{4.12}$$

in which $P(\rho)$ is the equation of state of the system.

4.2 Arbitrary Order Parameter

Not only are we able to compute changes in the free energy when T or ρ is changed, but we can also compute the free energy change when an arbitrary order parameter λ is changed at constant N, V, and T.

$$\begin{aligned}\left(\frac{\partial F}{\partial \lambda}\right)_{N,V,T} &= -\frac{1}{\beta}\frac{\partial}{\partial \lambda}\ln Z\left(N,V,T,\lambda\right) \\ &= -\frac{1}{\beta Z}\frac{\partial Z\left(N,V,T,\lambda\right)}{\partial \lambda} \\ &= \frac{\int d\mathbf{r}^N \left(\frac{\partial U(\mathbf{r}^N,\lambda)}{\partial \lambda}\right)\exp\left[-\beta U\left(\mathbf{r}^N,\lambda\right)\right]}{\int d\mathbf{r}^N \exp\left[-\beta U\left(\mathbf{r}^N,\lambda\right)\right]} = \left\langle \frac{\partial U\left(\mathbf{r}^N,\lambda\right)}{\partial \lambda}\right\rangle_{\lambda}\end{aligned} \tag{4.13}$$

where we have changed the order of integrating over $\mathbf{r}^N$ and differentiating with respect to λ. The brackets $\langle \cdots \rangle_\lambda$ denote an ensemble average at a fixed value of λ. This ensemble average

$$\left\langle \frac{\partial U\left(\mathbf{r}^N, \lambda\right)}{\partial \lambda} \right\rangle_\lambda \tag{4.14}$$

can be calculated in a simulation. Therefore, the free energy difference between $\lambda = 0$ and $\lambda = 1$ equals

$$\Delta F = F(\lambda = 1) - F(\lambda = 0) = \int_0^1 d\lambda \left(\frac{\partial F}{\partial \lambda}\right)_{N,V,T} = \int_0^1 d\lambda \left\langle \frac{\partial U\left(\mathbf{r}^N, \lambda\right)}{\partial \lambda} \right\rangle_\lambda \tag{4.15}$$

In practice, we need of the order of 10 simulations at different values of λ to evaluate this integral numerically. In the next chapter, we will use this method to compute the free energy of a solid.

Question 28 (Gibbs-Bogoliubov inequality)
Suppose that we would like to compute the free energy difference between a system for which $\lambda = 0$ and a system for which $\lambda = 1$. In addition, the energy depends linearly on the coupling parameter λ:

$$U(\mathbf{r}^N, \lambda) = (1 - \lambda) \times U_I(\mathbf{r}^N, \lambda) + \lambda \times U_{II}(\mathbf{r}^N, \lambda) \tag{4.16}$$

Show that in this case

$$\left(\frac{\partial^2 F}{\partial \lambda^2}\right)_{N,V,T} \leq 0 \tag{4.17}$$

4.3 Computing the Chemical Potential

A simple method to compute the chemical potential is Widom test particle method, which estimates the derivative of Eq. 4.2:

$$\mu \approx \frac{F(N+1,V,T) - F(N,V,T)}{N+1-N} = -k_B T \ln \frac{Z(N+1,V,T)}{Z(N,V,T)} \tag{4.18}$$

Using Eq. 1.2 and reduced coordinates ($s_i = r_i/L$; $V = L^3$) leads to

$$\begin{aligned}
\mu &= -k_B T \ln\left(\frac{V\Lambda^{-3}}{N+1}\right) - k_B T \ln\left(\frac{\int d\mathbf{s}^{N+1} \exp\left[-\beta U(\mathbf{s}^{N+1})\right]}{\int d\mathbf{s}^N \exp\left[-\beta U(\mathbf{s}^N)\right]}\right) \\
&= -k_B T \ln\left(\frac{V\Lambda^{-3}}{N+1}\right) - k_B T \ln\left(\frac{\int d\mathbf{s}^N \exp\left[-\beta U(\mathbf{s}^N)\right] \int d\mathbf{s}_{N+1} \exp\left[-\beta U(\mathbf{s}_{N+1})\right]}{\int d\mathbf{s}^N \exp\left[-\beta U(\mathbf{s}^N)\right]}\right) \\
&\approx k_B T \ln(\rho\Lambda^3) - k_B T \ln \left\langle \exp\left[-\beta \Delta U^+\right] \right\rangle_{N,V,T} \\
&= \mu_{IG} + \mu_{ex}
\end{aligned} \tag{4.19}$$

in which $\Delta U^+ = U(\mathbf{s}^{N+1}) - U(\mathbf{s}^N)$ is the energy change when a test particle is inserted at a certain position in the system and $\mu_{IG} = k_B T \ln(\rho\Lambda^3)$ is the chemical potential of an ideal gas. Note that these insertions are never accepted. We should keep in mind that the ensemble average

$$\left\langle \exp\left[-\beta \Delta U^+\right] \right\rangle_{N,V,T} \tag{4.20}$$

is an average over all possible positions of the test particle *and* an Boltzmann average for our system of N particles at volume $V = L^3$ and temperature T. In practice, we simulate a system of N particles in volume V at temperature T using the conventional Monte Carlo algorithm. During this simulation, we keep track of the ensemble average of $\exp\left[-\beta\Delta U^{+}\right]$, in which ΔU^{+} is the energy change when an additional particle is inserted into the system at a random position (without ever accepting such an insertion).

Question 29 (Widom test particle method)
Explain why Widom test particle method does not work well at high densities.

Question 30 (Particle removal)
Instead of Eq. 4.18, we could also compute the chemical potential by computing the energy change when a randomly selected particle is removed from the system

$$\mu = k_B T \ln \frac{Z(N,V,T)}{Z(N+1,V,T)} \approx \mu_{IG} + k_B T \ln \left\langle \exp\left[\beta\Delta U^{+}\right]\right\rangle_{N+1,V,T} \tag{4.21}$$

in which $\Delta U^{+} = U(\mathbf{s}^{N+1}) - U(\mathbf{s}^{N})$. This requires a simulation of a system of $N+1$ particles. Explain why this method does not work for hard-core potentials. Explain that this method is not practical to compute the chemical potential of a system consisting of Lennard-Jones particles, although it is in principle correct.

4.4 Umbrella Sampling

In a Monte Carlo simulation, we often compute averages in the canonical ensemble

$$\langle A\rangle = \frac{\int d\mathbf{r}^N A(\mathbf{r}^N)\exp\left[-\beta U(\mathbf{r}^N)\right]}{\int d\mathbf{r}^N \exp\left[-\beta U(\mathbf{r}^N)\right]} \tag{4.22}$$

However, it is straightforward to perform the simulation in a slightly different ensemble π, in which configurations are sampled with a probability proportional to

$$\pi(\mathbf{r^N}) = \exp[-\beta U(\mathbf{r^N}) + W(\mathbf{r^N})] \tag{4.23}$$

in which $W(\mathbf{r^N})$ is a weight function (sometimes called biasing function) that only depends on $\mathbf{r^N}$. The ensemble average $\langle A\rangle$ in the canonical ensemble can be computed using

$$\begin{aligned}
\langle A\rangle &= \frac{\int d\mathbf{r}^N A(\mathbf{r}^N)\exp\left[-\beta U(\mathbf{r}^N)\right]}{\int d\mathbf{r}^N \exp\left[-\beta U(\mathbf{r}^N)\right]} \\
&= \frac{\int d\mathbf{r}^N A(\mathbf{r}^N)\exp\left[-W(\mathbf{r}^N)\right]\exp\left[-\beta U(\mathbf{r}^N)+W(\mathbf{r}^N)\right]}{\int d\mathbf{r}^N \exp\left[-W(\mathbf{r}^N)\right]\exp\left[-\beta U(\mathbf{r}^N)+W(\mathbf{r}^N)\right]} \\
&= \frac{\frac{\int d\mathbf{r}^N A(\mathbf{r}^N)\exp\left[-W(\mathbf{r}^N)\right]\exp\left[-\beta U(\mathbf{r}^N)+W(\mathbf{r}^N)\right]}{\int d\mathbf{r}^N \exp\left[-\beta U(\mathbf{r}^N)+W(\mathbf{r}^N)\right]}}{\frac{\int d\mathbf{r}^N \exp\left[-W(\mathbf{r}^N)\right]\exp\left[-\beta U(\mathbf{r}^N)+W(\mathbf{r}^N)\right]}{\int d\mathbf{r}^N \exp\left[-\beta U(\mathbf{r}^N)+W(\mathbf{r}^N)\right]}} \\
&= \frac{\langle A\exp[-W]\rangle_\pi}{\langle exp[-W]\rangle_\pi}
\end{aligned} \tag{4.24}$$

in which $\langle\cdots\rangle_\pi$ denotes an ensemble average in the ensemble π (Eq. 4.23). Applying a biasing potential can be useful when we wish to confine our system into a certain region of the phase space. Moreover, it is sometimes useful to split a simulation into a number of different simulations, each with a slightly different window potential $W(\mathbf{r}^N)$, and combine all the simulations afterward using Eq. 4.24.

Question 31 (Ensemble averages at different temperatures)
Suppose that we perform a Monte Carlo simulation at a certain temperature T. Using Eq. 4.24, we could in principle compute ensemble averages at any other temperature T^* different from T. Explain how this can be done. However, it turns out that in practice, this method only works well when $|T - T^*|$ is small. Explain this.

Chapter 5

Free Energy of Solids

A disadvantage of the Gibbs ensemble Monte Carlo method is that it can only be used for the determination of fluid equilibria as it involves the insertion and removal of particles, which is impossible for solid phases. If we like to determine phase equilibria for dense fluids or solids, we often resort to free energy calculations using thermodynamic integration to a reference system for which we do know the free energy. Only in a few cases the free energy is known explicitly. To compute the Helmholtz free energy of a dense fluid, one can construct a reversible path to a system for which the free energy is known, e.g. the ideal gas. However, for a solid, a direct path to the ideal gas without crossing a phase transition is not possible as the solid/liquid coexistence does not have a critical point (Fig. 5.2). Therefore, we will use another reference state, the so-called Einstein crystal. The free energy difference of a real crystal and the Einstein crystal can be computed using an auxiliary Hamiltonian, which depends on a coupling parameter λ (Eq. 4.13).

5.1 Einstein Integration

Rather than the ideal gas phase, a more useful reference system for a crystal is the so-called Einstein lattice, for which there is an exact expression for the free energy. In this system, particles are bound to grid positions by harmonic springs while the particles themselves do not interact with each other (Fig. 5.1). Each particle has a different equilibrium position in such a way that these grid positions resemble the real crystal as much as possible.

Consider a single particle that is bound with a spring to a grid position. The energy of this particle equals

$$u\left(r\right) = \frac{\alpha}{2} \times r^2 \tag{5.1}$$

in which r is the distance to the fixed grid position and α is a spring constant. In three dimensions, we can compute the partition sum by using spherical coordinates

$$Z = \int_0^\infty dr 4\pi r^2 \exp\left[-\beta u\left(r\right)\right] = \int_0^\infty dr 4\pi r^2 \exp\left[-\frac{1}{2}\alpha\beta r^2\right]. \tag{5.2}$$

Using this so-called Einstein crystal as a reference state for the free energy, the next step in computing the free energy of a real crystal is to construct a reversible path from the real crystal to the Einstein crystal. This can be done by introducing an order parameter λ with $0 \leq \lambda \leq 1$. In a computer simulation, we can simulate a crystal with the following potential:

$$U(\mathbf{r}^N, \lambda) = \lambda U_{ein}(\mathbf{r}^N) + (1-\lambda) U_{cr}(\mathbf{r}^N) \tag{5.3}$$

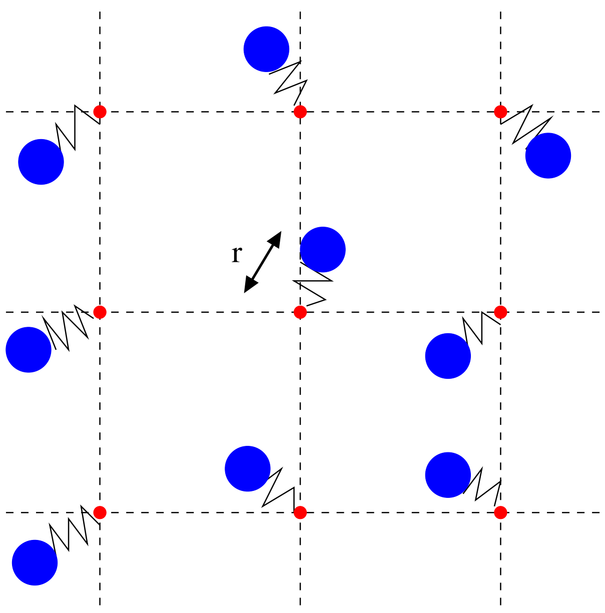


Figure 5.1: Schematic representation of an Einstein crystal. Atoms are bound to their lattice positions by harmonic springs and do not interact.

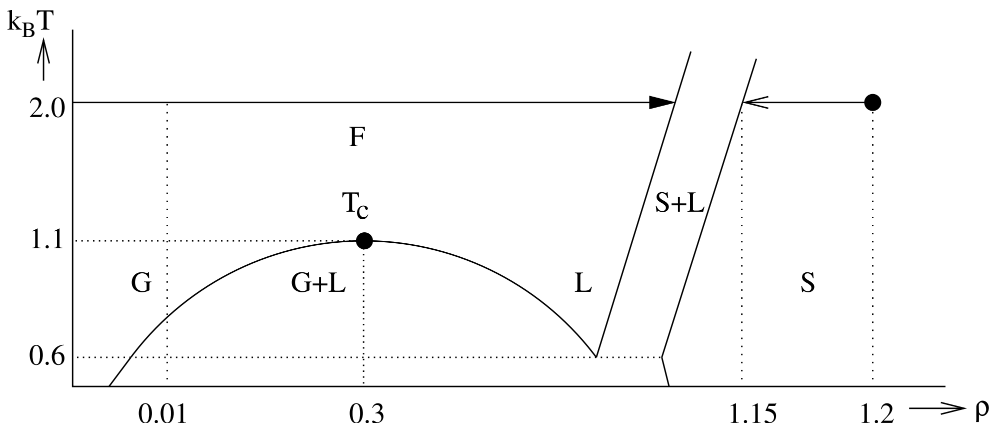


Figure 5.2: Schematic representation of the phase diagram of the Lennard-Jones system. The different phases are solid (S), liquid (L), and gas (G). Above the critical temperature ($k_BT_c \approx 1.1$) there is no vapor/liquid coexistence possible anymore; the resulting phase is called a fluid (F). The solid/liquid coexistence does not have a critical point.

in which U_{ein} is the potential energy that the system would have if it would be an Einstein crystal and U_{cr} is the potential energy if the system would be a real crystal. Thus, for $\lambda = 0$ we recover the real system and for $\lambda = 1$ we have the Einstein crystal for which we have an exact expression for the free energy. The free energy difference of the real crystal and the Einstein crystal is (Eq. 4.13)

$$\Delta F = F_{ein} - F_{cr} = \int_0^1 d\lambda \left(\frac{\partial F}{\partial \lambda} \right)_{N,V,T,\lambda} = \int_0^1 d\lambda \left\langle \frac{\partial U}{\partial \lambda} \right\rangle_\lambda = \int_0^1 d\lambda \left\langle U_{ein} - U_{cr} \right\rangle_\lambda . \tag{5.4}$$

However, there is a subtle difficulty. The problem is that fluctuations in U_{ein} become very large when $\lambda \to 0$. The reason for this is that for $\lambda = 0$ the particles are no longer bound to their lattice position and can in principle move freely. Therefore, $\langle r^2 \rangle$ will be of the order of the square of the system size which may be quite large. This problem is solved by performing the integration of Eq. 5.4 using a fixed center of mass resulting in ΔF^{CM}

$$\Delta F^{CM} = F_{ein}^{CM} - F_{cr}^{CM} = \int_0^1 d\lambda \left\langle U_{ein} - U_{cr} \right\rangle_{\lambda,CM} \tag{5.5}$$

in which the brackets $\langle \cdots \rangle_{\lambda,CM}$ denote an ensemble at a fixed value of λ and at a fixed center of mass. It is important to note that ensemble averages with and without a fixed center of mass are different, i.e.

$$\int_0^1 d\lambda \left\langle U_{ein} - U_{cr} \right\rangle_{\lambda,CM} \neq \int_0^1 d\lambda \left\langle U_{ein} - U_{cr} \right\rangle_\lambda . \tag{5.6}$$

However, we would like to compute the free energy of a system in which the center of mass is not fixed because this corresponds to the "real" system. The resulting expression for the free energy of the unconstrained system is derived in the paper by Polson *et al.* [6]. Most important is Eq. 17 of this paper to compute the excess free energy F_{ex}, which is the free energy of the system (F) minus the free energy of the system if it would be an ideal gas (F_{IG}),

$$F_{ex}(N,V,T) = F(N,V,T) - F_{IG}(N,V,T) \tag{5.7}$$

$$\frac{\beta F_{ex}}{N} = -\frac{3}{2} \ln \frac{2\pi}{\alpha\beta} - \frac{3}{2N} \ln \frac{\alpha\beta}{2\pi} + \frac{\ln \rho}{N} - \frac{2 \ln N}{N} - \ln \rho + 1 - \frac{\ln 2\pi}{2N} - \frac{\beta}{N} \Delta F^{CM} \tag{5.8}$$

in which ΔF^{CM} is given by Eq. 5.5 and F_{IG} by Eq. 4.10.

Question 32 (Free energy of an Einstein crystal)
Derive that the free energy of a crystal containing N particles that are all bound to different lattice positions with equal spring constants α, equals

$$F = -k_B T \ln Z = -\frac{3N}{2\beta} \ln \frac{2\pi}{\alpha\beta}. \tag{5.9}$$

You will need the following standard integral

$$\int_0^\infty dx x^2 \exp\left[-x^2\right] = \frac{\sqrt{\pi}}{4}. \tag{5.10}$$

Question 33 (Mean-square displacement)
Show that the mean-squared displacement of an Einstein crystal equals

$$\left\langle r^2 \right\rangle = \frac{3}{2\beta\lambda} \tag{5.11}$$

Question 34 (Hard-sphere solid)
Suppose we would like to study the solid/liquid coexistence of particles interacting with a hard-core potential (Eq. 1.27). Explain why the reversible path of Eq. 5.3 will not work in this case. Instead, we can switch on the spring constants while leaving the hard-core interactions between the particles unaffected:

$$U(\lambda) = U_0 + \lambda \sum_{i=1}^{N} (\mathbf{r}_i - \mathbf{r}_{i,0})^2 \tag{5.12}$$

in which $\mathbf{r}_{i,0}$ *is the equilibrium lattice position of particle* i *and* U_0 *is the energy of the crystal with hard-core interactions. Describe how the free energy of the real crystal should be computed using this method.*

Chapter 6

The pair correlation function

To describe the thermodynamic and structural properties of dense liquids, correlation or distribution functions were introduced in the 1920s, e.g.,

$$\begin{aligned}\rho^{(1)}(\mathbf{r}) &= \langle\sum_{i=1}^{N}\delta(\mathbf{r}-\mathbf{r}_i)\rangle &(6.1)\\ \rho^{(2)}(\mathbf{r},\mathbf{r}') &= \langle\sum_{i=1}^{N}\sum_{j\neq i}^{N}\delta(\mathbf{r}-\mathbf{r}_i)\delta(\mathbf{r}'-\mathbf{r}_j)\rangle, &(6.2)\end{aligned}$$

The first distribution function is called the one-particle distribution and the second function, the pair distribution function. Higher order distributions can be defined accordingly, but we do not need them here because we restrict attention to pairwise interactions. The angular brackets in Eqs. 6.1 and 6.2 denote an ensemble average, either canonical or grand canonical. $\rho^{(1)}(\mathbf{r})$ is a measure for the probability density that a particle is present at position $\mathbf{r}$. Because of the normalisation $\int d\mathbf{r}\rho^{(1)}(\mathbf{r}) = N$, we see that $\rho^{(1)}(\mathbf{r})$ is the local density, and equals $\rho = N/V$ in a homogeneous bulk system. $\rho^{(2)}(\mathbf{r},\mathbf{r}')$ is called the pair distribution function, and is a measure for the probability that there is a particle at position $\mathbf{r}$ and *another* one at $\mathbf{r}'$ *simultaneously*. Within the canonical ensemble, we obtain from Eq. 6.2 that

$$\begin{aligned}\rho^{(2)}(\mathbf{r},\mathbf{r}') &= \frac{1}{Q_N}\int d\mathbf{r}^N \exp[-\beta U(\mathbf{r}^N)]\left(\sum_{i=1}^{N}\sum_{j\neq i}^{N}\delta(\mathbf{r}-\mathbf{r}_i)\delta(\mathbf{r}'-\mathbf{r}_j)\right)\\ &= \frac{N(N-1)}{Q_N}\int d\mathbf{r}^N \exp[-\beta U(\mathbf{r}^N)]\delta(\mathbf{r}-\mathbf{r}_1)\delta(\mathbf{r}'-\mathbf{r}_2)\\ &= \frac{N(N-1)}{Q_N}\int d\mathbf{r}_3\cdots\mathbf{r}_N \exp[-\beta U(\mathbf{r},\mathbf{r}',\mathbf{r}_3,\cdots,\mathbf{r}_N)]. &(6.3)\end{aligned}$$

Recall the definition of the configuration integral

$$Q_N = Q(N,V,T) = \int d\mathbf{r}^N \exp[-\beta U(\mathbf{r}^N)]. \tag{6.4}$$

At sufficiently long distances $|\mathbf{r}-\mathbf{r}'|$ these probabilities become uncorrelated, and we have $\rho^{(2)}(\mathbf{r},\mathbf{r}') \to \rho^{(1)}(\mathbf{r})\rho^{(1)}(\mathbf{r}')$. In isotropic, homogeneous systems such as liquids and gases we can use translational invariance to define the *radial distribution function* $g(r)$ by

$$\rho^{(2)}(\mathbf{r},\mathbf{r}') = \rho^2 g(|\mathbf{r}-\mathbf{r}'|) = \rho^2 g(r). \tag{6.5}$$

Note that $\rho g(r)$ is the average particle density at a distance r from a fixed particle. If we take a fixed particle at the origin and we move out along any straight line from the origin, the average particle density can vary with r reflecting the short-range order. At short distances, $\rho(r)$ will sometimes be more, sometimes less, or sometimes equal to the bulk density ρ and thus, $g(r)$ can be greater than, less than or equal to unity. Also note that $\lim_{r\to\infty} g(r) = 1$.

For systems with pairwise additive interactions the thermodynamics follows completely from $g(r)$, that is from $g(r;\rho,T)$ in the canonical ensemble or from $g(r;\mu,T)$ in the grand canonical ensemble. There are three independent routes from $g(r)$ to thermodynamics, viz.

$$\begin{aligned}
p &= \rho kT - \frac{\rho^2}{6}\int d\mathbf{r} r u'(r) g(r) \quad \text{(virial route)} && (6.6)\\
\frac{E}{V} &= \frac{3}{2}\rho kT + \frac{\rho^2}{2}\int d\mathbf{r} u(r) g(r) \quad \text{(caloric route)} && (6.7)\\
kT\left(\frac{\partial \rho}{\partial p}\right)_T &= 1 + \rho\int d\mathbf{r}\big(g(r)-1\big) \quad \text{(compressibility route)} && (6.8)
\end{aligned}$$

The virial and caloric route follow straightforwardly from the canonical partition function of a pairwise additive system, as will be shown in one of the problems. The compressibility route is necessarily derived grand canonically, and follows directly from the normalisation $\int d\mathbf{r}d\mathbf{r}'\rho^{(2)}(\mathbf{r},\mathbf{r}') = N(N-1) = \langle N^2\rangle - \langle N\rangle$ and from

$$\frac{\langle N^2\rangle - \langle N\rangle^2}{\langle N\rangle} = kT\left(\frac{\partial \rho}{\partial p}\right)_T. \tag{6.9}$$

Knowledge of $g(r)$ does not only lead to the thermodynamics of the fluid, but also to the structure factor $S(q)$ (that can be measured in scattering experiments). The relation between $S(q)$ and $g(r)$ is obtained as follows,

$$\begin{aligned}
S(q) &\equiv \langle \frac{1}{N}\sum_{i,j}^{N} \exp[i\mathbf{q}\cdot\mathbf{r}_{ij}]\rangle && (6.10)\\
&= 1 + \langle \frac{1}{N}\sum_{i\neq j}^{N} \exp[i\mathbf{q}\cdot\mathbf{r}_{ij}]\rangle\\
&= 1 + \frac{1}{NQ_N}\int d\mathbf{r}^N \exp[-\beta U(\mathbf{r}^N)] \sum_{i\neq j}^{N} \exp[i\mathbf{q}\cdot\mathbf{r}_{ij}]\\
&= 1 + \frac{1}{N}\int d\mathbf{r}_1 d\mathbf{r}_2 \exp[i\mathbf{q}\cdot\mathbf{r}_{12}]\\
&\qquad \left(\frac{N(N-1)}{Q_N}\int d\mathbf{r}_3\cdots d\mathbf{r}_N \exp[-\beta U(\mathbf{r}^N)]\right)\\
&\overset{6.3}{=} 1 + \frac{1}{N}\int d\mathbf{r}_1 d\mathbf{r}_2 \exp[i\mathbf{q}\cdot\mathbf{r}_{12}]\rho^{(2)}(\mathbf{r}_1,\mathbf{r}_2)\\
&\overset{6.5}{=} 1 + \rho\int d\mathbf{r} \exp[i\mathbf{q}\cdot\mathbf{r}] g(r), && (6.11)
\end{aligned}$$

i.e. $S(q)$ is essentially the Fourier transform of $g(r)$. Since $g(r)$ approaches unity for large r it is convenient to rewrite Eq. 6.11 as

$$S(q) = 1 + \rho\int d\mathbf{r} \exp[i\mathbf{q}\cdot\mathbf{r}]\big(g(r)-1\big) + (2\pi)^3\rho\delta(\mathbf{q}), \tag{6.12}$$

where the last term is irrelevant as long as the scattering angle θ, and hence the scattering vector $\mathbf{q}$, do not vanish. Clearly, we can also invert Eq. 6.11 with the result

$$\rho g(r) = \frac{1}{(2\pi)^3} \int d\mathbf{q} \big(S(q) - 1 \big) \exp[i\mathbf{q} \cdot \mathbf{r}]., \tag{6.13}$$

which can be used to deduce $g(r)$ from a measurement of $S(q)$. Typical $g(r)$'s for dense fluids are shown in Figs.7.1 and 7.3.

For later reference we define the *potential of mean force* $w(r; \rho, T)$ by

$$w(r_{12}) = -kT \log g(r_{12}) \quad \Longleftrightarrow \quad g(r_{12}; \rho, T) = \exp[-\beta w(r_{12}; \rho, T)] \tag{6.14}$$

This name stems from the fact that the gradient $-\nabla_i w(r_{12})$ is the average force $\mathbf{f}(r_{12})$ acting on particle 1, keeping 1 and 2 fixed and averaging over all the others,

$$\begin{aligned} \nabla_1 w(\mathbf{r}_{12}) &\overset{(6.14)}{=} \frac{-kT}{g(r_{12})} \nabla_1 g(r_{12}) \\ &\overset{(6.3)}{=} \frac{\displaystyle\int d\mathbf{r}_3 \cdots d\mathbf{r}_N \exp[-\beta U(\mathbf{r}^N)] \big(-\nabla_1 U(\mathbf{r}^N) \big)}{\displaystyle\int d\mathbf{r}_3 \cdots d\mathbf{r}_N \exp[-\beta U(\mathbf{r}^N)]} \\ &\equiv \langle \mathbf{f}(r_{12}) \rangle. \end{aligned} \tag{6.15}$$

Chapter 7

Integral equation theories

7.1 Ornstein-Zernike equation

In this chapter, we present a method to calculate the radial distribution function $g(r)$ of a system in which the particles interact with pairwise additive potentials $u(r_{ij})$, i.e., the total potential energy of the system can be written as a sum of pair potentials:

$$U(\mathbf{r}_1, \mathbf{r}_2, \cdots, \mathbf{r}_N) = \frac{1}{2} \sum_{i=1}^{N} \sum_{j \neq i}^{N} u(r_{ij}) \tag{7.1}$$

The radial distribution function $g(r)$ describes the probability that if a particle is located at the origin, another particle of the fluid can be found at distance r. If the radial distribution function $g(r)$ is known, one can calculate all thermodynamic properties of the system. Because of the importance of $g(r)$ in theories of strongly interacting, dense fluids, many approaches have been devised to calculate (approximations to) this function, either analytically or numerically. Methods go by the names of "Kirkwood integral equation", "BBGKY hierarchy", and "Ornstein-Zernike" theory. The latter one will be discussed here. We first define the *total correlation function*

$$h(r_{12}) = g(r_{12}) - 1, \tag{7.2}$$

which is a measure for the "influence" of molecule 1 on molecule 2 a distance r_{12} away. In 1914 Ornstein and Zernike proposed to split this influence into two contributions, a direct part and an indirect part. The direct contribution is *defined* to be given by what is called the *direct correlation function*, denoted $c(r_{12})$. The indirect part is due to the direct influence of molecule 1 on a third molecule, labeled 3, which in turn influences molecule 2, directly and indirectly. Clearly, this indirect effect must be weighted by the density of particle 3, and averaged over all its possible positions. Mathematically this decomposition can be written as

$$h(r_{12}) = c(r_{12}) + \rho \int d\mathbf{r}_3 c(r_{13}) h(r_{32}), \tag{7.3}$$

which is called the Ornstein-Zernike (OZ) equation. It can be viewed as the defining equation for the direct correlation function $c(r)$. One may also argue, however, that we have rewritten a function we wish to calculate, $h(r)$, in terms of another function that we do not know, $c(r)$. In that sense Eq. 7.3 can be viewed as a single equation with two unknowns, which can only be solved if another relation between $c(r)$ and $h(r)$ is given. Such an additional relation is called the *closure*. The power of the decomposition given by Ornstein and Zernike is that approximate closures can be given, that allow for explicit calculation of $c(r)$ and $h(r)$ at a given density and temperature.

Before discussing several examples of such closures, we remark that the OZ equation 7.3 can be rewritten in terms of the Fourier transforms $\hat{h}(q)$ and $\hat{c}(q)$ of $h(r)$ and $c(r)$, respectively, as

$$\hat{h}(q) = \hat{c}(q) + \rho\hat{c}(q)\hat{h}(q), \tag{7.4}$$

where the Fourier transform $\hat{f}(q)$ of a spherical symmetric function $f(r)$ is given by

$$\hat{f}(q) = 4\pi \int_0^\infty drr^2 f(r)\frac{\sin qr}{qr}, \tag{7.5}$$

or inversely by

$$\hat{f}(r) = \frac{1}{2\pi^2}\int_0^\infty dqq^2 \hat{f}(q)\frac{\sin qr}{qr}. \tag{7.6}$$

From Eq. 7.4, we obtain that

$$\hat{c}(q) = \frac{\hat{h}(q)}{1+\rho\hat{h}(q)} \qquad ; \qquad \hat{h}(q) = \frac{\hat{c}(q)}{1-\rho\hat{c}(q)}. \tag{7.7}$$

From Eqs. 6.12 and 7.7 we find that $\hat{c}(q)$ is related to the structure through

$$S(q) = \frac{1}{1-\rho\hat{c}(q)}. \tag{7.8}$$

A very successful and relatively simple closure is named after Percus and Yevick (PY). It consists of the *approximation* that

$$c(r) \overset{PY}{\approx} g(r)\big(1 - \exp[+\beta u(r)]\big). \tag{7.9}$$

Introducing the function $\gamma(r) = h(r) - c(r)$, the PY closure can be rewritten as

$$c(r) = \big(\exp[-\beta u(r)] - 1\big)(\gamma(r) + 1). \tag{7.10}$$

The physical motivation behind the PY closure is as follows. First we note that the direct correlation $c(r)$ can be seen as the difference between the (total) pair correlation $g(r) \equiv \exp[-\beta w(r)]$ with $w(r)$ the potential of mean force and an *indirect* term $g_{\text{indirect}}(r)$. This indirect term, which is *defined* as $g_{\text{indirect}}(r) = g(r) - c(r)$, is now *approximated* as $g_{\text{indirect}}(r) = \exp[-\beta\big(w(r) - u(r)\big)] = g(r)\exp[+\beta u(r)]$, i.e. as the Boltzmann factor of $w(r) - u(r)$. Eq. 7.9 then follows readily. Having in mind that $w(r)$ is the (total) potential of mean force, while $u(r)$ is the pair potential, this approximation indeed captures the idea that the indirect contribution is *not* due to direct pair interactions. The solution of Eq. 7.3 with Eq. 7.9 can be obtained through numerical methods. For the hard-sphere fluid the solution can be obtained analytically. The pair potential of hard spheres with a diameter σ is described as:

$$u(r) = \infty \quad r < \sigma, \qquad u(r) = 0 \quad r \geq \sigma. \tag{7.11}$$

We thus find that

$$g(r) = 0 \quad r < \sigma, \tag{7.12}$$

and from Eq. 7.9 it follows that

$$c(r) = 0 \quad r > \sigma. \tag{7.13}$$

An alternative popular closure is the so-called mean spherical approximation, which is applied to spherical particles interacting through a hard-core potential with hard-core diameter σ

and a tail, which can take different functional forms for $r > \sigma$. Similarly to the PY closure, the MSA is formulated in terms of an 'ansatz' for the direct correlation function

$$c(r) = -\beta u(r) \qquad r > \sigma. \tag{7.14}$$

By analyzing the diagrammatic expansion of $c(r)$ at large distances, it can be shown that the MSA closure is correct in the limit $r \to \infty$. Due to the hard-core, we again find that

$$g(r) = 0 \qquad r < \sigma. \tag{7.15}$$

We also note that the MSA reduces to the PY approximation for a purely hard-sphere potential.

By analyzing the diagrammatic expansion of $c(r)$, other closures can be derived. For instance, the hypernetted-chain approximation reads

$$c(r) = \exp[-\beta u(r) + \gamma(r)] - \gamma(r) - 1, \tag{7.16}$$

and the Roger-Young closure

$$c(r) = \exp[-\beta u(r)]\big(1 + \frac{\exp[f(r)\gamma(r)] - 1}{f(r)}\big) - \gamma(r) - 1, \tag{7.17}$$

with $f(r)$ a 'switching function' that allows one to interpolate continuously between the PY and the HNC closere. One can verify that for $f(r) = 0$ in the limit $r \to 0$, Eq. 7.17 reduces to

$$c(r) = \exp[-\beta u(r)]\big(1 + \gamma(r)\big) - \gamma(r) - 1, \tag{7.18}$$

which is the PY closure 7.10, while by taking $f(r) = 1$ in the limit $r \to \infty$, Eq. 7.17 becomes

$$c(r) = \exp[-\beta u(r) + \gamma(r)] - \gamma(r) - 1, \tag{7.19}$$

which is the HNC closure. Rogers and Young chose the following expression for the 'switching' function

$$f(r) = 1 - \exp[-\alpha r], \tag{7.20}$$

where the parameter α is varied until the virial-compressibility consistency condition is satisfied. The OZ equation 7.3 with one of the closures constitutes two independent equations that can be solved for the two unknown functions $h(r)$ and $c(r)$. In practise this can be done numerically using an iterational procedure. The efficiency of this procedure has been improved over the years using a Picard iteration scheme, the Newton-Raphson technique, or other optimization algortihms. Below, we outline the basic procedure to solve the OZ equation. One first starts with an ansatz of $\gamma(r)$, which yields an estimate of $c(r)$ using one of the closures 7.10, 7.14, 7.16, or 7.17. This function is then Fourier transformed to obtain $c(k)$. Using the Fourier transform of the OZ equation 7.4, one obtains $\hat{\gamma}(k)$ using

$$\gamma(\hat{k}) = \frac{\rho c^2(k)}{1 - \rho c(k)}. \tag{7.21}$$

A second Fourier transform (FT) provides a new $\gamma(r)$, which can be used as a new input for the closure relation. This cycle of steps, $\gamma(r) \overset{closure}{\to} c(r) \overset{FT}{\to} c(k) \overset{OZ}{\to} \gamma(\hat{k}) \overset{FT}{\to} \gamma(r)$ is repeated until the output and input $\gamma(r)$ differ less than a required accuracy. The numerical procedure imposes the adoption of a finite grid of points in r-space and k-space for the different correlation functions, which means that one requires a discrete Fourier and inverse Fourier transform algorithm. The accurateness of the final solution will obviously depend on the number of grid points that is

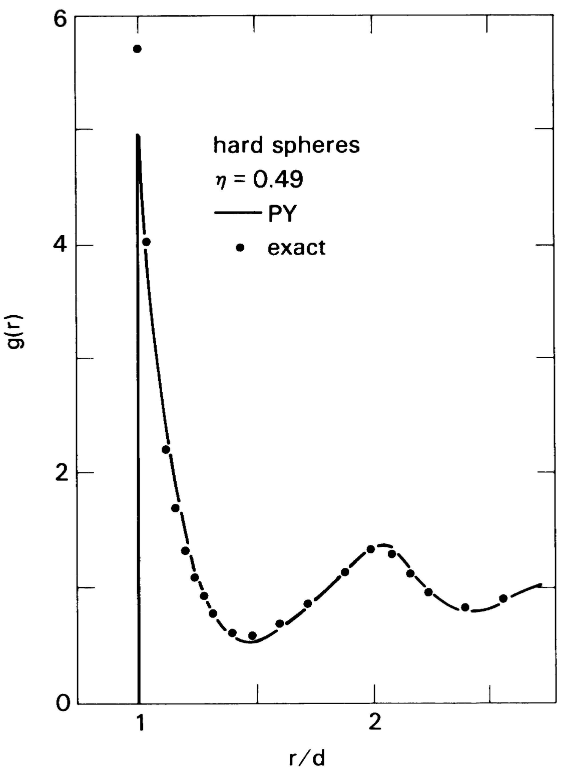


Figure 7.1: Radial distribution function $g(r)$ of a dense hard-sphere fluid.

used, whereas the efficiency of the OZ solver depends on the iteration scheme. The simplest procedure is a Picard iteration scheme, where a new ansatz of $\gamma(r)$ is chosen to be a linear combination of the old and the new solution.

Only for certain potentials and closures, the OZ equation is analytically solvable. For instance, analytic results have been found for the hard-sphere potential using the PY closure to the OZ equation. Independently from each other Wertheim and Thiele showed, in 1963, that the PY closure to the OZ equation of a hard sphere fluid (diameter σ) at the dimensionless density $\eta = (\pi/6)\rho\sigma^3$ (i.e. the packing fraction) yields for the direct correlation function

$$c(r) = \begin{cases} \dfrac{-(1+2\eta)^2 + 6\eta(1+\frac{1}{2}\eta)^2\left(\frac{r}{\sigma}\right) - \frac{1}{2}\eta(1+2\eta)^2\left(\frac{r}{\sigma}\right)^3}{(1-\eta)^4} & r < \sigma \\ 0 & r > \sigma \end{cases} \tag{7.22}$$

This can be analytically Fourier transformed, from which the structure factor follows using Eq. 7.8. Unfortunately $g(r)$ cannot be written down analytically, but a numerical Fourier transform of $S(q)$ is straightforward, and from Eq. 6.11 $g(r)$ follows. The result is in very good agreement with computer simulations of $g(r)$ of hard spheres for $0 < \eta <\simeq 0.5$. This is illustrated in Fig. 7.1. Since the hard-sphere fluid freezes at $\eta \approx 0.494$, the PY closure is accurate in the whole fluid regime of hard spheres. In one of the problems we will calculate that the explicit form Eq. 7.22 for $c(r)$ leads to the pressure p_c via the compressibility route (6.8), and to p_v via

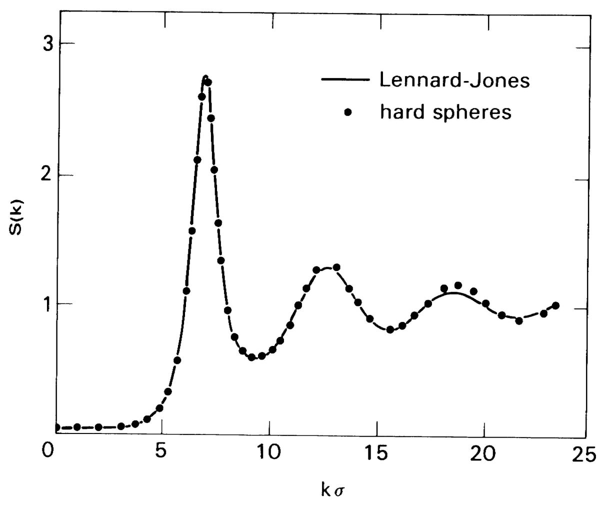


Figure 7.2: Structure factor of a Lennard-Jones fluid close to its triple point ($\rho\sigma^3 = 0.844$, $kT/\epsilon = 0.72$, and that of a hard-sphere fluid close to freezing ($\eta = 0.495$), as obtained from computer simulations.

the virial route (6.6), where

$$\begin{aligned}\frac{p_c}{\rho kT} &= \frac{1+\eta+\eta^2}{(1-\eta)^3}\\ \frac{p_v}{\rho kT} &= \frac{1+2\eta+3\eta^2}{(1-\eta)^2}.\end{aligned} \tag{7.23}$$

The difference between these two expressions increases with increasing η, but both give good account of the pressure that results from simulations. The (slight) inconsistency that results from the different routes is due to the PY approximation; an exact theory would lead to fully consistent thermodynamics. It turns out that the linear combination $p_{CS} = (2p_c + p_v)/3$, which is named after Carnahan and Starling, is indistinguishable from the simulations up to $\eta = 0.5$,

$$\frac{p_{CS}}{\rho kT} = \frac{1+\eta+\eta^2-\eta^3}{(1-\eta)^3}. \tag{7.24}$$

In one of the problems it will be worked out that the Helmholtz free energy, F_{CS}, that follows from p_{CS} reads

$$\frac{F_{CS}}{NkT} = \log\rho\Lambda^3 - 1 + \frac{4\eta-3\eta^2}{(1-\eta)^2}, \tag{7.25}$$

where the first two terms are ideal gas terms, and the last one the excess term due to the hard-sphere interactions.

Because a typical triple point density, ρ_{tr}, of a simple fluid like argon satisfies $\rho_{tr}\sigma^3 \approx 1$, it is interesting to compare the triple-point structure with that of hard spheres at $\rho\sigma^3 \simeq 1$, i.e $\eta \simeq 0.5$. Fig. 7.2 shows such a comparison: the structure of a real dense fluid is qualitatively, and actually almost quantitatively, identical to that of a dense hard-sphere fluid! This implies that the fluid structure is mainly determined by the short-ranged repulsions, while the attractions hardly affect the high-density structure. This is further illustrated in Fig. 7.3, where the liquid structure factor (close to the triple point) of Argon is shown. The similarity with Fig. 7.1 for $g(r)$ of hard spheres is striking.

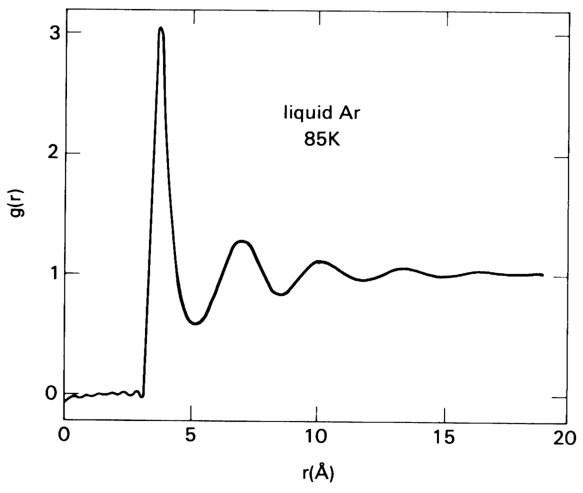


Figure 7.3: Radial distribution function of triple-point liquid Argon as measured by neutron scattering. The ripples at small r are artefacts of the data analysis.

This notion is a crucial ingredient of the perturbation theory which often uses the hard-sphere radial distribution function as input.

Chapter 8

Exercises

Use CodeBlocks to get C working on windows:

- Install CodeBlocks from `www.codeblocks.org`
- Then follow the instruction on `http://wiki.codeblocks.org/index.php/Installing_the_latest_official_version_of_Code::Blocks_on_Windows`.
  Note that these instructions install TDM-GCC and set up CodeBlocks to use it. To run the code, try the following:
- Click on "Create a new project"
- Choose "Empty project"
- Go through Wizzard, and add a Project name , and directory when asked. Unselect Create "Debug" configuration when asked. Make sure the GNU GCC compiler is chosen as the compiler. Then hit finish.
- To compile, hit "build" (the little gear icon).
- To run, hit run (the little green arrow).

Alternatively, one can use PuTTY to compile and run your C programs or one can also install gnu compilers on windows

1) MingW tool :

Download page: `https://sourceforge.net/projects/mingw-w64/`

Instructions: `https://www.ics.uci.edu/~pattis/common/handouts/mingweclipse/mingw.html`

2) Cygwin tool:

Download page: `https://cygwin.com/install.html`

Instructions: `https://preshing.com/20141108/how-to-install-the-latest-gcc-on-windows/`

Exercise 1: Programming language

Note: If you have never written a program before it may be a good idea to look closely at the example programs provided in Chapter 11.

1. Read the manual on C.
2. Write a "hello world" C program that prints some text on the screen. Compile your program with the following command:
   `gcc -Wall -O3 hello_world.c -o hello_world`.
   Run you program by typing `./hello_world` in the directory where it resides. The flag `-Wall` is to enable compile warnings. The flag `-O3` is to optimize the machine code.
3. Write a C program that prints out the first 1000 prime numbers. Check your program with Mathematica using `Prime[]`.

Exercise 2: Estimating Π

A simple exercise that is often used to showcase the strength of the Monte Carlo method, is estimating π. In this exercise we will write a C program for estimating π to at least two decimal places using the *Hit and miss* method as explained in the lecture. To this end, we take a circle of unit radius that is centered at the origin and circumscribed in a square of side length 2×2. We generate a number of trial shots, say N, inside the square by selecting two random numbers from a uniform distribution in $[-1, 1]$, which then form the coordinates in the x- and y-direction. Subsequently, for each trial, we determine if the coordinate falls inside the circle, which is then counted as a "hit". The ratio of the number of "hits" and the number of trials can be used to estimate π since $N_{hits}/N \simeq \pi/4$. Use the provided pseudo-random number generator (PRNG) by including the header `"mt19937.h"`. A simple example of how to use the random number generator is given below.

```
#include <stdio.h>
#include <time.h>
#include "mt19937.h"

int main(int argc, char* argv[]){
    dsfmt_seed(time(NULL));
    printf("random number %lf \n", dsfmt_genrand());

    return 0;
}
```

Note how we first seed the PRNG using the current time. Internally, a PRNG contains a state that is initialized using the provided seed. It always produces the same sequence of numbers for that seed.

1. Write a C program using the direct sampling method. Run the program twenty times for number of trials $N = 10, 100, 1000, 10^4$. Estimate the mean square deviation $\langle (N_{hits}/N - \langle N_{hits}/N \rangle)^2 \rangle$ and plot it as a function of N. How does the mean square deviation scale with N? **[2 points]**

2. Write a C program using the Markov Chain method. In the Markov Chain method, we start at a random position inside the square. Subsequently, we select two random numbers Δx and Δy from a uniform distribution in $[-\delta, \delta]$ that corresponds to a random displacement in the x- and y-direction. The new position becomes $[x + \Delta x, y + \Delta y]$. In the case the new position falls outside the square, we reject this trial move and keep the old position. We repeat this procedure until we have generated N trial moves. Run the program twenty times for a large number of trial attempts N and varying maximum displacements $\delta \in [0L, 3L]$ with L the side length of the square. Plot the mean square deviation $\langle (N_{hits}/N - \langle N_{hits}/N \rangle)^2 \rangle$ and the rejection rate as a function of δ. Which value of the rejection rate yields the highest precision? **[3 points]**

3. Write a C program using the Markov Chain method for a discrete lattice with $M \times M$ lattice sites using a neighbour table. Check that, for long runs, all sites are visited with equal probability. **[3 points]**

4. Determine all eigenvalues and eigenvectors of the transfer matrix of a 3×3 lattice (using a linear algebra routine in Mathematica). Compute the eigenvalues of a $M \times M$ lattice and plot the correlation time τ as a function of lattice size M. **[2 points]**

Exercise 3: Random Walk

Random walk models are found ubiquitously in physics, from Browian motion, to a simple description of a polymer, and even to quantum field theory. In this exercise we explore some of the properties of random walks. A simple random walk consists of consecutive steps with fixed step length and random directions. The path is also called a "drunk man's" walk.

1. Write a C program for a simple random walk in 2 dimensions. **[2 points]**

2. Modify your code so that it writes an output file "xy.dat" that contains on line N the corresponding x and y coordinates of the Nth step of the random walk. For instance,

   ```
   0.00    0.00
   0.97    0.24
   ...     ...
   ```

   You can use the `fopen` and `fclose` routines for opening and closing a file, and `fprintf` for printing to it. To read in the data file "xy.dat" in your Mathematica notebook you can use the following Mathematica code:

   ```
   list = Import["G:\xy.dat"];
   ```

   Plot the result with:

   ```
   ListPlot[list,Joined->True]
   ```

   [1 points]

3. Change your C code such that it also calculates $r^2(N)$ of the random walk, where $r(N)$ is the distance from the starting point after N steps. We now wish to calculate $\langle r^2(N)\rangle$, where $\langle\cdots\rangle$ means an average over many different and independent random walks. What is the functional dependence of $\langle r^2(N)\rangle$ on the number of steps N? **[2 points]**

4. Use now periodic boundary conditions in your C code. Periodic boundary conditions are often applied in simulations to avoid surface effects. In this case, the random walk is confined in a central box, which we assume to be square. The central box is replicated in all directions to form an infinite lattice. If the random walk leaves the central box during a simulation at one side, the random walk enters the central box again at the opposite side. Calculate again $\langle r^2(N)\rangle$ and compare the results with the previous ones. **[2 points]**

5. Change your C code now to three dimensions. Make sure that the code generates a random unit vector that is uniformly distributed on the surface of a unit sphere. Why is it not possible to generate a random unit vector by generating a random angle $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi]$? Does the functional dependence of $\langle r^2(N)\rangle$ on N change? **[3 points]**

Hand-in your .c files and a Mathematica notebook with the graphs showing $\langle r^2(N)\rangle$ for the different cases in exercise 2.2. Also, make sure you provide any files that may be needed by the Mathematica notebook. You can set the directory Mathematica looks for files by using the command `SetDirectory[NotebookDirectory[]]`. This way you can import the data files using a path relative to the directory in which the Mathematica notebook is located.

Exercise 4: Monte Carlo simulation of Hard spheres in the NVT ensemble

In order to be able to start a Monte Carlo simulation, we first need a good starting configuration. We will write a C code that creates a starting configuration of non-overlapping hard spheres that can be used later in a Monte Carlo simulation.
We will use https://webspace.science.uu.nl/~herme107/viscol/ to plot the configurations.

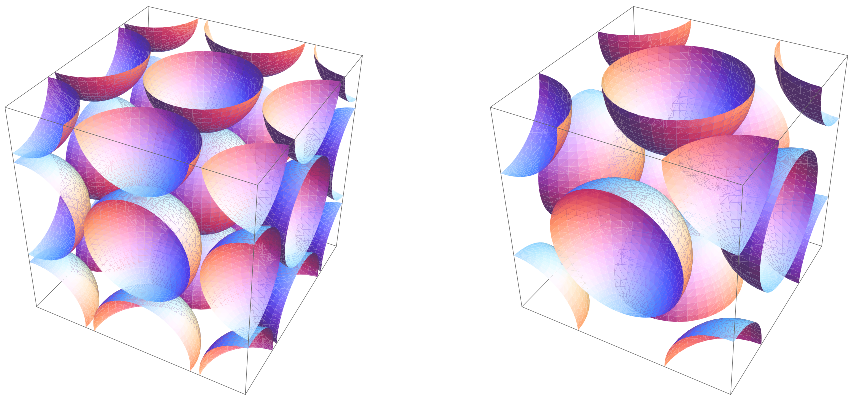


Figure 8.1: A cubic and an fcc crystal lattice of hard spheres

1. Write a C program that places $N = N_x \times N_y \times N_z$ hard spheres with diameter $d = 1$ on a cubic lattice with variable lattice spacing l, in a box with periodic boundary conditions. See Fig. 8.1 and Wikipedia for a description of a cubic crystal structure: `http://en.wikipedia.org/wiki/Face-centered_cubic`. We use the diameter of the spheres as our unit of length. Write the x-, y, and z-coordinates to an output file "xyz.dat" with the top line the number of particles, and the second line the dimensions of your box, and then on each following line, the x-, y-, and z-coordinate as well as the diameter of each disk, e.g.

```
 1145
-4.754203 4.754203
-4.754203 4.754203
-4.754203 4.754203
-2.533263 2.787014 1.781742 0.9609611
4.005931 -0.133235 4.296286 1.063887
  ...                  ...
```

   [0.5 points]

2. What is the maximum volume fraction you can obtain with a cubic lattice without creating overlaps? **[0.5 points]**

3. The stable crystal structure of hard spheres is a face-centered cubic (fcc) lattice. See Fig. 8.1 and Wikipedia for a description of a fcc crystal structure: `http://en.wikipedia.org/wiki/Face-centered_cubic`. Write a C program that places N hard spheres with diameter $d = 1$ on an fcc lattice with variable lattice spacing l, in a box with periodic boundary conditions, and write your results to an output file "xyz.dat" with the same format as previously. **[0.5 points]**

4. What is the maximum volume fraction you can obtain with an fcc lattice without creating overlaps and compare this number with that for a cubic lattice? **[0.5 points]**

We will now use the starting configurations that we have created in the previous part of the exercise in a Monte Carlo simulation. If you did not manage to complete the exercise, you can use the configurations that are provided. It is convenient to use a vector notation for the coordinates of the particles so that one can easily switch from two dimensions (2d) to three dimensions (3d) by simply changing `NDIM` from 2 to 3. In the code that is provided for this week's exercise, we define the dimension using the following directive:

```
#define NDIM 3
```

We also define the maximum number of particles:

```
#define N 100
```

The particle positions are stored in the array `r[N][NDIM]`, while the box dimensions, in the array `box[NDIM]`. Note that we need to pick `N` large enough to make sure the array is large enough to accommodate the particles we wish to simulate. A few variables are defined at the beginning of the code, including the number of Monte Carlo steps to perform, `mc_steps`. Before you proceed, take a look at the provided code and make sure you understand what it does and how it works.

5. Fill in the code for the `read_data()` subroutine. This should read the data provided by the file whose path is given by the variable `init_filename`. More specifically, the array `r[N][NDIM]` should be filled with the particle coordinates, `box[NDIM]`, with the box dimensions, and the variable `n_particles` should hold the number of particles in the simulation. Make sure your function works by either printing the variables, or comparing to one of the output configurations. A function which might come in handy is `fscanf`, which works similarly to `fprintf`, but instead reads variables based on the provided format. Also, make sure you understand the concept of pointers, and how to pass variables by address. **[1.5 points]**

6. Fill in the code for the `move_particle()` subroutine. This should attempt to displace a random particle by a random amount in the range [-`delta`, `delta`] in each direction. If any overlaps occur, the subroutine should return 0, indicating that the move was rejected. In the opposite case, the particle position is updated and the subroutine returns 1, indicating the move was accepted. For this part, we will not bother with checking for overlaps, the function should always return a 1 indicating it is always accepted. Instead, make sure the particles always stay in the box by applying periodic boundary conditions to their new position. In order to pick a particle at random, you should use the random number generator `dsfmt_genrand()`, and you should truncate the result to an integer, by using a cast `(int)()`. The displacement is homogeneously and randomly distributed between -`delta` and `delta` for all directions. What is a reasonable value for `delta`?. **[1.5 points]**

7. Now, complete the implementation of the `move_particle()` subroutine by writing code for checking if a particle is overlapping with any other particles in its new position. Calculate the distance between the randomly chosen and displaced particle and all the other particles in the simulation box and check if the center-of-mass distance is less than 1 particle diameter. If there are distances less than one particle diameter than there is a particle overlap and the move should be rejected (return 0). Stop checking if you find an overlap. Make sure to take care of the periodic boundary conditions when you calculate distances

(use nearest image convention). If the displacement of a randomly picked particle does not result in an overlap, accept the displacement and update the new coordinates of this particle. **[2 points]**

8. You should now have your first working NVT Monte Carlo simulation of hard spheres. Use the Webgl code `http://www.staff.science.uu.nl/~herme107/viscol/` to visualize the configurations. **[1 points]**

9. Make a rough estimate of the packing fraction at which the fcc crystal phase of hard spheres melts in three dimensions by visualizing the configurations. You can run simulations at different packing fractions by modifying the variable `packing_fraction` to the desired value. **[2 points]**

Hand in typical configurations of the 3d system of hard spheres and the answers to the questions in this exercise. Also hand in the code you wrote for the `move_particle()` and `read_data()` subroutines.

Exercise 5: MC simulation of hard spheres in the NPT ensemble

We will now modify our Monte Carlo simulation code written in the previous exercise from the NVT to the NPT ensemble. In order to fix the pressure in a simulation, one has to allow for fluctuations in the volume of the simulation box. To this end, we have to perform trial moves that consist of an attempt to change the volume. In this exercise, we will implement the volume moves in 3d, but it is straightforward to perform the volume moves in 2d. Before you proceed, take a look at the provided code and make sure you understand what it does and how it works.

1. Fill in the code for the `change_volume()` subroutine in the provided code. This function will perform a trial move to change the volume of the simulation box. To this end, we attempt to change the volume of the box by ΔV, where ΔV is a random number uniformly distributed in the interval $[-\Delta V_{\max}, \Delta V_{\max}]$. Calculate the corresponding box length, and scale the particle coordinates with the ratio of the new box length and the old box length. Check if the new configuration contains particle overlaps. Reject the volume move if there are any overlaps. If the new configuration contains no overlaps, one will accept the new configuration according to the acceptance rule derived in the lectures:

   $$\mathrm{acc}(o \to n) \;=\; \min\left(1, \exp[-\beta(U(\mathbf{s}^N; V') - U(\mathbf{s}^N; V) + P(V' - V) - N\beta^{-1}\ln(V'/V))]\right)$$

   See the lecture notes for more details and an example of a pseudo-code. As in the previous exercise in the `move_particle()` subroutine, you should return 1 to indicate that the move was accepted, and 0 otherwise. **[4 points]**

2. Measure the average volume in a Monte Carlo simulation of hard spheres for varying pressures. We first calculate the volume as a function of pressure for the crystal phase. We start with an fcc crystal phase as an initial configuration and we fix the pressure at a very high value, e.g. $\beta P\sigma^3 = 50$ with $\beta = 1/k_B T$ and σ the diameter of the spheres. We then measure the average volume. Subsequently, one can decrease the pressure step by step and measure the volume as a function of pressure. In addition, we also measure the volume as a function of the pressure for the fluid phase. We now use a fluid configuration as the initial configuration of the simulation. We measure now the volume by increasing the pressure step by step. A fluid configuration can be generated by performing a simulation at very low pressure. Plot the dimensionless pressure $\beta P\sigma^3$ vs the packing fraction, η for both cases. **[4 points]**

3. Compare your results for the fluid phase with the Carnahan and Starling equation of state:

   $$\frac{p_{\mathrm{cs}}}{\rho kT} \;=\; \frac{1 + \eta + \eta^2 - \eta^3}{(1-\eta)^3} \tag{8.1}$$

   Plot a figure showing the packing fraction, η as a function of the dimensionless pressure, $\beta P\sigma^3$, for the fluid and crystal phase in comparison with the Carnahan-Starling equation of state. **[2 points]**

Hand the plots as well as the code you wrote for the `change_volume()` subroutine.

Exercise 6: Pair correlation function using MC simulations in the NVT ensemble

To compare with experiments, we calculate the pair correlation function in our simulations. Correlations can be measured experimentally by neutron, x-ray, or light scattering in Fourier space or by confocal microscopy in real space. The pair correlation function (or radial distribution function) for a homogeneous system with N particles is given by:

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

Here δ is the Dirac delta function and $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$ is the displacement vector between particle i and particle j. We note that the radial distribution function always goes to one for large distances and low densities.

1. Can you explain why the radial distribution function always goes to one for large distances? **[2 points]**

2. Calculate the radial distribution function from a histogram of all particle-particle distances, where a particular bin b of the histogram corresponds to the interval $(r, r + \delta r)$. We calculate the histogram by going over all particle pairs and incrementing the appropriate bin corresponding to the center-to-center distance of the particle pair. In this way, we can measure $n_{his}(b)$, which is the number of particle pairs that have a particle-particle distance in the interval $(r, r + \delta r)$. Subsequently, we divide $n_{his}(b)$ by the number of particles N and by the average number of particles in the same interval in an ideal gas at the same density $\rho = N/V$: $n_{id}(b) = (4\pi\rho)[(r + dr)^3 - r^3]/3$. The radial distribution function reads:

$$g(r + 0.5\delta r) = \frac{1}{N}\frac{n_{his}(b)}{n_{id}(b)}. \tag{8.2}$$

   Calculate the radial distribution function for varying densities in the fluid phase using the provided NVT code, and plot them. **[6 points]**

3. Explain the behaviour of the radial distribution function as a function of density. **[2 points]**

Hand in the plots and answers to the questions.

Exercise 7: Ornstein-Zernike solver

In this exercise you will write and run a program that solves the Ornstein-Zernike equation with the Percus-Yevick (PY) closure. Recall that the PY closure reads

$$c(r) = \big(\exp[-\beta u(r)] - 1\big)(\gamma(r) + 1), \tag{8.3}$$

where $\gamma(r) = h(r) - c(r)$. Note that $h(r)$ and $c(r)$ can be discontinuous at a discontinuity in the pair potential, $\gamma(r)$ will be continuous. You can see this from the Ornstein-Zernike equation in terms of $\gamma(r)$

$$\gamma(r) = \rho \int c(r)h(r) \tag{8.4}$$

When the OZ equation is Fourier transformed, and the result is expressed in terms of $\hat{\gamma}(q)$ it is easy to show that:

$$\hat{\gamma}(q) = \frac{\rho c^2(q)}{1 - \rho c(q)}. \tag{8.5}$$

For spherically symmetric functions, the Fourier transform of $f(r)$ is given by

$$\hat{f}(q) = \frac{4\pi}{q} \int_0^\infty drrf(r)\sin(qr). \tag{8.6}$$

If we approximate the integral in Eq. 8.6 by a sum, i.e., by using the trapezoid rule, it follows that

$$\hat{f}(q) = \frac{4\pi dr}{q} \sum rf(r)\sin(qr). \tag{8.7}$$

where dr and dq are the spacings of the grid points in r- and q-space, respectively. The sum in Eq. 8.6 can be performed efficiently by using a fast-Fourier-transform (FFT) $\sin$ routine (see source code). The discretization in this $\sin$ FFT routine (from Numerical Recipes) is based on $drdq = \pi/N$ and requires that the number of grid points L is a power of 2. The inverse Fourier transform has a similar form, specifically,

$$f(r) = \frac{1}{2\pi^2 r} \int_0^\infty dqqf\hat{(q)}\sin(qr). \tag{8.8}$$

or on a grid as

$$f(r) = \frac{dq}{2\pi^2 r} \sum qf\hat{(q)}\sin(qr). \tag{8.9}$$

Since discontinuities cause problems for discrete Fourier transform methods, one has to avoid them as much as possible; in particular, if there is a discontinuity in the potential, one should choose the grid in such a way that the discontinuity is at one of the grid points exactly. We now turn to the numerical solution of the OZ equation with the PY closure. We will solve the equations iteratively (this is known as the Picard method). Proceed as follows:

I Choose some initial $\gamma(r)$ (e.g., $\gamma(r) = 0$ for $0 \leq r \leq L$).

II Use the closure, Eq. 8.3, to calculate $c(r)$.

III Use the FFT routine to compute $\hat{c}(q)$ on a grid.

IV Use the OZ equation (8.5) to compute a new value of $\hat{\gamma}(q)$.

V Again use the FFT to invert $\hat{\gamma}(q)$ and to obtain $\gamma_{new}(r)$ on a grid.

VI Compare $\gamma_{new}(r)$ and $\gamma_{old}(r)$ at each grid point. If $\sum |\gamma_{new}(r) - \gamma_{old}(r)| \leq TOL$ you have found a solution. Typically TOL can be 10^{-4}. If not, you must replace $\gamma_{old}(r)$ by $\gamma_{new}(r)$ and go back to step 2. In actual practice, it turns out that this simple iteration does not converge very rapidly and can have oscillations. You can improve the convergence rate in several ways: 1) Introduce Broyles mixing; specifically, before repeating the iteration, let $\gamma_{old}(r) = \alpha\gamma_{new}(r) + (1-\alpha)\gamma_{old}(r)$ where the mixing parameter α is around 0.5. 2) You can slowly change one of the physical parameters, e.g., density or temperature. For example, if you want a high density solution, you can first obtain a low density one, and use it as the initial guess for the high density one. Clearly, you may have to do this in several stages.

1. Use your program to obtain and plot 1) the pair correlation function $g(r)$, 2) structure factor $S(q)$, and 3) direct correlation function $c(r)$ for a fluid of hard spheres with diameter σ and packing fraction, $\eta = \pi\sigma^3 N/6V =$ 0.2, 0.3, and 0.4. Use an FFT grid containing 2048 points, and choose the upper cutoff in the r integrals to correspond to about 10σ. **[6 points]**

2. Compare the direct correlation function with the analytic expression 7.22 and make a plot. **[2 points]**

3. Compare the pair correlation functions $g(r)$ with those obtained using your MC simulation results and make a plot. **[2 points]**

Exercise 8: MC simulation of a Lennard-Jones fluid in the NVT ensemble

The aim of this exercise is to calculate the equation of state and some thermodynamic variables such as the pressure and the energy of a Lennard-Jones system. We will employ the "truncated and shifted" Lennard Jones potential between particle i and j:

$$u(r_{ij}) = \begin{cases} 4\epsilon \left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6} \right] - e_{\text{cut}} & r_{ij} \leq r_{\text{cut}} \\ 0 & r_{ij} > r_{\text{cut}} \end{cases}$$

where

$$e_{\text{cut}} = 4\epsilon \left[\left(\frac{\sigma}{r_{\text{cut}}}\right)^{12} - \left(\frac{\sigma}{r_{\text{cut}}}\right)^{6} \right] \tag{8.10}$$

and r_{cut} is the cutoff radius. It is convenient to chose σ as the unit of length, ϵ as the unit of energy, and m as the unit of mass.

For this assignment an NVT-Monte Carlo program for the Lennard Jones potential will be provided. The code is a modification of the NVT-Monte Carlo program for hard spheres that was provided earlier. The code assumes that the box length of the simulation box is at least 2 times r_{cut}, where we have fixed $r_{\text{cut}} = 2.5\sigma$. The potential is calculated using the nearest image convention which is only valid provided the cutoff radius is less than half the box length. To set the density of the system, change `density` in the C code to the desired value. The code will change the box length upon initialization accordingly. In order to determine whether or not to accept an attempt to displace a particle, one has to calculate the potential energy. The subroutine `particle_energy_and_virial()` calculates the potential energy contribution u_i due to the interactions of particle i with all the other particles:

$$u_i = \sum_{j=1;j\neq i}^{N} u(r_{ij}) \tag{8.11}$$

The total energy U_{tot} is given by:

$$U_{\text{tot}} = \frac{1}{2} \sum_{i=1}^{N} u_i \tag{8.12}$$

In addition, the subroutine `particle_energy_and_virial()` also calculates the contribution of particle 'i' to the virial term which is required for calculating the average pressure. This subroutine is called from the `move_particle()` subroutine, to determine the energy change from moving a random particle. The move is accepted using the Metropolis criterion.

1. We calculate first the average pressure using the virial equation for the pressure:

$$\langle P \rangle = \rho k_B T + \frac{1}{3V} \left\langle \sum_{i<j} \mathbf{f}(\mathbf{r_{ij}}) \cdot \mathbf{r_{ij}} \right\rangle \tag{8.13}$$

where the virial $\left\langle \sum_{i<j} \mathbf{f}(\mathbf{r_{ij}}) \cdot \mathbf{r_{ij}} \right\rangle$ can be written in terms of the intermolecular force $\mathbf{f}(\mathbf{r}) = -\nabla u(r)$. Express the virial equation for the pressure in terms of the interparticle distances r_{ij}. Show that the result corresponds to the virial calculation in the subroutine `particle_energy_and_virial()`. Modify the subroutine `measure()` to calculate the average pressure using the virial equation. **[1.5 points]**

2. We now modify the C code to calculate the excess chemical potential using the Widom test particle method as described in the course manual. Implement the subroutine `measure()` such that it makes an attempt to add a test particle into your system. Calculate the change in potential energy ΔU_i^+ and the corresponding Boltzmann factor $\exp[-\beta \Delta U_i^+]$ for the ith test particle attempt. In order to obtain good statistics, you have to average over many attempts, say N_{test}, to insert a particle. The excess chemical potential μ_{ex} is given by

$$\mu_{\text{ex}} = -k_B T \ln \left(\sum_{i=1}^{N_{\text{test}}} \frac{\exp[-\beta \Delta U_i^+]}{N_{\text{test}}} \right). \tag{8.14}$$

   Note that

```
particle_info_t info = particle_energy_and_virial(npart+1)
```

   calculates the energy change (`info.energy`) and the change in the virial term (`info.virial`) when particle `npart+1` is added. Also, keep in mind that the subroutine `measure()` returns a structure of type `measurement_t` which has two members, `average_pressure` and `mu_excess`. You should set the value of the appropriate members. These will in turn get written to a file called `measurement.dat`, where the first column is the Monte Carlo cycle, the second is the pressure, and the third column is the chemical potential. **[1.5 points]**

3. Can we also use the virial expression for the pressure and the Widom test particle method for calculating the pressure and chemical potential in the case of hard spheres? Explain in a few sentences. **[1 points]**

4. Plot the pressure as a function of the density $\rho\sigma^3 \in [0, 1.1]$ and at $T^* = k_B T/\epsilon = 2.0, 1, 0,$ and 0.5. Explain the results. **[2.5 points]**

5. In which regime is the ideal gas law a good approximation? **[1 points]**

6. Plot also the chemical potential $\mu = \mu_{\text{id}} + \mu_{\text{ex}}$ as a function of the density $\rho\sigma^3 \in [0, 1.1]$ and at $T^* = k_B T/\epsilon = 2.0, 1, 0,$ and 0.5. Here, μ_{id} denotes the chemical potential of an ideal gas at the same density. Explain the results. **[2.5 points]**

Please, hand in all your results and answers to the questions, as well as the subroutine `measure()`.

Exercise 9: Grand-canonical Monte Carlo simulations

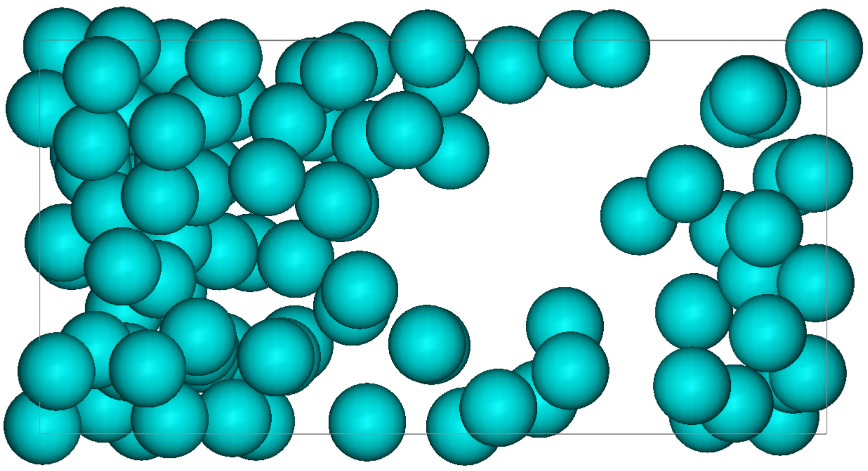


Figure 8.2: Gas-liquid coexistence in a rectangular box

In this exercise, we perform a grand-canonical Monte Carlo simulation (GCMC) of a Lennard Jones system close to its critical point. The goal is to map out the gas-liquid phase diagram of such system. In the grand-canonical ensemble, the chemical potential (μ), the volume (V), and temperature (T) are fixed. The number of particles (N) fluctuates. The thermodynamic conditions for phase equilibria are equal chemical potential, equal pressure, and equal temperature. Once that the simulation box volume and the temperature are fixed, we can tune the chemical potential to identify the value at bulk coexistence, yielding equal probability to observe both phases.

In order to perform GCMC simulations, you need to adapt the code from the previous assignment. In each MC cycle we make N attempts to translate a random particle, and 1 attempt to randomly either insert a new particle or remove a random particle. You can convert the subroutine written for the implementation of the Widom test particle method into a function that takes care of particle insertion/deletion. The acceptance rule for inserting a particle is:

$$\text{acc}(N \rightarrow N+1) = \min\left[1, \frac{V}{\Lambda^3(N+1)} \exp[\beta(\mu - U(N+1) + U(N))]\right]$$

The acceptance rule for removing a particle is:

$$\text{acc}(N \rightarrow N-1) = \min\left[1, \frac{\Lambda^3 N}{V} \exp[-\beta(\mu + U(N-1) - U(N))]\right]$$

where Λ is the thermal wavelength that is set equal to σ. Finally, you need a subroutine to calculate a histogram $h(N)$ of the number of particles (N) in the simulation. You can now start studying the phase diagram of the system.

1. Fix the temperature $T^* \equiv k_B T/\epsilon = 2.0$ and measure the histogram of the number of particles $h(N)$ for several values of the chemical potential $\mu^* \equiv \mu/\epsilon$. Plot the obtained histograms $h(N)$ and describe how they change upon varying μ^*. **[3 points]**

2. What is a reasonable range of values for μ^* in order to have an average density $\rho = N\sigma^3/V$ in between 0 and 0.9? Check if your answer is consistent with the results obtained from the Widom test particle method. **[2 point]**

3. Determine the Helmholtz free energy per volume F/Nk_BT as a function of density $\rho\sigma^3$ by using a thermodynamic integration of the equation of state, pressure as a function of density, as obtained in the previous exercise at temperature $T^* = k_BT/\epsilon = 2.0, 1, 0$, and 0.5, and plot the result along with the chemical potential μ^* as a function of density $\rho\sigma^3$. **[2 point]**

4. Determine gas-liquid coexistence, either by plotting the pressure versus chemical potential or by using a common tangent construction, for temperature $T^* = k_BT/\epsilon = 2.0, 1, 0$, and 0.5. **[3 points]**

Exercise 10: Sudoku solver with simulated annealing

8					1	2		
	7	5						
				5			6	4
		7						6
9			7					
5	2				9		4	7
2	3	1						
		6		2		1		9

Sudoku puzzles can be solved using pen and paper, but it can also be solved by writing a computer algorithm that finds the solution via simulated annealing. Simulating annealing is a method to find the optimal solution to a problem in a large search space. The name annealing comes from metallurgy where the number of defects in a crystalline material is reduced by controlled heating and cooling of the sample. Heating will cause the atoms to wander randomly through states of high energy, whereas the cooling gives them a chance of finding configurations with a lower internal energy. By analogy, each step of the simulating annealing algorithm attempts to replace the current solution by a random solution. The new solution may then be accepted with a probability that depends on the ″energy″ difference and on the ″temperature″ T, that is gradually decreased during the process. The dependency is such that the choice between the previous and current solution is almost random when T is large, but increasingly selects the better or ″downhill″ solution (for a minimization problem) as T goes to zero. The allowance for ″uphill″ moves potentially saves the method from becoming stuck at local optima. The simulated annealing method is thus based on an ordinary Monte Carlo (MC) simulation in the canonical ensemble, where the temperature is slowly decreased. The algorithm to solve a Sudoku puzzle starts by inserting random numbers between 1 to 9 into the vacant slots of the puzzle. One then calculates the ″energy″ of the configuration, which determines how ″good″ or ″bad the solution is. A high energy corresponds to a very bad solution with many faults, and a low energy corresponds to a good solution. The energy function can be for instance the number of duplicates in all the blocks, columns, and rows. Subsequently, one can generate a new solution by changing one of the numbers in the vacant slots of the puzzle and calculate the ″energy″ for this new configuration. The new solution is then accepted according to the acceptance rules of an ordinary NVT-MC simulation.
The grading will be as follows:
[6 points] Writing simulated annealing code to solve sudoku puzzles
[2 points] Investigation of different parameters and levels of sudoku puzzles
[1 point] Plot the energy/temperature as a function of time
[1 point] Discuss different optimalisation strategies or alternative ways to solve sudoku puzzles, how the time to solve a sudoku puzzle depends on the level of the sudoku puzzle (easy or difficulte ones), some literature search, etc.

Exercise 11: Sedimentation of Lennard Jones Particles

In this exercise we examine the behaviour of Lennard Jones particles in a gravitational field. Due to the competition between the thermal energy and the gravitational potential energy, suspensions of colloidal particles are spatially inhomogeneous, i.e. the density is a function of the height of the system $\rho(z)$, where z is the vertical axis. From this density distribution it is possible to extract various physical quantities such as the equation of state and Boltzmann's constant.

1. To study systems in a gravitational field we need to establish the direction of the field. We shall take the positive z direction to be "up". We now ignore the periodic boundary conditions in the z direction. As a first step, include a hard wall at the bottom of the simulation box by rejecting particle moves which displace the particles out of the box. Additionally, there is no need to have a "ceiling" in the simulation box - i.e. the particles should be trapped in the z direction such that $0 \leq z < \infty$. Make all required changes to the periodic boundary conditions to allow for this. **[1 points]**

2. Next, add a gravitational field. The gravitational potential energy is given by

$$U_g(\mathbf{r}) = mgr_z \tag{8.15}$$

   where m is the particle mass, g is the gravitational constant and r_z is the z component of the coordinate of the particle $\mathbf{r}$. **[1 points]**

3. Write a routine that initializes $Npart$ particles randomly in your simulation box. **[1 points]**

4. Write a routine to calculate the number density as a function of the height of the simulation box ($\rho(z)$). To this end, divide the $z-$direction of the simulation box up into sections of size dz and count how many particles are in each bin. (This is very similar to the calculation in Exercise 3 for the radial distribution function.) Determine what the proper normalization constant is, and normalize the result. Hint: Calculate $\int_0^\infty \rho(z)\, dz$. **[1 points]**

5. Test your program by running it for various choices of the gravitational constant (mg = 1, 2.5, 5). Which phases do you see? Can you see the interface between various phases in the density profiles? Explain. **[2 points]**

6. Show that

$$P(z) = mg \int_z^\infty \rho(z')dz' \tag{8.16}$$

   [1 points]

7. For β = 0.5 and $mg = 1$ use the equation you just derived to calculate the equation of state. Is there a gas-liquid coexistence at $\beta = 0.5$? Compare the result to the equation of state you calculated using the virial expression for the pressure. (To obtain good results you will probably need to use rather large system sizes, such as $Npart = 500$, and rather long simulation runs, such as 50.000 initialization cycles and 100.000 sampling cycles.) **[3 points]**

Chapter 9

Appendix: Essential Mathematics

Elementary Statistical Thermodynamics does not make use of any sophisticated mathematics. Here we briefly review the mathematics that is most frequently used. Below, we give neither a proper derivation nor a proof of any of the results that we quote. However, in some cases we do provide a non-rigorous ”justification”. We assume that the reader is familiar with the most common functions and algebraic manipulations.

9.1 Properties of $\ln x$ and $\exp x$

The essential properties of logarithms and exponentials are - of course - well known $\cdots$ but still often forgotten.

$$\begin{aligned}
\ln(a \times b) &= \ln a + \ln b && (9.1)\\
\ln(a/b) &= \ln a - \ln b && (9.2)\\
\ln(a^b) &= b \times \ln a && (9.3)\\
{}^g\log(a) &= \ln(a)/\ln(g) && (9.4)\\
\exp(a+b) &= (\exp a) \times (\exp b) && (9.5)
\end{aligned}$$

9.2 Chain Rule

When differentiating a function $F(u)$, where $u(x)$ is a function of the independent variable x, we can use the so-called chain rule

$$\frac{\partial F(u(x))}{\partial x} = \frac{\partial F(u)}{\partial u} \times \frac{\partial u(x)}{\partial x} \tag{9.6}$$

More generally, if $F(u)$ is a function of u and u is a function of $v \cdots$ and y is a function of z, then

$$\frac{\partial F}{\partial z} = \frac{\partial F(u)}{\partial u} \times \frac{\partial u(v)}{\partial v} \times \frac{\partial v(w)}{\partial w} \times \cdots \times \frac{\partial y(z)}{\partial z}$$

9.3 Derivative of $\exp(ax)$ and $\ln(x)$

The derivative of $\exp(ax)$:

$$\frac{\partial \exp(ax)}{\partial x} = a \exp(ax) \tag{9.7}$$

This result can easily be derived from the definition of $\exp(ax)$:

$$\exp(ax) = \lim_{n\to\infty}(1 + \frac{ax}{n})^n \tag{9.8}$$

Conversely, the primitive function of $\exp(ax)$ is $a^{-1}\exp(ax)$.
The derivative of $\ln x$ with respect to x is

$$\frac{\partial \ln x}{\partial x} = \frac{1}{x} \tag{9.9}$$

This is easily derived from Eq. 9.7. If $y = \ln x$, then $x = \exp(y)$, hence

$$\frac{\partial \ln x}{\partial x} = \frac{\partial y}{\partial \exp y} = \frac{1}{\exp y} = \frac{1}{x} \tag{9.10}$$

Conversely, the primitive function of $1/x$ is $\ln x$.

9.4 Taylor Expansion

If $f(x)$ and all its derivatives are smooth functions of x, then we can write:

$$f(x+a) = f(x) + \left(\frac{\partial f}{\partial x}\right)_x a + \frac{1}{2!}\left(\frac{\partial^2 f}{\partial x^2}\right)_x a^2 + \cdots + \frac{1}{n!}\left(\frac{\partial^n f}{\partial x^n}\right)_x a^n + \cdots \tag{9.11}$$

The first two terms in the Taylor expansion are often used to approximate $f(x+a)$ if a is sufficiently small

$$f(x+a) \approx f(x) + \left(\frac{\partial f}{\partial x}\right)_x a$$

Specific examples are:

$$\begin{aligned}\exp(x) &\approx 1 + x\\ \ln(1+x) &\approx x\\ \sqrt{1+x} &\approx 1 + \frac{1}{2}x\\ (1+x)^n &\approx 1 + nx\\ \sin(x) &\approx x\end{aligned}$$

where, in all cases, it has been assumed that $|x| \ll 1$.

9.5 Geometric Series

Consider the sum

$$S = \sum_{i=0}^{n} ax^i \tag{9.12}$$

Clearly,

$$xS = \sum_{i=0}^{n} ax^{i+1} = S - a + ax^{n+1} \tag{9.13}$$

Hence

$$S(1-x) = a(1-x^{n+1}) \tag{9.14}$$

or

$$S = \frac{a(1 - x^{n+1})}{1 - x} \tag{9.15}$$

If $|x| < 1$, we can take the limit $n \to \infty$

$$S_{n\to\infty} = \frac{a}{1 - x} \tag{9.16}$$

so

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1 - x} \tag{9.17}$$

for $|x| < 1$.

9.6 Factorials and Permutations

The symbol $N!$ denotes the "factorial" of N. For positive, integer N, it is defined as

$$N! = N \times (N - 1) \times (N - 2) \times \cdots \times 2 \times 1 \tag{9.18}$$

In addition, $0! \equiv 1$. The number of permutations of a set of N labeled objects is equal to $N!$. This can be demonstrated by induction. The number of ways in which a single object can be ordered is clearly equal to 1, which is equal to 1!. Hence, the relation holds for $N = 1$. The next step is to show that if the relation holds for N objects, it also holds for $N + 1$ objects. This is easily demonstrated as follows. Assuming that there are $N!$ permutations for N objects, then for every permutation there are $N + 1$ positions in the sequence where we could insert object $N + 1$. Hence the total number of permutations for $(N + 1)$ objects is $(N + 1) \times N! = (N + 1)!$. This completes the proof.

Now consider the following question: we have N labeled objects and we wish to count the number of distinct ways that these objects can be divided into two sets, such that one set contains M elements and the other $N - M$ elements. For instance, 3 objects can be distributed in 3 ways over a subset of size one and a subset of size 2:

$$(1, 2\ 3),\ (2, 3\ 1) \text{ and } (3, 1\ 2) \tag{9.19}$$

Note that we do not count different permutations within one subset as distinct. To compute this number in general, we consider all possible permutations of N objects. There are $N!$ such permutations. For every permutation, we attribute the first M elements to one set, and the remaining $N-M$ elements to the other. In this way, we get that the total number of permutations with M elements in one set and $N - M$ in the other is equal to $N!$. However, in this counting procedure, we have considered different permutations of the objects in either set as distinct. To get the total number of ways to distribute N objects over the two subsets, we should divide by the number of permutations in the set of M objects and in the set of $N - M$ objects. The result is that the number of ways to divide N objects over two subsets of size M and $N - M$ respectively, is given by

$$\frac{N!}{M!(N - M)!} \tag{9.20}$$

9.7 Binomial and Multinomial Distributions

As explained above, the number of ways to distribute N objects over two classes, in such a way that M objects end up in class I and $N - M$ objects in class II is given by

$$\frac{N!}{M!(N-M)!} \equiv \binom{N}{M} \tag{9.21}$$

For example: the number of ways to throw N coins, such that M are head and $N - M$ are tail, is $\binom{N}{M}$. If we assume that the probability of head and tail are both equal to $1/2$, then the probability that I throw M heads and $N - M$ tails is

$$P(M, N-M) = \binom{N}{M} \left(\frac{1}{2}\right)^N \tag{9.22}$$

In the more general case that the probabilities for the two events are not equal - say the probability to throw head (tail) is p $(1-p)$, then the probability to throw head M times and tail $N - M$ times is

$$P(M, N-M) = \binom{N}{M} p^M (1-p)^{N-M} \tag{9.23}$$

Of course, the sum of the probabilities of all different outcomes should add up to one

$$\sum \binom{N}{M} p^M (1-p)^{N-M} = (p + (1-p))^N = (1)^N = 1 \tag{9.24}$$

To give a specific example, consider two containers, one with volume V_1 and the other with volume V_2. We assume that the probability that a molecule will be in volume 1 is equal to $V_1/(V_1+V_2)$. The probability to find a molecule in volume 2 is then $1 - V_1/(V_1+V_2) = V_2/(V_1+V_2)$. The probability to find M molecules in V_1 and $N - M$ molecules in V_2 is then

$$P(M, N-M) = \binom{N}{M} \frac{V_1^M V_2^{N-M}}{(V_1+V_2)^N} . \tag{9.25}$$

The probability to find *all* molecules in volume 1 is

$$P(N, 0) = \frac{V_1^N}{(V_1+V_2)^N} \tag{9.26}$$

In case we distribute N objects over a larger number of classes - say m - the number of realizations is given by

$$\frac{N!}{\prod_{i=1}^{m} M_i!} \tag{9.27}$$

where M_i is the number of objects in class i and $\sum M_i = N$.

9.8 Some Integrals

Certain integrals occur time and again in statistical mechanics. First of all, there are the integrals of the type:

$$\int_0^\infty dx\, x^n \exp(-ax) \tag{9.28}$$

All these integrals can be derived through integration by parts from the integral

$$\int_0^\infty dx\ \exp(-ax) = 1/a \tag{9.29}$$

For instance

$$\begin{aligned}\int_0^\infty dx\, x\exp(-ax) &= -\frac{x}{a}\exp(-ax)|_0^\infty + \int_0^\infty dx\ \frac{\exp(-ax)}{a}\\ &= \frac{1}{a^2}\end{aligned} \tag{9.30}$$

The general result is

$$\int_0^\infty dx\ x^n\exp(-ax) = \frac{n!}{a^{n+1}} \tag{9.31}$$

This result can also be obtained by noting that

$$x^n\exp(-ax) = (-1)^n\left(\frac{\partial^n\exp(-ax)}{\partial a^n}\right) \tag{9.32}$$

and that therefore

$$\int_0^\infty dx\ x^n\exp(-ax) = (-1)^n\left(\frac{\partial^n(1/a)}{\partial a^n}\right) = \frac{n!}{a^{n+1}} \tag{9.33}$$

A second type of integral of particular importance is the Gaussian integral

$$I = \int_{-\infty}^\infty dx\ \exp(-cx^2) \tag{9.34}$$

A trick to compute this integral, is to consider its square

$$I^2 = \left(\int_{-\infty}^\infty dx\ \exp(-cx^2)\right)^2 = \int_{-\infty}^\infty dx\ \exp(-cx^2)\int_{-\infty}^\infty dy\ \exp(-cy^2) \tag{9.35}$$

We can write the latter product of integrals as

$$\int_{-\infty}^\infty dx\ \exp(-cx^2)\int_{-\infty}^\infty dy\ \exp(-cy^2) = \int_{-\infty}^\infty\int_{-\infty}^\infty dy\, dx\ \exp(-cx^2)\exp(-cy^2) \tag{9.36}$$

The latter integral is a two-dimensional integral. It can be simplified by using the polar coordinates r and ϕ, such that $x = r\cos\phi$ and $y = r\sin\phi$. Clearly, $x^2 + y^2 = r^2$. The integration range for ϕ is $\{0, 2\pi\}$ and r ranges from 0 to ∞. Finally, we replace the area element $dx\ dy$ by $rd\phi\ dr$. We can then write

$$\begin{aligned}I^2 &= \int_0^{2\pi} d\phi\int_0^\infty dr\ r\exp(-cr^2)\\ &= 2\pi\int_0^\infty \frac{1}{2}dr^2\ \exp(-cr^2)\\ &= \pi\int_0^\infty dr^2\ \exp(-cr^2)\\ &= \frac{\pi}{c}\end{aligned} \tag{9.37}$$

where, in the third line, we have used $dr^2 = 2r\ dr$. To arrive at the last equality, we used Eq. 9.29. Hence

$$\int_{-\infty}^\infty dx\ \exp(-cx^2) = \sqrt{\frac{\pi}{c}} \tag{9.38}$$

9.9 Stirling's Approximation

From Eqs. 9.31 and 9.38 above, we can derive Stirling's approximation for $N!$.

$$N! = \int_0^\infty dx\, x^N \exp(-x) = \int_0^\infty dx\ \exp(-x + N \ln x) \tag{9.39}$$

where we have used Eq .9.31 with $a = 1$. The integrand is sharply peaked at $x = N$. The value of the exponent at $x = N$ is $-N + N \ln N$. The first derivative is zero (we are at a maximum). The second derivative is $-1/N$. Hence, we can approximate the integral by

$$\begin{aligned} N! &\approx \int_0^\infty dx\ \exp(-N + N\ln N - \frac{(x-N)^2}{2N}) \\ &= \int_{-N}^\infty du\ \exp(-N + N\ln N - \frac{u^2}{2N}) \end{aligned} \tag{9.40}$$

where we have defined $u \equiv x - N$. As the function is sharply peaked, we can replace the lower limit of the integration by $-\infty$. We then have

$$\begin{aligned} N! &\approx \exp(-N + N\ln N) \int_{-\infty}^\infty du\ \exp(-\frac{u^2}{2N}) \\ &= \exp(-N + N\ln N)\sqrt{2\pi N} \\ &= N^N \exp(-N)\sqrt{2\pi N} \end{aligned} \tag{9.41}$$

where we have used Eq. 9.38. This is Stirling's approximation for $N!$. In fact, Stirling's approximation is the first term of a series

$$N! = N^N \exp(-N)\sqrt{2\pi N}\left(1 + \frac{1}{12N} + \frac{1}{288N^2} - \frac{139}{51840N^3} + \cdots\right) \tag{9.42}$$

Chapter 10

Appendix: Essential Thermodynamics

10.1 Thermodynamic States

Thermodynamics gives a macroscopic description of systems. A system in equilibrium is said to be in a *thermodynamic state*, which is characterized by *state variables*, e.g. volume V, temperature T, pressure P and amount of material. Often the amount of material is expressed in terms of the number of moles n. Throughout this work we shall use the number of particles $N = N_{av}n$, in which N_{av} is Avogadro's number. *Equations of state* are relations between state variables that express physical properties of the system. The ideal gas is an idealized dilute gas that satisfies the equation of state

$$PV = nRT = Nk_BT \tag{10.1}$$

where R is the gas constant and $k_B = R/N_{av}$ Boltzmann's constant. For a full thermodynamic description of a system a second equation of state is necessary. Often an *energy equation* is chosen, which expresses the energy or a free energy in terms of the state variables. For an ideal gas of point particles the energy equation is

$$U(V,T) = U(T) = \frac{3}{2}Nk_BT \tag{10.2}$$

in which $U(V,T)$ is the energy of the system as a function volume and temperature. Note that the energy equation is not completely independent of the equation of state, e.g. using the general relations Eq. 10.10 and Eq. 10.39 it follows from Eq. 10.1 that for an ideal gas $(\partial U/\partial V)_{N,T} = 0$, which is consistent with Eq. 10.2. On the other hand, Eq. 10.2 cannot be derived from Eq. 10.1 alone. The interdependence arises because both equations can be derived from the dependence of the free energy A on N, V and T. Indeed, from Eq. 10.16 we see that the pressure P for *any* system is given by $P = -(\partial A/\partial V)_{T,N}$. This is essentially the equation of state, resulting in Eq. 10.1 for an ideal gas. The entropy in the form $S = -(\partial A/\partial T)_{V,N}$ can be used in Eq. 10.15 to obtain the energy from $U = A + TS$.

State variables are called *extensive* if they are proportional to the extent of the system, i.e. if their value doubles when the volume, energy and number of particles are simultaneously doubled and *intensive* if they are not affected by such a doubling. The *molar* value of an extensive variable is an intensive variable obtained by dividing the extensive variable divided by N the number of mols in the system [*]. It is denoted by an overline. For example, the molar volume $\overline{V} \equiv V/N$. As the intensive properties of a one-component material do not change when particles are

[*]In thermodynamics one often divides by $n = N/N_{av}$, This alternative definition of molar variables leads to values that are N_{av} times larger and needs not lead to confusion.

added or removed in such a way that pressure and temperature remain constant, we may define molar quantities as partial derivatives as well, e.g. for the molar volume

$$\overline{V} = \frac{V}{N} = \left(\frac{\partial V}{\partial N}\right)_{P,T} \tag{10.3}$$

This definition is easily generalized to define *partial molar variables* in multi-component systems, e.g. the partial molar volume with respect to species i is defined as

$$\overline{V}_i = \left(\frac{\partial V}{\partial N_i}\right)_{P,T,N_{j\neq i}} \tag{10.4}$$

Here the notation indicates that for all components j the number of particles N_j should be kept constant, except for the component i with respect to which the partial molar volume is calculated.

10.2 Processes

The thermodynamic state of a system can change during *reversible* or *irreversible* processes. The process is described by *path variables*, e.g. q, the heat added or w, the work done on the system. Contrary to state variables, path variables are no system properties, they do not have a specific value for a system, the *the heat of a system* or *the work of a system* does not exist, not even if the system is in thermodynamical equilibrium. During a reversible process the system is in (or infinitesimally close to) a thermodynamic state at all moments, implying that all thermodynamic state variables are well-defined throughout the course of the process. During irreversible processes a system may be far out of equilibrium and one or more state variables may be undefined (e.g. there may be pressure waves or temperature gradients present in the system and the concept of *the* pressure P or *the* temperature T of the system does not make sense). In practice, reversible processes are idealized versions of real processes that proceed very slowly, while real spontaneous processes are irreversible.

10.3 First Law of Thermodynamics

The internal energy U of a system is an extensive state function that can only change by heat q that is added to the system or work w that is done on the system:

$$\Delta U = q + w \tag{10.5}$$

Important forms of work are volume work (compression or expansion under external pressure P), where $dw = -PdV$ and addition or removal of particles from a reservoir with chemical potential μ, where $dw = \mu dN$.

10.4 Second Law of Thermodynamics

There exists an extensive state function S, called entropy. When heat is reversibly added to the system, the entropy changes according to:

$$dS = \frac{dq_{\mathrm{rev}}}{T} \tag{10.6}$$

An isolated system has constant energy U, constant volume V and constant number of particles N. During irreversible processes in an isolated system the entropy of the system increases

$$(\Delta S)_{U,V,N,\mathrm{irr}} > 0 \tag{10.7}$$

Using the entropy and the definition of heat capacity C_V at constant V or C_P at constant P we may write

$$C_V = \left(\frac{\partial U}{\partial T}\right)_{N,V} = T\left(\frac{\partial S}{\partial T}\right)_{N,V} \tag{10.8}$$

$$C_P = \left(\frac{\partial H}{\partial T}\right)_{N,P} = T\left(\frac{\partial S}{\partial T}\right)_{N,P} \tag{10.9}$$

10.5 The Basic Machinery

The first and second law can be combined to give

$$dU = TdS - PdV + \mu dN \tag{10.10}$$

From this formula it is seen that U is the *characteristic state function* for systems in which S, V and N are constant or can be easily controlled. Irreversible processes at constant S, V and N lead to a decrease of internal energy

$$(\Delta U)_{S,V,N,\mathrm{irr}} < 0 \tag{10.11}$$

For systems with other control variables other characteristic state function are available.

- *Enthalpy* for S, P and N

$$H = U + PV \tag{10.12}$$

$$dH = TdS + VdP + \mu dN \tag{10.13}$$

$$(\Delta H)_{S,P,N,\mathrm{irr}} < 0 \tag{10.14}$$

- *Free energy or Helmholtz free energy* for N, V and T

$$A = U - TS \tag{10.15}$$

$$dA = -SdT - PdV + \mu dN \tag{10.16}$$

$$(\Delta A)_{N,V,T,\mathrm{irr}} < 0 \tag{10.17}$$

- *Gibbs free energy* for N, P and T

$$G = U - TS + PV = A + PV = H - TS \tag{10.18}$$

$$dG = -SdT + VdP + \mu dN \tag{10.19}$$

$$(\Delta G)_{N,P,T,\mathrm{irr}} < 0 \tag{10.20}$$

From Eq. 10.19 it follows that G increases proportional to N if the system is (mentally) built up by adding molecules, meanwhile adjusting the volume and adding heat in such a way that P and T remain constant. This proofs that $G = \mu N$ which expresses that the chemical potential is the molar Gibbs free energy $\mu = \overline{G}$. From this equation it follows that $dG = \mu dN + Nd\mu$. Upon combining with Eq. 10.19 we find the *Gibbs-Duhem* relation

$$VdP - SdT - Nd\mu = 0 \tag{10.21}$$

This relation shows that the three intensive variables P, T and μ can not be varied independently.

10.6 Definitions and Relations

For and state function $f(x, y, z)$, we can write

$$df = \left(\frac{\partial f}{\partial x}\right)_{y,z} dx + \left(\frac{\partial f}{\partial y}\right)_{x,z} dy + \left(\frac{\partial f}{\partial z}\right)_{x,y} dz \tag{10.22}$$

Therefore, the first derivatives of characteristic functions can be used as *thermodynamic definition* for certain state variables, i.e. from

$$dA = -SdT - PdV + \mu dN \tag{10.23}$$

follows immediately that

$$-S = \left(\frac{\partial A}{\partial T}\right)_{V,N} \tag{10.24}$$

$$-P = \left(\frac{\partial A}{\partial V}\right)_{T,N} \tag{10.25}$$

$$\mu = \left(\frac{\partial A}{\partial N}\right)_{V,T} \tag{10.26}$$

Differentiation of Eqs. 10.10, 10.13, 10.16 and 10.19 with respect to N leads to several equivalent definitions of chemical potential

$$\mu = \left(\frac{\partial U}{\partial N}\right)_{S,V} = \left(\frac{\partial H}{\partial N}\right)_{S,P} = \left(\frac{\partial A}{\partial N}\right)_{T,V} = \left(\frac{\partial G}{\partial N}\right)_{T,P} = \overline{G} \tag{10.27}$$

It depends on the control variables of the system which definition is preferred. These definitions should be used carefully. E.g. note that the molar free energy $\overline{A}$ is not equal to μ, even though the expressions look similar:

$$\overline{A} = \frac{A}{N} = \left(\frac{\partial A}{\partial N}\right)_{P,T} = \mu - P\left(\frac{\partial V}{\partial N}\right)_{P,T} = \mu - k_B T \tag{10.28}$$

where, of course, the last equality is valid for an ideal gas only. If temperature is a control variable, then one gets two entropy definitions

$$S = -\left(\frac{\partial A}{\partial T}\right)_{N,V} = -\left(\frac{\partial G}{\partial T}\right)_{N,P} \tag{10.29}$$

If entropy is a control variable, then one could define temperature by

$$T = \left(\frac{\partial U}{\partial S}\right)_{N,V} = \left(\frac{\partial H}{\partial S}\right)_{N,P} \tag{10.30}$$

The thermodynamic definitions of pressure

$$P = -\left(\frac{\partial U}{\partial V}\right)_{S,V} = -\left(\frac{\partial A}{\partial V}\right)_{N,T} \tag{10.31}$$

give experimental access to the volume dependence of U and A. Similarly, the thermodynamic definitions of volume

$$V = \left(\frac{\partial H}{\partial P}\right)_{S,N} = \left(\frac{\partial G}{\partial P}\right)_{N,T} \tag{10.32}$$

give the pressure dependence of H and G. The temperature dependence of U and H is found from heat capacity measurements as is seen in Eq. 10.8 and Eq. 10.9. The temperature dependence of A and G is found from *Gibbs-Helmholtz relations*, which are derived by combining Eq. 10.15 with Eq. 10.8, and Eq. 10.18 with Eq. 10.9 respectively:

$$\left(\frac{\partial A/T}{\partial T}\right)_{N,V} = -\frac{U}{T^2} \tag{10.33}$$

$$\left(\frac{\partial G/T}{\partial T}\right)_{N,P} = -\frac{H}{T^2} \tag{10.34}$$

Integrating experimental data properly thus forms the basis of tabulating the energy, enthalpy and the (Gibbs) free energy as a function of T and P.

10.7 Maxwell Relations

Maxwell relations are relations between derivatives of state variables. These derivatives are a twice differentiated state function. The relations are obtained by interchanging the order of differentiation. Take e.g. the Gibbs free energy. From Eq. 10.19 it follows that

$$S = -\left(\frac{\partial G}{\partial T}\right)_{N,P} \tag{10.35}$$

and

$$V = \left(\frac{\partial G}{\partial P}\right)_{N,T} \tag{10.36}$$

The identity

$$\left(\frac{\partial^2 G}{\partial P \partial T}\right)_N = \left(\frac{\partial^2 G}{\partial T \partial P}\right)_N \tag{10.37}$$

leads to

$$\left(\frac{\partial S}{\partial P}\right)_{N,T} = -\left(\frac{\partial V}{\partial T}\right)_{P,N} \tag{10.38}$$

Similarly, using Eq. 10.16 leads to

$$\left(\frac{\partial S}{\partial V}\right)_{N,T} = \left(\frac{\partial P}{\partial T}\right)_{P,N} \tag{10.39}$$

These relations could not easily be guessed from physical arguments alone. They show that, although S can not be measured experimentally, its derivatives with respect to P, V and T (see Eqs. 10.8 and 10.9 can be experimentally determined. Integrating experimental data properly thus forms the basis of tabulation of entropy as a function of T and P.

10.8 Phase Equilibrium

If a system at given P, T and N contains two or more phases of the same type of particles, then the two phases will exchange volume, heat and particles until the minimum Gibbs free energy is reached. At the moment that equilibrium has been reached one of the two phases may have disappeared. If both phases are still present, then they are both at pressure P, temperature T and they have equal chemical potential μ. A (P,T)-phase diagram displays the phase transition lines and points, where two or more phases are in equilibrium with each other, i.e. where the chemical potential at the given P and T are the same for these phases. The points with two phases in equilibrium are located on phase equilibrium lines, e.g. the melting and boiling lines. Where two of these lines intersect three phase equilibrium is possible, and such a point is called triple point. If a line ends in a point without intersection with other lines, this is called a critical point.

Along phase equilibrium lines not only extensive variables, like S and V, are different for both phases, but also the molar value for all components j, like the molar entropy $\overline{S} = S/N$ and the molar volume $\overline{V} = V/N$. Approaching a critical point the differences of such molar quantities become smaller, at a critical point they become equal. Applying the Gibbs-Duhem relation to two phases in equilibrium, the slope of phase equilibrium lines can be expressed in terms of the differences $\Delta_{\mathrm{trs}}\overline{S}$ and $\Delta_{\mathrm{trs}}\overline{V}$. The result is the Clapeyron equation for phase transition lines

$$\left(\frac{dP}{dT}\right)_{\mathrm{coex}} = \frac{\Delta_{\mathrm{trs}}\overline{S}}{\Delta_{\mathrm{trs}}\overline{V}} = \frac{\Delta_{\mathrm{trs}}\overline{H}}{T\Delta_{\mathrm{trs}}\overline{V}} \tag{10.40}$$

For calculating the equilibrium vapor pressure of a solid or a liquid, it is often justified to assume that the vapor is an ideal gas, i.e. $\overline{V}_{\mathrm{vap}} = k_B T/P$ and that the liquid molar volume is negligibly small, $\Delta\overline{V} \approx \overline{V}_{\mathrm{vap}}$. Then

$$\left(\frac{d\ln P}{dT}\right)_{\mathrm{coex}} \approx \frac{\Delta_{\mathrm{vap}}\overline{H}}{k_B T^2} \tag{10.41}$$

which is known as the Clausius-Clapeyron equation.

Chapter 11

Appendix: Introduction to C

11.1 Introduction

11.1.1 The language

C is a general-purpose computer programming language developed in 1972 by Dennis Ritchie at the Bell Telephone Laboratories for use with the Unix operating system. Although C was designed for implementing system software, it is also widely used for developing portable application software. C is one of the most popular programming languages. It is widely used on many different software platforms, and there are only a few computer architectures for which a C compiler does not exist. C has greatly influenced many other popular programming languages, most notably C++, which originally began as an extension to C. C is a relatively simplistic language, lacking some of the features of newer languages. However, for calculation and simulation, these features can be done without, as the needed operations there are mostly straightforward as well.

Useful webpages on C are:

- `https://www.tutorialspoint.com/cprogramming/`
- `https://www.cprogramming.com/tutorial/c-tutorial.html`

11.1.2 A first program

As is usual when starting out with a programming language, here is a simple program:

```
#include <stdio.h>

int main()
{
  printf("Hello, world!\n");
  return (0);
}
```

To run this program, we will have to compile it into an executable file. For this, we save the file as hello.c and compile the code with a compiler called gcc. We can run it from a terminal window:

```
gcc hello.c -o hello
```

The gcc command is always supplied with the file(s) to compile, and a number of compiler options. The only option here is "-o hello", which tells the compiler that the computer program it creates should be called "hello".

If you have entered this correctly, you should now see a file called "hello". This file is the binary version of your program, and when run should display "Hello, world!"

Here is an example of what compiling and running looks like when using a terminal on a unix system. ls is a common unix command that will list the files in the current directory, which in this case is the directory 'progs' inside the home directory (represented with the special tilde, ∼ , symbol).

```
~/progs$ ls
hello.c
~/progs$ gcc -o hello hello.c
~/progs$ ls
hello  hello.c
~/progs$ ./hello
Hello, world!
~/progs$
```

We'll take a quick look at the lines of the program individually. The first line tells the compiler to include the standard header file <stdio.h>. There are many such header files, each containing standard function descriptions which can be used in your program. In this case, <stdio.h> supplies a number of standard input and output functions, such as printf.

The line 'main()' shows the start of the function 'main'. Each C program has a 'main' function. Generally, the 'main' function is where a program begins. The body of the function is enclosed by braces ({}), which show where the function starts and ends.

The line that does the actual printing is below this: the printf function is a function for printing formatted strings to the screen, and is defined in <stdio.h>. It can print strings, but also the value of any variables in your code. See the section on printing later on for details. The "\n" in this case is an escape sequence which adds a new line at the end of the printed text.

Every command in C is ended by a semicolon (;). This is easy to forget when starting out, and can cause confusing error messages from the compiler. This also means that starting a new line in your code if a statement is very long is always possible.

Note that many of the examples given in this chapter are not full programs. Often just a few lines or functions are given. You may have to add a main function and include the right libraries to try them out.

11.1.3 Operators

The operators you will need for basic programming are fairly straightforward:

```
x = 1;          //Set x to one
x = 1 + 2;      //Addition
y = 5 - x;      //Subtraction
z = y * x;      //Multiplication
```

```
x = x / 2;          //Division
y = 10 % 2;         //10 mod 2
x += 6;             //Add 6 to x.
x /= 3;             //Divide x by three.
x -= 2;             //*= and -= work similarly
x = (1 + 2) * 3;        //Brackets
```

Spaces in these statement are not important at all, but may make your code look nicer.

11.1.4 Coding

When coding, there are a few things to keep in mind.

First of all, whitespace generally does not affect the code. This means you can put extra spaces, tabs, and line breaks anywhere you want.

It is generally useful to use indentation to show the structure of your program. Use spaces to move lines within a function, loop, or if-statement to the right. This way, you can easily see what code is grouped together.

If a line of code contains two slashes in a row (//), then anything after this is designated as a comment, and ignored by the compiler. This allows you to leave notes explaining what your code is doing, or to temporarily disable a line of code. In addition, anything between /* and */ is also ignored, even if this spans multiple lines.

Example:

```
#include <stdio.h>

main()
{
  int i;
  for (i = 1; i < 20; i++)  //Start loop
  {
    if (i % 2 == 0)         //is i divisible by two?
    {
      printf ("%d is even!\n", i);
    }
    else
    {
      printf ("%d is odd!\n", i);
    }
  }
}
```

The following code is equivalent, but a lot less easy to read...

```
#include <stdio.h>

main(){
int i;
for (i=1;i<20;i++)
if (i%2==0) printf("%d is even!\n",i);
else printf("%d is odd!\n",i);}
```

11.2 Header files

As shown in the first little program, it is often necessary to include header files at the start of your program to use some of the built-in functions in C. If there are multiple files to include, you can simply use multiple #include lines.

<stdio.h> includes all standard input and output functions. This includes printing to the screen, reading from and writing to files and getting input from the keyboard. You will likely need this in every C program you write.

<stdlib.h> is the standard library, including a number of standard functions. The important ones for this course are the random number generator ′drand48()′, the absolute value ‘abs()′, and memory allocating functions.

<string.h> includes a number of functions for manipulating strings.

<math.h> includes a large number of mathematical functions, such as ′exp′, ′sin′, ‘atan′, ‘acos′, ′sqrt′, etc. Also useful is the function ′pow(x,y)′, which returns x^y. NOTE: When including <math.h> you also have to let the compiler know to link this library. You do this by adding the compiler option ″-lm″ to your compile command.

```
gcc -lm prog.c -o prog
```

Some examples of the math functions:

```
#include <math.h>

main()
{
  double x,y,z;
  double pi;
  pi = atan (1) * 4;     //Calculate pi
  x = exp(pi);
  y = sinh(pi);
  z = pow(y,x);
}
```

11.3 Variables

11.3.1 Data types

There are many types of variables in C. The most useful ones include:

int: This is an integer, either positive or negative.

double: A double-precision floating point number, used for any real number you might want to use.

float: A single-precision floating point number. For most calculations that require any accuracy, it is better to use doubles.

char: A single character, such as 'a'.

You can change the data types of a value by assigning it to a variable with the new data type, or by directly telling the program it should treat a variable as a variable of a different type. This is done by placing the new variable type between brackets (see below). For example, if x is a double, '(int) x' instead uses x as an integer, rounding down.

```
double x = 3.5;
int i = x;          //i is now 3
double a = i;          //a is now 3.0
double k = i / 2;      //Integer division: k is now 1
double m = (double) i / 2;
             //Treat i as a double: m is 1.5
printf ("%d\n", (int) x); //Prints x as an integer, so 3.
```

11.3.2 Declaring variables

Before using a variable anywhere in your program, it must be declared. Declaring variables is the way in which a C program shows the number of variables it needs, what they are going to be named, and how much memory they will need. After the declaration, the variable does not have a well-defined value yet. You should only use the value of a variable after it has actually been given one. In the example below, the print statement tries to print the value of y, but since it has not been given a value, the outcome of this is unpredictable. Do not assume the value is zero.

```
#include <stdio.h>

int j = 5;        //Global variable

main()
{
  int i;          //Some declarations
  double x,y;
  double a = 0.5;    //Declaration and initialization
  i = 3;          //Initialization of i
  i++;            //Add one to i, i is now 4
  x = i * j * a;
  printf ("i = %d, x = %lf, y = %lf\n", i,x,y);
}
```

Note that variable names (and function names) are case-sensitive.

Any variables declared outside any function, just below the #include lines, is considered a global variable. This means that any function in your program can access and modify this variable. If a variable is declared within a function, it is only available in that function. This also means you can use a variable with the same name in another function.

11.3.3 Arrays

In many cases, you will need to work with lists of values, such as coordinates, rather than single values. A list in C is called an array, and can be declared as follows:

```
int values[3];
values[0] = 1;
values[1] = 2;
values[2] = values[0] + values[1];
// values[3] = 4;  is not allowed!!
```

Array indices always start at zero, and reading or writing outside the array bounds can cause your program to crash. If this happens, you will usually see the message "Segmentation fault". Note that declaring an array as a global variable can only be done if the size of the array is a constant:

```
#include <stdio.h>
#define SIZE 5      //Replaces all occurences of SIZE
           //in the program by 5

int s = 5;
int values1[5];     //This works
int values2[SIZE];  //This works
int values3[s];     //This doesn't work

main()
{
  //Do stuff
}
```

Operations on arrays can generally not be done in one command in C. Normally, array operations are done in loops.

11.4 Loops and conditions

11.4.1 if

With if statements you can create code that is only executed if certain conditions are met. You can use 'else' if you also want other code to execute if the if statement turns out to be false. The code to execute is enclosed by curly braces, though these can be omitted if you only want to have one statement depend on the condition.

```
func(int i)
{
  if (i == 0)
  {
    printf("This number is zero!\n");
  }
  else if (i > 0)
```

```
{
  printf ("A positive number\n");
}
else
{
  printf ("Negative!\n");
}
if (i != 1 && i != 8 ) printf ("Not one or eight.\n");
  // && means "and"
  // || means "or"
  // != means "not equal"
```

Note that to check if two numbers are equal, you need two equals signs (==).

11.4.2 for

To execute a block of code multiple times, you can use for loops. These generally look like this:

```
int i;
for (i = 0; i < 10; i++)          //i++ adds one to i
{
  //Code to run 10 times
  printf ("i is now %d\n", i);
}
```

The for statement has three parts. The first part is executed when the loop begins, and sets the starting value for the loop index. The second part is the condition for the loop: it keeps going until this condition is untrue. The last part is executed at the end of every cycle in the loop. In the code above, this causes the integer i to run from 0 to 9, and the body of the for loop is executed for each value of i. This is very useful for doing operations on arrays.

11.4.3 while

Another common type of loop is the while loop. While loops simply continue looping until a certain condition evaluates to false.

```
int i = 1000;
int n = 0;
while (i >= 1)  //i++ adds one to i
{
  i = i / 2;
  n += 1;   //adds one to n
}
```

11.4.4 Escaping loops

Sometimes you want to get out of a loop before it would normally end. In this case, the break statement will break out of the innermost loop. The continue statement will just move on to the next iteration of the loop.

```
//Search for squares in a silly way:
int i;
int notthis = 3;    //This number we skip
for (i = 0; i < 10; i++)
{
  if  (i == notthis) continue;
  for (j = 0; j < i; j++)
  {
    if (j * j == i)
    {
      printf ("%d squared = %d\n", j, i);
      break;    //Move on to the next i
    }
  }
}
```

11.5 Functions

In C programming, all executable code resides within a function. A function is a named block of code that performs a task and then returns control to a caller. A function is often executed (called) several times, from several different places, during a single execution of the program. After finishing a subroutine, the program will branch back (return) to the point after the call.

The following function, for example, calculates the square of a number:

```
int square(int x)
{
   int sq;
   sq = x * x;
   return sq;
}

printsquares ()
{
  int i,isquare;
  for (i = 0; i < 10; i++)
  {
    isquare = square(i);
    printf ("%d squared is %d\n", i, isquare);
  }
}
```

Functions can return a value, but they don't necessarily have to do so. If a function returns a value, it will state what type of value it returns at the start of the function, before the name. The above function square returns an integer, while printsquares does not return anything.

A function can also take any number of arguments, which can be used within the function. The function square, for example, takes one integer as its argument. In C, a function normally is only able to change any of its arguments locally: try the following program.

```
#include <stdio.h>
main ()
```

```
{
  int x = 3;
  func (x);
  printf ("In main, x is %d\n", x);
}

func (int x)
{
  x = 2 * x;
  printf ("In func, x is %d\n", x);
}
```

The variable x is set to 3 in the main function. The function func receives this value 3, doubles it, and prints the result. However, the value of x in the main function has not changed. If you want to use results from a function, you have three options. The first way is to let the function return a value, as shown above. Another option is the use of global variables. Any changes you make to global variables in any function will be seen by any other function using those variables. The last option is a bit more complicated, and requires the use of pointers.

11.5.1 Pointers

Pointers are a slightly more tricky concept in C, but they have many uses. We won't need them in many cases in C, so this section only covers the basics. A pointer is a variable that holds the memory address of another variable. To work with pointers, two operators are important. The operator & means 'address of', and a * means 'value at'. The following example shows the use of a pointer to an integer:

```
int* pt;    //pt is a pointer to an integer
int i = 5;
pt = &i;    //Set pt equal to the address of i
*pt = 2 * *pt;  //Double the value at pt;
printf ("i is now %d\n", i);
```

Of course, we could use 'i = 6' instead, so what's the use? It becomes a lot more useful if we pass the address of i to a function:

```
#include <stdio.h>
main ()
{
  int i = 5;
  twice (&i);
  printf ("i is now %d\n", i);
}

twice (int* pt)
{
  *pt = 2 * (*pt);
}
```

This way, we can change the value one or more parameters of a function, rather than just returning one value and/or changing global variables. There are several built-in functions that work the same way, such as fscanf.

11.5.2 Static variables

When leaving a function, any local variables will be forgotten. The next time you enter the function, it will not be influenced by any previous function calls. You can make your program remember local variables by declaring them with the static keyword:

```
count ()
{
  static int counter = 0;
  int i = 0;
  i ++;
  counter ++;
  printf("Function called %d times, but i is still %d\n",
              counter, i);
}
```

11.6 Printing and reading

Most programs will at some point write something to the screen or to a file. You can write to the screen this using a printf ('print formatted') statement, as shown in many examples in this chapter. The printf arguments consist of one string to print, and a number of variables to fill in at specified points in the string. As mentioned before, '\n' is a line break.

```
double x = 1.0/3.0;
int i = 4;

printf ("%d, %lf\n", i, x); //%d: integer
                   //%lf: double
    //lf stands for 'long float'
```

You can format the printed numbers in various ways. For example '%.2lf' will round a double to two digits after the decimal point, and '%3d' will add spaces in front of the integer to make sure the total width printed is at least 3.

You can also print to files, using fprintf. Files have to be opened first, and closed after you have finished writing. File pointers have the type 'FILE*'.

```
FILE* file;
file = fopen ("out.dat", "w");

fprintf (file, "This text ends up in the file\n");

fclose(file);
```

The above code opens a file named out.dat, and writes to it. The 'w' passed to fopen tells it to open the file for writing. This means the file is emptied before writing to it. If you replace the 'w' with an 'a', it will append to the file instead, leaving anything already there in place. You can also open for reading:

```
#include <stdio.h>
#define MAX 1000
```

```
double x[MAX];
double y[MAX];
int npart;

main()
{
  int i;
  readparts();
  for (i = 0; i < npart; i++)
  {
    printf ("%2d: (%lf, %lf)\n", i, x[i], y[i]);
  }
}

readparts()
{
  int i;
  int numbersread;
  FILE* file;
  file = fopen ("in.dat", "r");
  if (file == NULL) //fopen returns NULL if it fails
  {
    printf ("File not found!\n");
    return;
  }
  fscanf (file, "%d\n", &npart);
  if (npart > MAX) npart = MAX; //At most MAX particles
  for (i = 0; i < npart; i++)
  {
    numbersread=fscanf(file,"%lf %lf\n",&(x[i]),&(y[i]));
    if (numbersread != 2) //Something wrong?
    {
      printf ("Read error!\n");
      break;
    }
  }
  fclose(file);
}

/* Example in.dat:
2
1.0 1.0
2.0 2.0
*/
```

The above program reads in a file called in.dat, and expects it to contain a line with the number of particles, and then lines with xy-coordinates for each particle. If the file isn't there, or if it can't read the right numbers, it will complain. It then prints the result. The function fscanf works very similar to fprintf, except that it takes in the address of any variables it should assign data to, using the & operator. It returns the number of values read, which can be used to check if anything went wrong.

In addition to printing to the screen and to files, it can also be useful to print to a string, using the function sprintf. Strings are just arrays of characters.

```
char str[20];
int x = 3;
sprintf (str, "x%d", x);
printf ("String: %s\n", str);
```

11.7 Example program

As a conclusion, let's look at an example.

11.7.1 Step 1: Specify the Problem

We would like to write a program which evaluates the roots of the quadratic equation $ax^2+bx+c=0$ which are given by the following formula:

$$x = \frac{-b \pm \sqrt{b^2-4ac}}{2a} \tag{11.1}$$

Note that there will be three cases of interest, when the discriminant ($D = b^2 - 4ac$) is 0, positive or negitive.

11.7.2 Step 2: Write a Pseusocode (or Algorithm)

- read in the values of a, b, and c for which you would like to know the roots.
- if a is zero stop - in this case we do not have a quadratic equation
- evaluate D
- if $D = 0$ then the root is $\frac{-b}{2a}$
- if $D > 0$ then there are two real roots $\frac{-b\pm\sqrt{D}}{2a}$
- if $D < 0$ then there are two imaginary roots $\frac{-b\pm i\sqrt{-D}}{2a}$
- print the solution

11.7.3 Step 3: Write the Program

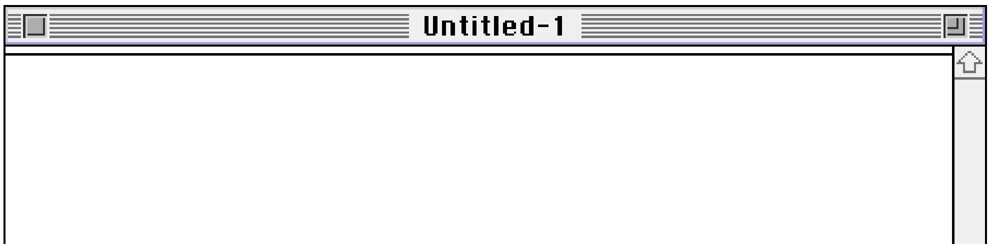
```
#include <stdio.h>
#include <math.h>

main()
{
  double a, b, c;
  double D, real, imag;
  double root1, root2;
  double delta = 0.0000000001; //rounding error
  printf ("What are a, b, and c?\n");
  scanf ("%lf\n%lf\n%lf", &a, &b, &c);
  printf ("%lf x^2 + %lf x + %lf\n", a, b, c);
  if (a == 0)
  {
    printf ("Not a quadratic equation!\n");
    return;
  }
  D = b*b -4*a*c;
  if ( D > -delta && D < delta )
  {
    root1 = -b / (2 * a);
    printf ("There is only one root: %lf\n", root1);
  }
  else if (D > 0)
  {
    root1 = (-b - sqrt(D)) / (2 * a);
    root2 = (-b + sqrt(D)) / (2 * a);
    printf ("There are two roots: %lf, %lf\n", root1, root2);
  }
  else
  {
    real = -b / (2 * a);
    imag = sqrt(-D) / (2 * a);
    printf ("There are two complex roots: %lf + %lfi, %lf - %lfi\n",
                    real, imag, real, imag);
  }
}
```

Chapter 12

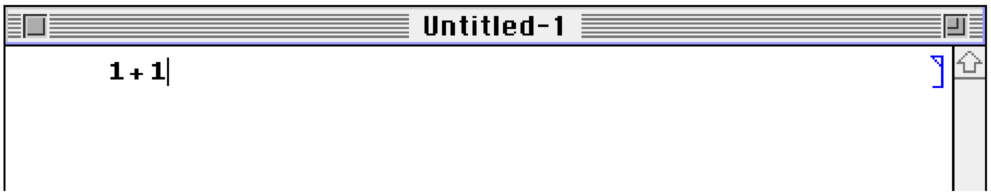
Appendix: Short Introduction into Mathematica

12.1 Getting started

To start Mathematica, double-click on the Mathematica icon. When you start Mathematica you will be presented with an empty window similar to the window shown here:

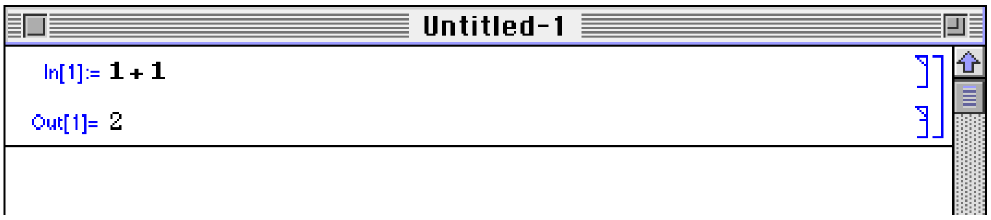


You can open a new window in your copy of Mathematica by selecting New from the File menu. To enter your input, select the new window by clicking on it, and start typing. After entering your input, the window will look something like this:



```
1 + 1
```

To evaluate your input, press Shift-Enter (press the enter key while holding down the shift key). When the calculation is finished, the result is displayed just below your input:



```
In[1]:= 1 + 1

Out[1]= 2
```

When you are finished with your calculations you can end your Mathematica session (which will close this window) by selecting Quit from the File menu. To stop a calculation, select interrupt or abort Evaluation or Quit Kernel from the Evaluation tab.

12.2 Help

1. A good tutorial can be found here:
   http://library.wolfram.com/conferences/devconf99/withoff/index2.html.
2. For help with individual functions, for example with a function whose name begins with Expression, type:

?Expression	
??Expression	for more help
?E*	for help on all functions beginning with E.

3. For help on mathematical functions, use the Function Browser under the Help menu.
4. With `Esc` you can type in special characters like by typing `Esc pi Esc`.

12.3 Elementary calculations

12.3.1 Arithmetic

+ is the sum of two numbers, while - is minus

2.3 + 5.63 - 3.00

```
4.93
```

/ stands for division and ˆ for power

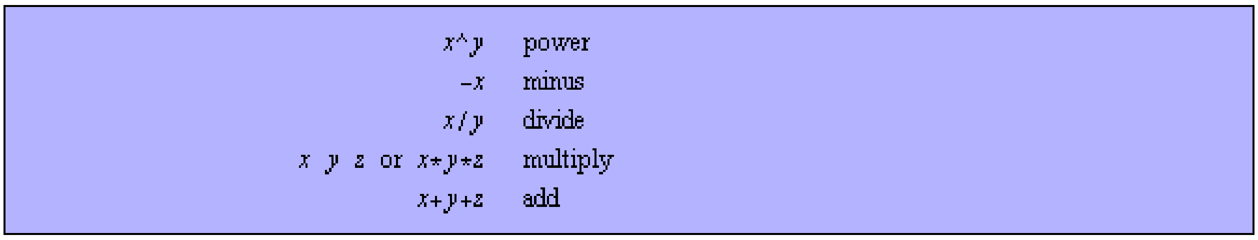
2.4/8.9ˆ2

```
0.0302992
```

Spaces stand for multiplication, but you can also use *

2 3 4

```
24
```



x^y	power
-x	minus
x/y	divide
x y z or x*y*z	multiply
x+y+z	add

//N gives an approximate numerical value and % denotes the previous output. %% the output before this, et cetera. %n denotes Out[n].

2^100

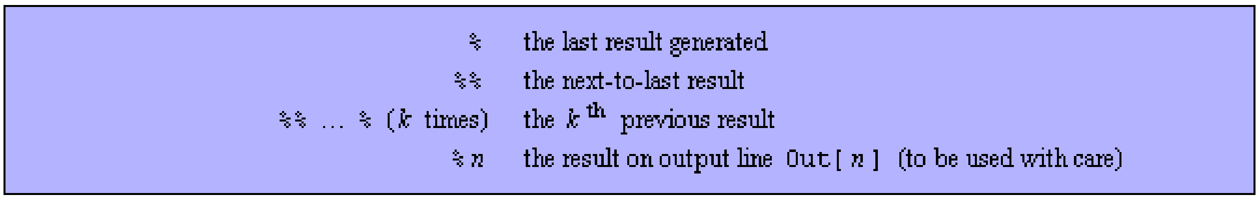
```
1 267 650 600 228 229 401 496 703 205 376
```

N[%]

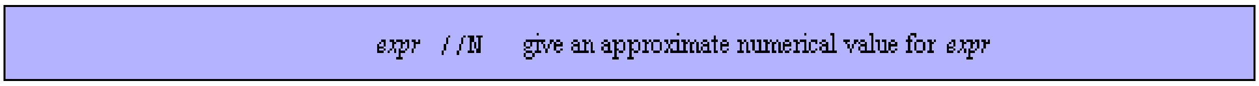
1.26765x10^{30}

1/3 + 1/16 //N

```
0.395833
```



%	the last result generated
%%	the next-to-last result
%% ... % (*k* times)	the *k* th previous result
% *n*	the result on output line Out[*n*] (to be used with care)



expr //N	give an approximate numerical value for *expr*

12.3.2 Some mathematical functions

Sqrt[x] gives the square root of x

Sqrt[3.0]

```
1.73205
```

Mod[m,n] gives the remainder on division of m by n.

Mod[1,2]

```
1
```

Random[] gives a uniformly distributed random number in the range 0 to 1.

Random[]

```
0.227695
```

RandomReal[{a,b}] gives a uniformly distributed real random number in the range a to b

RandomReal[{-0.5,0.5}]

```
-0.427695
```

RandomInteger[{a,b}] gives a uniformly distributed random integer in the range a to b

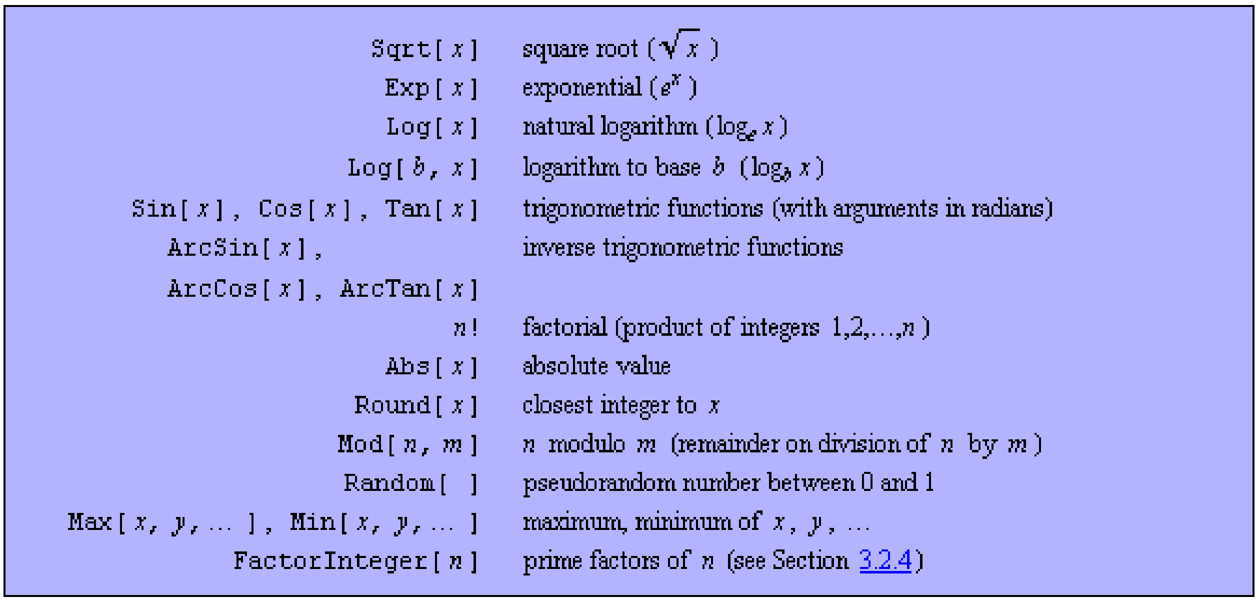
RandomInteger[{1,100}]

```
56
```

IntegerPart[x] gives the integer part of x.

IntegerPart[13.45]

```
13
```



`Sqrt[x]`	square root ($\sqrt{x}$)
`Exp[x]`	exponential (e^x)
`Log[x]`	natural logarithm ($\log_e x$)
`Log[b, x]`	logarithm to base b ($\log_b x$)
`Sin[x], Cos[x], Tan[x]`	trigonometric functions (with arguments in radians)
`ArcSin[x], ArcCos[x], ArcTan[x]`	inverse trigonometric functions
`n!`	factorial (product of integers 1,2,…,n)
`Abs[x]`	absolute value
`Round[x]`	closest integer to x
`Mod[n, m]`	n modulo m (remainder on division of n by m)
`Random[ ]`	pseudorandom number between 0 and 1
`Max[x, y, … ], Min[x, y, … ]`	maximum, minimum of x, y, …
`FactorInteger[n]`	prime factors of n (see Section 3.2.4)

12.3.3 Defining Variables

When you do long calculations, it is often convenient to give names to your intermediate results. Just as in standard mathematics, or in other computer languages, you can do this by introducing named variables.

This sets the value of the variable x to be 5.

x = 5

```
5
```

Whenever x appears, Mathematica now replaces it with the value 5.

x^2

```
25
```

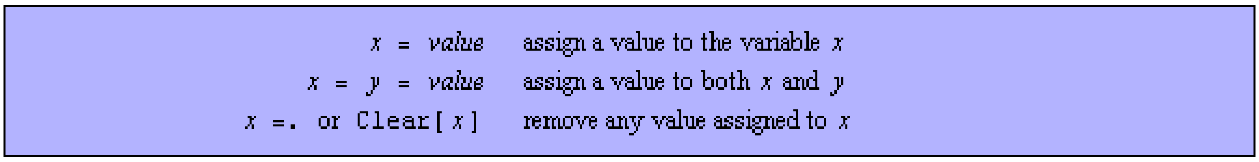
To unassign x use:

x =. ;

x

```
x = x
```

You can perform two evaluations by separating them with a semicolon " ; "



x = *value*	assign a value to the variable *x*
x = *y* = *value*	assign a value to both *x* and *y*
x =. or Clear[*x*]	remove any value assigned to *x*

12.3.4 Making tables

This gives a table of the values of i^2, with i running from 1 to 6.

x=Table[i^ 2, {i, 1, 6}]

```
{1, 4, 9, 16, 25, 36}
```

//TableForm displays the table in tableform

% //TableForm

```
 1
 4
 9
 16
 25
 36
```

x[[4]] extracts the fourth element

x[[4]]

```
16
```

This creates a 2x3 table, and gives it the name m.

m = Table[i - j, {i, 3}, {j, 2}]

```
{{0,-1},{1,0},{2,1}}
```

This extracts the first sublist from the list of lists that makes up the table.

m[[1]]

```
{0,-1}
```

This extracts the second element of that sublist.

m[[1,2]]

```
-1
```

This displays m in a "tabular" form.

TableForm[m]

```
 0  -1
 1  0
 2  1
```

Flatten[m,1] flattens table m to level 1.

Flatten[m,1]

```
{0,-1,1,0,2,1}
```

This is a 2x2 matrix.

m = {{a, b}, {c, d}}

```
{{a, b}, {c, d}}
```

Here is the first row.

m[[1]]

```
{a,b}
```

Here is the element m_{12}.

m[[1,2]]

```
b
```

Another example:

xy=Table[{i,j},{i,1,3},{j,1,4}]

```
{{{1,1},{1,2},{1,3},{1,4}},{{2,1},{2,2},{2,3},{2,4}},

{{3,1},{3,2},{3,3},{3,4}}}
```

TableForm[xy]

```
1  1  1  1
1  2  3  4
2  2  2  2
1  2  3  4
3  3  3  3
1  2  3  4
```

xy=Flatten[xy,1]

```
{{1,1},{1,2},{1,3},{1,4},{2,1},{2,2},{2,3},{2,4},{3,1},{3,2},{3,3},{3,4}}
```

TableForm[xy]

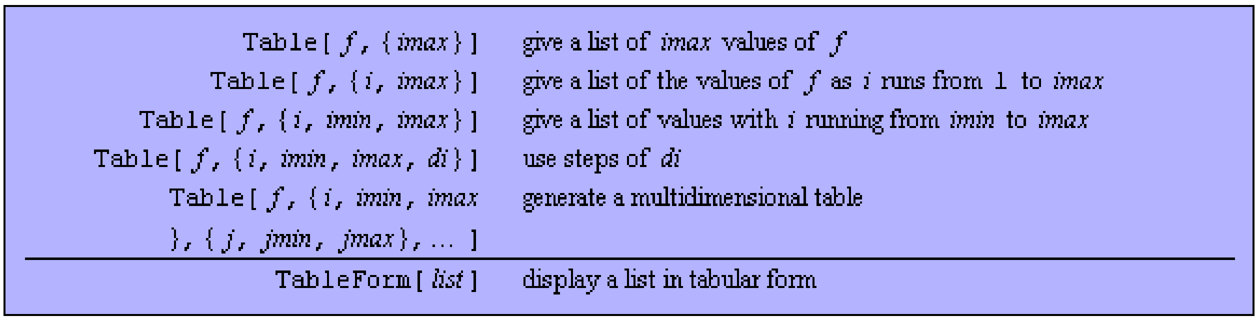
```
1  1
1  2
1  3
1  4
2  1
2  2
2  3
2  4
3  1
3  2
3  3
3  4
```

Here is a xyt-table

nmove=2; npart=3;

xyt = Table[{x[[j]],y[[j]],i},{i,nmove},{j,npart}]

```
{{{x[[1]],y[[1]],1},{x[[2]],y[[2]],1},{x[[3]],y[[3]],1}},

{{x[[1]],y[[1]],2},{x[[2]],y[[2]],2},{x[[3]],y[[3]],2}}}
```



Table[*f*, { *imax* }]	give a list of *imax* values of *f*
Table[*f*, { *i*, *imax* }]	give a list of the values of *f* as *i* runs from 1 to *imax*
Table[*f*, { *i*, *imin*, *imax* }]	give a list of values with *i* running from *imin* to *imax*
Table[*f*, { *i*, *imin*, *imax*, *di* }]	use steps of *di*
Table[*f*, { *i*, *imin*, *imax* }, { *j*, *jmin*, *jmax* }, …]	generate a multidimensional table
TableForm[*list*]	display a list in tabular form

12.3.5 For, if, goto, label

For [start,test,incr,body] executes start, then repeatedly evaluates body and incr until test fails to give True

For[i=0,i<=3,i++,Print[i]]

```
0
1
2
3
```

If [condition,t] gives t if condition evaluates to true

ipart=5;

j=4;

If [ipart != j, Print[ipart]]

```
5
```

Note that "$a \neq b$" should be entered as "!=" and "$a \leq b$" as "<=".
Goto[tag] scans for Label [tag], and transfers control to that Point

npart=5;

For[i=1,i ≤npart,i++,

If[i>3,Goto[end]];

Print["i smaller than npart"]; Print [i]; Print[npart];

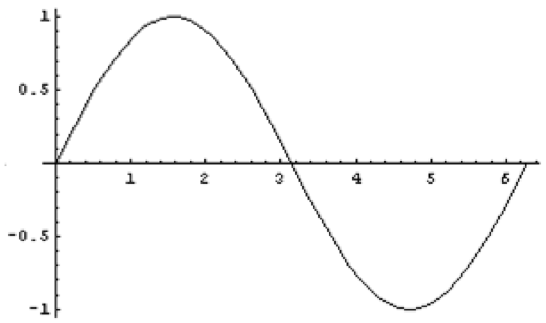
Label[end];]

```
i smaller than npart
1
5
i smaller than npart
2
5
i smaller than npart
3
5
```

12.3.6 Basic Plotting

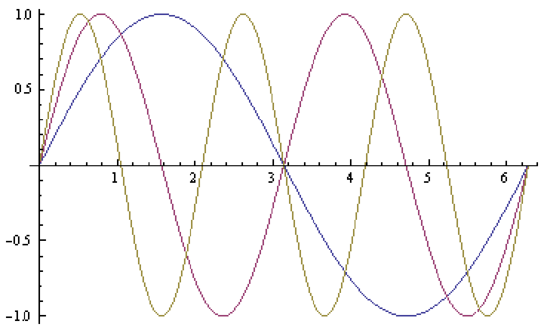
To plot a graph of sin(x) as a function of x from 0 to 2π, use

Plot[Sin[x], {x, 0, 2Pi}]

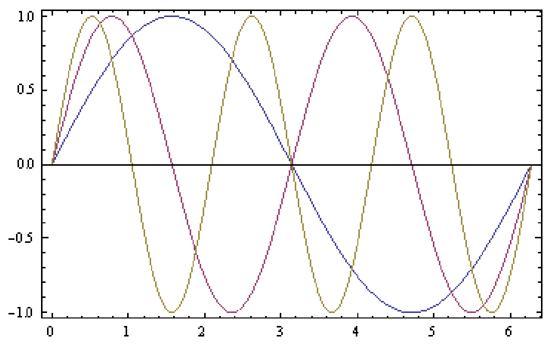


You can give a list of functions to plot.

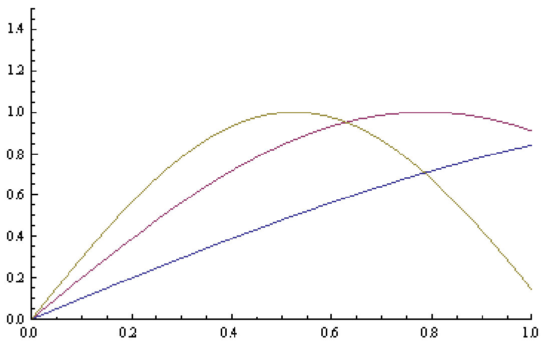
Plot[{Sin[x], Sin[2x], Sin[3x]}, {x, 0, 2Pi}]



Plot[{Sin[x], Sin[2x], Sin[3x]}, {x, 0, 2Pi}, Frame –>True]

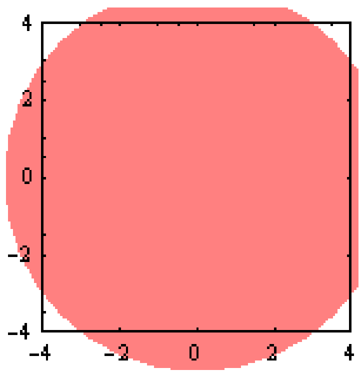


Plot[{Sin[x],Sin[2 x],Sin[3 x]},{x,0,2 Pi}, PlotRange –>{{0,1},{0,1.5}}]

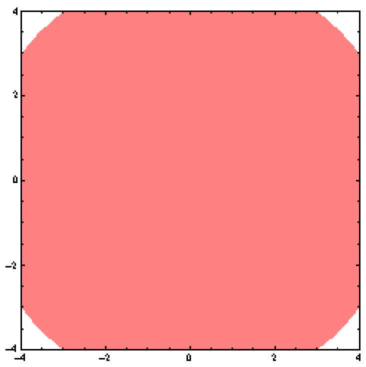


Graphics[{Pink,Disk[{0,0},5]}] plots a disk in the color pink at position {0,0} with a radius of 5. PlotRangeClipping allows graphics objects to spread beyond PlotRange:

Graphics[{Pink,Disk[{0,0},5]},Frame –>True,PlotRange –>4, PlotRangeClipping –>False]

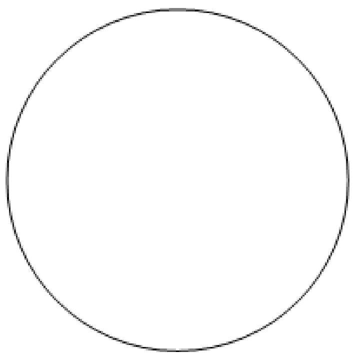


Graphics[{Pink,Disk[{0,0},5]},Frame –>True,PlotRange –>4, PlotRangeClipping –>True]

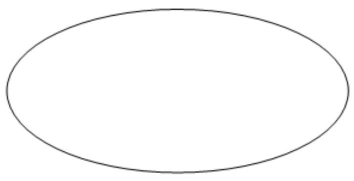


AspectRatio uses numerical values to specify the height-to-width ratio:

Graphics[Circle[ ],AspectRatio –>Automatic]

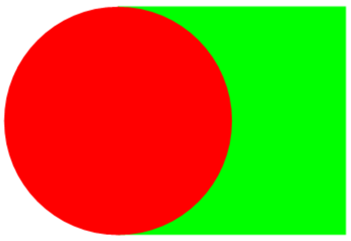


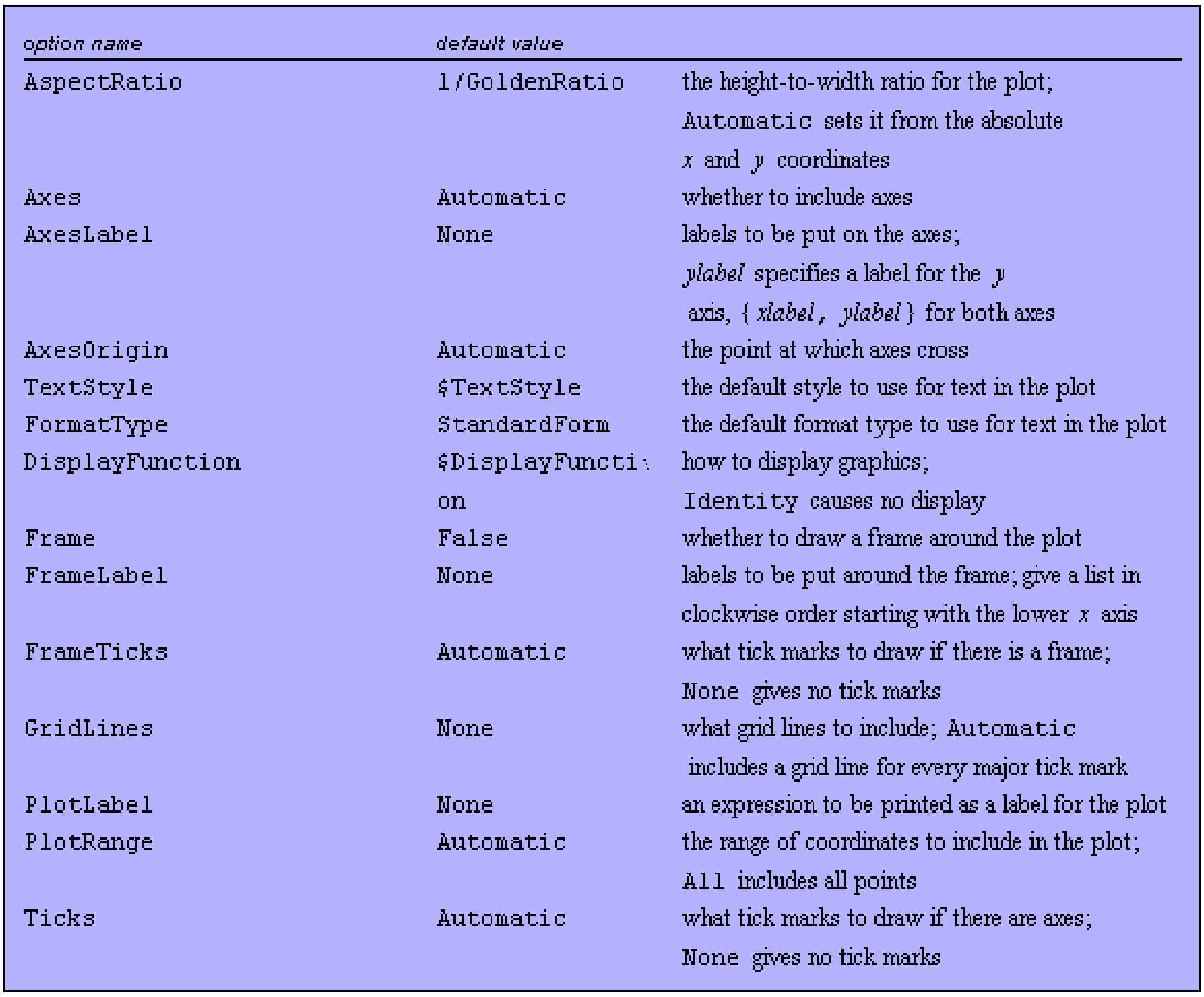
Graphics[Circle[ ],AspectRatio –>0.5]



Graphics[primitives,options] represents a two-dimensional graphical image.

Graphics[{Green,Rectangle[{0,-1},{2,1}],Red,Disk[]}]}





option name	default value	
`AspectRatio`	`1/GoldenRatio`	the height-to-width ratio for the plot; `Automatic` sets it from the absolute *x* and *y* coordinates
`Axes`	`Automatic`	whether to include axes
`AxesLabel`	`None`	labels to be put on the axes; *ylabel* specifies a label for the *y* axis, { *xlabel*, *ylabel* } for both axes
`AxesOrigin`	`Automatic`	the point at which axes cross
`TextStyle`	`$TextStyle`	the default style to use for text in the plot
`FormatType`	`StandardForm`	the default format type to use for text in the plot
`DisplayFunction`	`$DisplayFunction`	how to display graphics; `Identity` causes no display
`Frame`	`False`	whether to draw a frame around the plot
`FrameLabel`	`None`	labels to be put around the frame; give a list in clockwise order starting with the lower *x* axis
`FrameTicks`	`Automatic`	what tick marks to draw if there is a frame; `None` gives no tick marks
`GridLines`	`None`	what grid lines to include; `Automatic` includes a grid line for every major tick mark
`PlotLabel`	`None`	an expression to be printed as a label for the plot
`PlotRange`	`Automatic`	the range of coordinates to include in the plot; `All` includes all points
`Ticks`	`Automatic`	what tick marks to draw if there are axes; `None` gives no tick marks

Bibliography

[1] Metropolis, N.; Rosenbluth, A.W.; Rosenbluth, M.N.; Teller, A.N.; Teller, E. *J. Chem. Phys.* 1953, *21*, 1087–1092.

[2] Manousiouthakis, V.I.; Deem, M.W. *J. Chem. Phys.* 1999, *110*, 2753–2756.

[3] Frenkel, D.; Smit, B., *Understanding Molecular Simulation: from Algorithms to Applications*, 2nd ed. Academic Press; San Diego, 2002.

[4] Panagiotopoulos, A.Z. *Mol. Phys.* 1987, *61*, 813–826.

[5] Panagiotopoulos, A.Z. *Mol. Phys.* 1987, *62*, 701.

[6] Polson, J.M.; Trizac, E.; Pronk, S.; Frenkel, D. *J. Chem. Phys.* 2000, *112*, 5339–5342.