Anders Malthe-Sorenssen*

Elementary Thermal Physics Using Python

Aug 30, 2015

Springer

*Department of Physics, University of Oslo, Norway.

Contents

Chapter 1
Introduction

Thermal physics is the physics of macroscopic objects — objects that consist of many individual elements such as atoms, molecules or large compounds. In thermal physics we bring the tools from other parts of physics together and introduce entirely new concepts needed to address fundemental and practical problems. We combine mechanics, electromagnetics and quantuum mechanics to describe processes on macroscopic scales. Thermal physics is needed to address many of the large scale phenomena seen around you from the scales of nanometers to lightyears. Thermal physics is about many of the questions we wonder about: How does you blood carry oxygen? How does an ionic pump in a cell in your body work? What determines chemical equilibria? Why is ice frozen while water flows? How can we desalinate water, generate energy, conserve energy, or understand the biological and geological processes responsible for the world we live in? Thermal physics is also about limits: How much energy can we get from heat? How does the universe evolve in the really long run? Thermal physics is about fundamental questions: What are the processes responsible for the complex patterns and structures in the Earth's crust? Why do many processes only spontaneously occur in one direction in time? Can we build a machine that generates energy from nothing? Why is more different [**?**]?

Thermal and statistical physics are about the wonders of the world, but as with all physics, we need tools to address and understand the wonders. However, as you develop these tools and improve your knowledge and insights into thermal and statistical physics, you will see that these tools allow you to pose further questions and gain further insights. But they also allow you to be more precise with both your observations of nature and in your ability to predict and utlize nature. Thermal physics contains many of the elements needed to innovate to address some of the key questions that face us, and the Earth, now: How can we generate, store and distribute energy efficiently and in environmentally sustainable ways? How can we generate clean water and air? How can we generate energy efficient homes, cities and societies?

Thermal physics is the physics of systems of many. It may therefore be tempting to ask what the limits of thermal physics are. What happens when we get better at engineering systems that are very, very small. Can we then escape from some of the

limitations of statistical physics? Is the physics of nano-scale systems fundamentally different from that of macroscopic systems? Is there a need for new physical concepts on the meso-scale, on the scale between individual atoms and molecules and the thermal physics scale of many atoms? These are questions and concepts that are still under development, as we develop better computational and experimental tools to address systems with thousands to billions of atoms.

Many of the key concepts of thermal and statistical physics can also be applied in completely different fields, but this should be done with care and caution. If you have a background in statistics you may indeed recoqnize many of the concepts introduced in this text, such as the strong similarity between the partition function and the likelihood function. Indeed, concepts from statistical physics, and particular algorithmic modeling approaches, are being applied across fields, n areas such as life science, epidemology, economics, sociology, finance, and marketing. This illustrates that the concepts you learn in statistical physics are useful and powerful, and that many of their applications are yet to be discovered.

The goal of this text is to provide you with a first introduction to key elements of thermal and statistical physics for students with no background in thermodynamics or statistical physics. The approach of this book is to start from the microscopic and develop the thermodynamic concepts from there. This means that we will start from an atomic picture, similar to what you have already learned to master in your basic courses in mechanics and quantuum mechanics and move slowly towards thermodynamics. I believe this coupling between the microscopic and the macroscopic to be conceptually important, indeed one of the great achievements of physics, but it is fully possible to apply thermodynamic principles without understanding the underlying microscopic foundations. When we have motivated thermodynamics from statistical physics, we will continue to apply and use thermodynamic concepts to address processes without always reverting to a microscopic picture. If you are primarily interested in the applictions, you will therefore have to bear with me through the first, conceptual introduction.

1.0.1 Thermodynamics

Thermal and statistical physics is the study of macroscopic objects with the aim of understanding and predicting their behavior from the microscopic interactions between the particles making up the system. In our study of thermal physics we will discover new laws that extend beyond the laws of energy conservation or Newton's laws that we already know from mechanics, but these new laws are only valid for systems with many particles. A wonderful result in statistical physics is that many collective properties of macroscopic systems do not depend on the microscopic details. For example, all liquids have common properties that do not depend on the details of the interatomic interactions of the atoms or molecules making up the liquid. The behavior is the same if the liquid is made of metal, glass, water, polymers (macromolecules) or liquid hydrogen! There are common properties of matter that

simply depends on the statistical behavior of systems of many particles, and thermal and statistical physics is the study of these properties. *Thermal physics* and *thermodynamics* contain the laws describing macroscopic objects and *statistical physics* is the theoretical framework that allows us to derive the laws of thermodynamics from our understanding of the underlying microscopic interactions.

In thermodynamics we introduce new physical concepts such as temperature, thermal energy (heat), and heat capacity. We introduce the concepts needed to discuss thermodynamic processes and thermodynamic equilibrium. We introduce the basic notions needed to characterize the equilibrium chemical processes such as free energies and chemical potentials and use these concepts also to discuss phase transitions and mixtures, including solutions.

Macroscopic objects have their own laws and behaviors which we must understand, learn, and preferrably develop a theoretical foundation for. This is the aim of *thermodynamics*. Thermodynamics can be developed as an axiomatic system, based on a few basic laws, but it can also be developed from a microscopic model of a system — using statistical mechanics.

1.0.2 The great success of statistical mechanics

Statistical mechanics is the theory that allows us to demonstrate the validity of thermodynamics and allows us to calculate many of the fundamental thermodynamic properties of a system. For example, we will later show how we can find the ideal gas law from the quantuum mechanical behavior of a particle in a box. Indeed, statistical mechanics is a great success in physics from both a conceptual and a practical point of view. It gives us insight into the laws of thermodynamics, since we can derive them, and it gives us tools to calculate real material properties from first principles. Statistical mechanics also gives insight into why some of the microscopic details are exactly that — details that do not matter — and why some details are important. In this text you, we start from a microscopic picture and derive the laws of thermodynamics before we apply and extend them.

1.0.3 Integrated numerical methods

This text grew from a need to have a text that integrates the use of numerical methods into the exposition, examples and exercises of a course in thermal physics. Computations have always been at the center of physics. It can indeed be argued that Newton invented calculus in order to calculate motion. But with the advent of computers and with their rapid growth in computational power, computating is now an integrated part of all aspects of physics, and it is also central to how physical is developed and applied in research and industry. However, in most textbooks, only analytical tools are introduced and taught. In this text, I have attempted to fully integrate the use

of computational tools. This provides us with some advantages, since we are not limited to problems that can be solved analytically, but it does add new challenges as new computational methods need to be learned and mastered.

Integration of computational methods allows us to develop examples and exercises that more closely follow a work-flow similar to that used in research and industry, and it allows us to use realistic data for analysis. It will also teach you computational skills that we expect to be useful later.

The basic philosophy of this text is that the students should implement and run simulations themselves instead of using packages or applets that can be tweaked. We have found that this provides students with useful tools and insight, but also that it is very fun and inspiring to be able to generate realistic physical simulations yourself and to visualize and understand your own data. Again, this comes at a cost, since you will not be able to develop as advanced code as a commercial software package can provide, but we believe that the satisfaction you gain by implementing it yourself is more important and enjoyable.

We have also integrated the use of symbolic calculations in Python in both the exposition and the examples. This is to introduce you to a useful tool in its natural context — this is how physicits and engineers actually solve problems. We use the symbolic toolbox in Python in order to keek everything within a single programming framework, even if we know that for example Mathematica provides a better set of tools for many symbolic problems.

This text is part of a series of texts, where each text introduces key computational concepts relevant for the particular type of physics introduced in the text. For thermal and statistical physics, we will introduce new concepts in algorithmic and stochastic modelling, molecular dynamics modeling, statistical descriptions of large data sets, optimization methods needs for minimization of free energies, and continuum modeling methods needed for kinetic theory and transport.

1.0.4 Molecular dynamics modeling

A particular focus of this text, will be the introduction of molecular dynamics (MD) modeling. Molecular dynamics is the integration of the trajectories of many interacting atoms forward in time. There are now many commercial and open source tools that allow you to use such simulations to measure thermodynamic properties of many types of systems, and molecular dynamics may therefore be a tool that is practically useful for you in your future professional career. In addition, it is a tool that allows us to observe the collective effects and to measure the statistical properties of systems of many atoms, and hence molecular dynamics also provide a powerful method to gain intuition into the concepts in statistical and thermal physics. Molecular dynamics is also a tool that is used to understand basic atomic and molecular processes in chemistry, material science, and life science. With the advent of ever more powerful techniques that combine quantuum mechanical calculations with molecular dynamics simulations of many atoms, simulations tools are

starting to become real predicitive tools that can be used for calculations of material properties, rates of chemical processes, or the interaction of chemical compounds with biologically active molecules or cell machinery. Indeed, molecular dynamics simulations is now a routine part of research and development in areas such as the development of novel materials, catalysis in chemical processing or in pharmaceutical research.

We have chosen to use the open source tool LAMMPS [**?**] to integrate the motion of atoms. However, our focus is on the primary principles and on the analysis of data from simulations in order to gain a firm understanding of the key concepts. It is therefore simple to replace this with another tool of your choice.

1.0.5 Learning Physics

The typical advice you get on how to learn physics is usually based on solid, pedagogical research. You learn physics by doing physics, by trying to apply the concepts introduced in new situations. This is why most textbooks in physics contains many worked examples and exercises: You learn best by first reading the exposition, working your way through a worked example, then work on the shorted exercises before you start working on larger projects. This book therefore contains these three elements:

Worked examples: Worked examples are used to motivate the theory, and are then integrated into the text, or are provided as examples as to how to solve an exercise or a problem. They typically contain both analytical, symbolic and computational methods.

Exercises: Short problems that adress a key concept.

Projects: Long problems that combine all the key concepts introduced in a chapter into a coherent story. These problems will typically contain both analytical, symbolic, and computational approaches and various levels of refinement of a theory.

1.0.6 Prerequisites

An introductory course in thermal and statistical physics typically follows courses in mechanics, electromagnetism, and quantuum physics. However, this text does not require any significant knowledge of electromagnetism — the few concepts needed, such as for electromagnetic waves, will be introduced when needed. Basic concept from mechanics such as Newton's laws, mechanical energy and work is used throughout the text.

A few key concepts from quantuum mechanics is needed, but we will not need to solve the Schrodinger equation. The concepts needed from quantuum mechanics is

the quantization of energy states for a particle in a box (used for an ideal gas), for a two-state system, and for an harmonic oscillator. In order to calculate the rotational behavior of a diatomic gas, you also need to apply the formula for the rotational energy states. However, in all these cases no derivations are needed, we only apply a given formula for the quantization of states. In addition, we need the concepts of Fermions and Bosons to describe Fermi and Bose gases, but the only concept we use is that two Fermions cannot be in the same quanuum mechanical state.

It is not necessary to have an introductory programming course in order to follow this text, but it may be an advantage to have some experience in scripting in either Python or Matlab. We have, however, provided a quick introduction to programming in Appendix **??**.

1.0.7 Structure of this book

1.0.8 Key questions

In our studies of thermal and statistical physics, we will ask and answer many fundamental as well as practical questions:

- What is temperature?
- Can a machine run forever?
- Why does a movie look strange when played backwards?
- How efficient can an engine or a heat-exchanger work?
- Can we generate electricity by lowering the temperature of the sea?
- How much energy does it take to boil a liter of water?
- Why does water droplets form on the outside of may cold glass of water?
- Why do clouds form?

These and many other questions will be addressed in this text. Stay with us!

Chapter 2
The Arrow of Time

Abstract In this chapter we introduce the notion of the arrow in time and provide the first insights into the difference between a system with few and many particle. We introduce heterogeity in the distribution of energy in simple mechanical models, such as a bouncing ball and in a monatomic gas. For the monatomic gas, we introduce two modelling approaches that will be used frequently throughout the book: Molecular dynamics modeling and algorithmic modeling. We show how a system starting outside equilibrium approaches equilibrium irreversibly, relating the behavior of many atoms to the arrow of time.

Thermal and statistical physics is about the behavior of systems of many particles. But in what way is a system of many particles different from that of a few particles. Mainly in the number of particles! There are concepts and effects that only make sense in systems with many particles. Let us now try to gain intuition into what the effects of many particles are.

2.1 One and many particles

Let us start exploring from a simple macroscopic system consisting of either a single particle or subdivided into many particlesd. A system you are well aquinted with from mechanics is that of a bouncing ball. Does the arrow of time appear in the dynamics of a bouncing ball?

2.1.1 Reversible bouncing of a solid ball

Fig. 2.1 illustrates a classical problem in mechanics — a single ball bouncing on an elastic floor. The ball is affected by two potentials: Gravity $U_G = mgz$, and a linear spring modeling the interaction with the floor, $U_F = -(1/2)kz^2$ when $z < 0$ and 0 when $z > 0$. The figure illustrates the motion of the center of mass of this

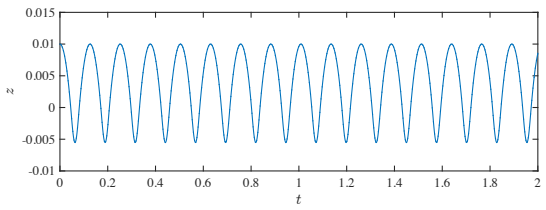

Fig. 2.1 The motion of a single ball bouncing on an elastic floor shown in a plot of $z(t)$.

ball. This curve is completely symmetric in time — if we reverse time, $t \to -t$, the curve would look the same. You would not be able to see if we play a movie of the ball backward or forward. This is because the motion of this individual particle is reversible in time. You can see this directly from Newton's second law:

$$m\frac{d^2\mathbf{r}}{dt^2} = -\nabla U(\mathbf{r}) = m\frac{d^2\mathbf{r}}{d(-t)^2}\,. \tag{2.1}$$

We get the same equation if we replace t with $-t$. Indeed, this is also obvious from the conservation of energy, which gives

$$\frac{1}{2}mv^2 = U(\mathbf{r}) \;\Rightarrow v = \pm\sqrt{2U(\mathbf{r})/m}\,, \tag{2.2}$$

where there are two possible signs corresponding to two possible directions of movement.

For a single particle (affected by conservative forces) motion is *reversible in time*, and it would not look unphysical if we played a movie of the motion backward. This will always be the case for the systems we are looking at here — the motion of individual particles will always be conservative and reversible. Irreversibility comes from many particles. But how?

2.1.2 Irreversible bouncing of an elastic ball

Now, let us study our first system with many particles: Instead of modeling a solid ball, we study a ball that deforms elastically consisting of a line of masses connected with springs as shown in Fig. 2.2. Only the bottom-most mass interacts with the floor, just like before, and the overall mass and the initial height is the same as before. What happens if we drop this ball? Fig. 2.2 shows the position of the center of mass, $z_{\rm cm}(t)$ as a function of time. Hmmm. This does not look the same as in Fig. 2.1. The ball does not bounce back up to its original height, but bounces around at a lower height. What has happened?

The total energy in the system has not changed. The motion is conservative and the change in energy is negligible. However, we can divide the energy into two parts, the energy of the center of mass and the internal energy in the form of motion relative

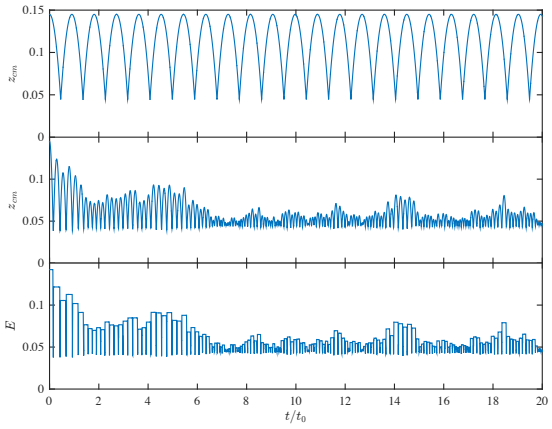

Fig. 2.2 The motion of a single ball bouncing on an elastic floor shown in a plot of $z(t)$.

the center of mass and of internal compression or elongation of the springs. Fig. 2.2 shows the center-of-mass energy, $E_{\rm cm}$ and the internal energy $\Delta E = E_{\rm TOT} - {\rm e}_{\rm cm}$ as a function of time. We see that initially all the energy is the in the center of mass motion, but after a short while, energy is distributed between the center of mass motion and the internal energy.

The initial point therefore seems to be a special point, whereas configurations further on seems more representative. We are not surprised as the system bounces a bit up and down as time progresses, but we would be very surprised if the system suddenly bounced so that all the energy ended up in the center of mass motion, and no energy was in the internal motion, corresponding to what would happen if we played a movie of the motion backwards. Notice that it is not physically impossible to go backwards. Indeed, if we reversed all the velocities after a short time, the system would return to its initial state where all the energy is in the center of mass motion.

It seems very *unlikely* that the system would move to this special state, corresponding to the initial state. *Improbable*, but not *impossible*. And we are now at the core of the arrow of time, motion forward in time is not the only possibility, but the most probable possibility. And we see that the system develops towards a state where the energy is distributed throughout all the possible degrees of freedom in the system, and that this state corresponds to an *equilibrium*. Still, all these words are vague and without a precise meaning, but we can see the initial contours of the difference between one particle and many particles. In systems with many particles there are many ways to distribute the energy, and the more probable states corresponds to a homogeneous distribution of energy.

We are now onto the new laws of physics of many particles. The are related to likelihoods — to probabilities — and they are therefore different than the law's of Newton, which are absolute laws. We will start to address the differences in a simple atomic system — an atomic gas.

Finding the motion of an elastic object

We model a one-dimensional elastically deforming ball as a set of masses, m, connected by springs of spring constant k. The springs are enumerated $i = 1,\ldots,N$ from the bottom to the top. The bottom-most mass also interacts with the floor. The force on particle i is then

$$m_i a_i = F_i = -m_i g - k\left(z_{i+1} - z_i - b\right) + k\left(z_i - z_{i-1}\right) - F_w \,, \tag{2.3}$$

where $F_w = -k_w z_i$ when for $i = 1$ when $z_i < 0$ and 0 otherwise. The motion is solved using the following program:

```
from pylab import *
N = 10 # Nr of elements
k = 100000
kw = 1000000 # Spring constants
d = 0.1  # Size in m
m = 0.1  # Mass in kg
t = 20.0 # Simulation time
g = 9.81 # Acc of gravity
z0 = 0.1 # Initial height
dt = 0.00001 # Timestep
nstep = ceil(t/dt)
z = zeros((N,nstep),float)
vz = zeros((N,nstep),float)
z[:,0] = d*(array(range(N)))/N+z0 # Initial positions
l = d/N # Distance between nodes
for i in range(nstep): # Calculate motion
    dz = diff(z[:,i])-l
    F = -k*append(0.0,dz) + k*append(dz,0.0) - m*g # Internode forces
    F[0] = F[0] - kw*z[0,i]*(z[0,i]<0) # Bottom wall
    a = F/m
    vz[:,i+1] = vz[:,i] + a*dt
    z[:,i+1] = z[:,i] + vz[:,i+1]*dt;
```

Notice the use of vectorized math to find all the differences $z_{i+1} - z_i$ at once using 'diff'. The resulting motion is shown in Fig. 2.2.

2.1.3 More degrees of freedom

The behavior of many particles becomes even clearer in a system with more degrees of freedom, as illustrated in Fig. 2.3, where we have simulated a two-dimensional ball bouncing against a flat surface. The ball bounces lower and lower as time progresses, but the fluctuations are now smaller, probably because we have much more particles to distribute the energy amonst. If we played the motion in this system backwards — if we reversed time — it would look strange and unphysical.

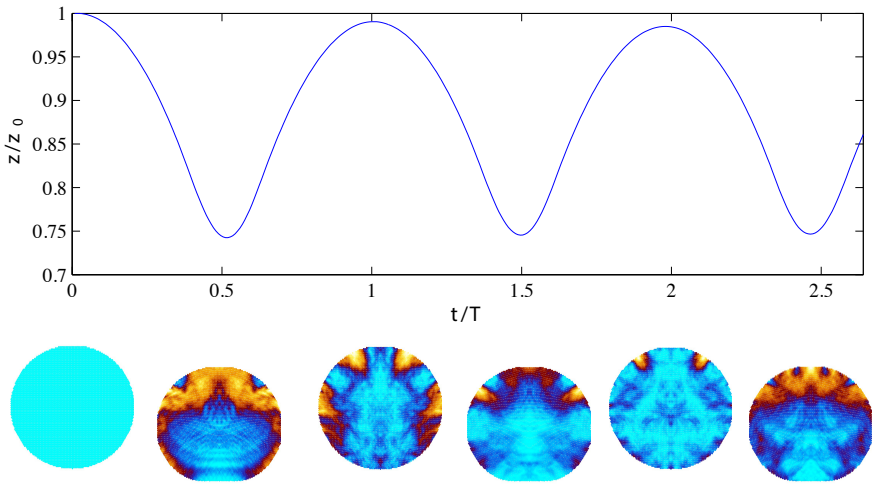

Fig. 2.3 Illustration of a computer simulation of a bouncing ball. Energy is conserved in the simulation, yet the ball does not bounce as high in its second bounce as its starting point. The color scale indicates the magnitude of the velocity. We see that initially, the whole ball has the same velocity, but after the ball has hit the ground, the distribution of velocities is heterogeneous.

Energy is conserved in this system for both forward and backward motion. This means that energy conservation is not the only law that is important for macroscopic behavior. We need more laws! The new laws are related to how the energy is distributed in the object. From Fig. 2.3 we see that in the first few moments as the ball falls down, all the energy is in one particular degree of freedom, the motion of the center of mass of the ball. But as the ball bounces, the energy is distributed among many degrees of freedom. We will learn more about how we describe the distribution of energy between different degrees of freedom throughout this book, through the introduction of the concept of *entropy*. However, first we simply need to acknowledge that energy conservation is not enough to describe the arrow of time. We need something more!

2.2 Approach to equilibrium — molecular dynamics

Let us see how much insight we can gain from the atomic hypothesis and the motion of all the atoms in a system. Given the precise motion of all the atoms, we should be able to understand the behavior of the macroscopic system. However, as we shall see, it turns out not to be that simple. Description or direct calculation does not always provide insights by themselves. But we may be able to build understanding based on description and calculation.

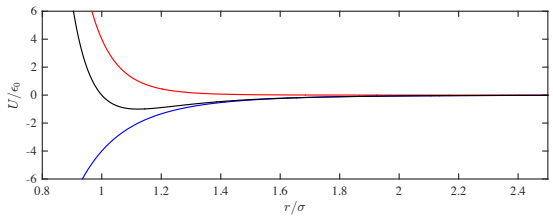

Fig. 2.4 Plot of the Lennard-Jones potential showing the repulsive part (*blue*), the attractive van-der-Waals interaction (*red*) and the combined Lennard-Jones poential (*black*).

Here, we use a molecular dynamics simulation program to find the motion of a set of atoms and then analyze the motion. You can read more about these tools in chapter (3).

How do we determine the time development of a system of atoms? This is a problem in quantum mechanics: The motion and interaction of atoms, including the full motion of the electrons, needs to be determined from the basic equations of quantum mechanics. However, it turns out that we can learn very much without a quantuum mechanical approach. The atomic hypothesis in itself combined with Newtonian mechanics and a simplified description of the interactions between atoms provide valuable insights as well as good predictions of the behavior of macroscopic systems.

How can we find the interactions between atoms? Again, we may use quantuum mechanics to find the effective potential energy for a system of many atoms. Here, we will start from a very simple model for atom-atom interactions: We will assume that the energy only only depends on the distance between pairs of atoms.

One of the simplest models for atom-atom interactions comes from a description of the interactions between nobel-gas atoms, but is widely used to model many types of atomic interactions. There are two main contributions to the interaction between two noble-gas atoms:

- There is an attractive force due to a dipole-dipole interaction. The potential for this interaction is proportional to $(1/r)^6$, where r is the distance between the atoms. This interaction is called the *van der Waals* interaction and we call the corresponding force the *van der Waals* force.
- There is a repulsive force which is a QM effect due to the possibility of overlapping electron orbitals as the two atoms are pushed together. We use a power-law of the form $(1/r)^n$ to represent this interaction. It is common to choose $n = 12$, which gives a good approximation to the behavior of Argon.

The combined model is called the Lennard-Jones potential (see Fig. 2.4):

$$U(r) = 4\varepsilon \left(\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right) . \tag{2.4}$$

The force is given as the derivative of the potential.

We will use this potential as a basic tool to understand the behavior of macroscopic objects. To solve the equations of motion we will use molecular dynamics simulation package called LAMMPS, which is an efficient package for integrating

the equations of motion of the system. (Learn more about molecular dynamics and how to set up and analyze simulations in Chap. 3).

2.2.1 Approach to equilibrium

Let us address how a gas approaches equilibrium by starting the system outside equilibrium and studying the approach to equilibrium in detail.

We want to model a gas of atoms: a system where the atoms are rather far apart from each other. (We will be more specific later). We start with a box for the gas and divide the box into two equal parts. We fill one part of the box with atoms and keep the other part empty. The atoms start with random velocities. (But with zero total momentum). From $t = 0$ we allow the system to develop in time.

We run this simulation for 2000 timesteps in LAMMPS using

```
lammps < in.gastwosection01.lmp
```

where the input file is in.gastwosection01.lmp[1]

```
# 2d Lennard-Jones gas
units lj
dimension 2
atom_style atomic
lattice hex 0.25
region mybox block 0 20 0 10 -0.5 0.5
create_box 1 mybox
region 2 block 0 10 0 10 -0.5 0.05
create_atoms 1 region 2
mass 1 1.0
velocity all create 0.5 87287
pair_style lj/cut 2.5
pair_coeff 1 1 1.0 1.0 2.5
neighbor 0.3 bin
neigh_modify every 20 delay 0 check no
fix 1 all nve
dump 1 all custom 10 twosec01.lammpstrj id type x y z vx vy vz
restart 2000 mydata.restart01 mydata.restart011
thermo 100
run 2000
```

The initial setup is shown in Fig. 2.5a. After 2000 timesteps (Fig. 2.5b) the system has expanded to fill almost the whole area, but we can still see that there are more atoms on the left side and the atoms still appear to be lumped together inhomogeneously. The resulting dynamics is shown in the movie of the simulation in gastwosection01.mp4[2].

What if we ran the simulation backwards in time, starting from the state after 2000 timestep? The resulting behavior can be seen by running the simulator

[1] http://folk.uio.no/malthe/fys2160/in.gastwosection01.lmp

[2] http://folk.uio.no/malthe/fys2160/gastwosection01.mp4

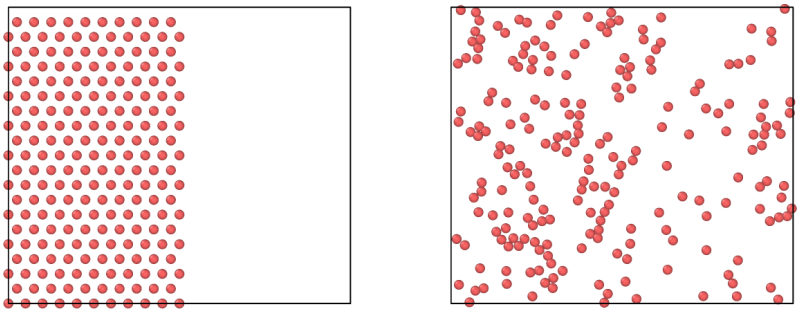

Fig. 2.5 Snap-shot from VMD of initial configuration of simulation of gas system (*left*) and after 2000 timesteps (*right*).

using in.gastwosection11.lmp[3], which results in the movie shown in gastwosection11.mp4[4].

```
# 2d Lennard-Jones gas
read_restart mydata.restart01
# Reverse all velocities
variable vx atom -vx
variable vy atom -vy
velocity all set v_vx v_vy NULL
fix all nve
dump 1 all custom 10 twosec11.lammpstrj id type x y z vx vy vz
restart 2000 mydata.restart11 mydata.restart111
thermo 100 # Output thermodyn variables every 100 timesteps
run 2000 # Number of timesteps
```

It *almost* gets back to where it started, with some small discrepancies due to numerical errors. The resulting behavior seen in the movie is clearly very different from the behavior we would expect. Indeed the behavior seems unphysical: We would not expect the atoms suddenly to organize themselves so that they all are on one side. Instead, we would expect the atoms to continue to be well mixed, and this is also what we would see if we continued to run the simulation forward in time. If we run the simulation for 10000 timesteps starting from the ordered state at $t = 0$, as is done by the input file gastwosection02.lmps[5] and visualized in the movie gastwosection02.mp4[6], we do not observe that the particles organize themselves regularly. Instead, we simply see that the particles mix homogeneously, and remain mixed while they move randomly about. Somehow, this mixed state seems more probable than the segregated state we started at, and the development in the movie in

[3] http://folk.uio.no/malthe/fys2160/in.gastwosection11.lmp
[4] http://folk.uio.no/malthe/fys2160/gastwosection11.mp4
[5] http://folk.uio.no/malthe/fys2160/gastwosection02.lmps
[6] http://folk.uio.no/malthe/fys2160/gastwosection02.mp4

gastwosection02.mp4[7] appears to be physically reasonable. At least it is not clearly unphysical, as for the segregating system.

How can we quantify this behavior? We can measure the number of particles (or the relative number of particles) on the left and the right half of the system as a function of time. Let us use $n(t)$ for the number of atoms on the left side. Since the number of atoms is conserved and equal to the total number of atoms N, the number of atoms on the right side is $N - n(t)$. It is therefore enough to characterize the behavior of the number of atoms on the left side of the system.

How can we measure this from the data? We can read the data from the simulation directly into Python using the `pizza.py` program and use `dump` to read and interpret the data from the files. You can learn more about how this works in Chap. 3. Here, I will simply use these routines to read data from the simulation.

To start python using `pizza.py` you type

```
python -i ~/pizza/src/pizza.py
```

if this is whereyou have downloaded/installed `pizza.py`. You should now have access to all the essential tools. We use the following script to measure the number of atoms on the left side of the system:

```
#start1
from pylab import *
d = dump("twosec02.lammpstrj") # Read output states
t = d.time()
nt = size(t)
nleft = zeros(n,float) # Store number of particles
# Get information about simulation box
tmp_time,box,atoms,bonds,tris,lines = d.viz(0)
halfsize = 0.5*box[3] # Box size in x-dir
for it in range(nt):
   xit = array(d.vecs(t[it],"x"))
   # Find list of all atoms in left half
   jj = find(xit<halfsize)
   nleft[it] = size(jj)
plot(t,nleft), xlabel('t'), ylabel('n'), show()
#end1
np.savetxt('tmp.d', (t,Npart[0,:]))
```

and the resulting plot that shows the number of particle on the left is shown in Fig. 2.6. Now, this is interesting. We see that the system clearly moves in a particular direction. But it is also clear that we need more data to be really sure about what is happening. Let us run the simulation for 50000 timesteps and replot the data. (Using in.gastwosection20.lmps[8] and nrleft02.py[9]). The resulting plot in Fig. 2.6 now much more convincingly shows what happens: We see that $n(t)$ converges towards an approximately constant value and then fluctuates around this value.

[7] `http://folk.uio.no/malthe/fys2160/gastwosection02.mp4`

[8] `http://folk.uio.no/malthe/fys2160/in.gastwosection20.lmps`

[9] `http://folk.uio.no/malthe/fys2160/nrleft02.py`

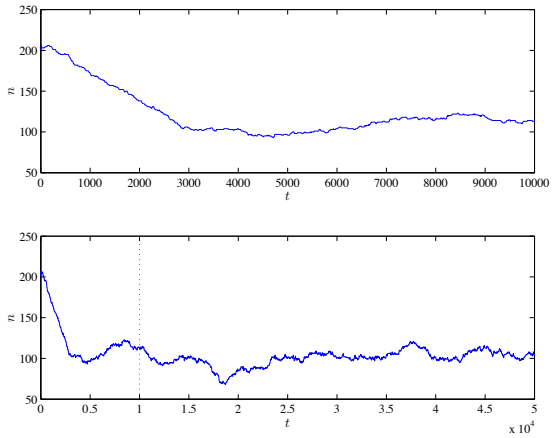

Fig. 2.6 Plot of the number of atoms on the left half (n) as a function of time (t) for 10000 timesteps (top) and for 50000 timesteps (bottom)

From this plot it is also clear that time is not fully reversible. It looks like we might move back and forth a bit in time, but if we go too far back we clearly end up in a situation that is different – all the atoms suddenly segrate. This seems unphysical. It is clearly not impossible – since it did indeed happen and would happen if we were able to run the system perfectly backwards in time. (You can achieve this numerically by having a sufficiently small time-step in your simulation). However, it seems to be unlikely, in the sense that even a tiny change in the positions or velocities, such as that generated by rounding errors, would mean that we would not be able to reverse the simulation. If you take the 50000 timestep simulation and reverse it, you will not get back to the segregated system. Instead, the system will fluctuate around the average value with no large excursion.

This arrow of time is central to our understanding of macrosopic systems. And it is clearly a new law — a law in addition to conservation of momentum or conservation of energy.

2.3 Approach to equilibrium — algorithmic model

The atomic model directly illustrates the arrow of time, and we can use molecular dynamics to directly measure the approach towards an equilibrium situation: From the trajectories of all the atoms we can measure a macroscopic quantity such as $n(t)$. But we can gain even more insight into the process by trying the extract the most important part of the process — the part of the process that really introduced the arrow of time — and leave out all the unnecessary details. We do this by introducing a simplified model that only contains the most important feature of the process — we introduce an *algorithmic model* of the process. This method of defining a model which is as simple as possible is a common method used in statistical physics, which we will return to many times throughout this book.

How would such as simplified, algorithmic look like? What is the most important features of the process? It is usually not easy to know what the most essential feature are, and often you would need to experiment and test many different model before you arrive at the simplest (or at least a very simple) model that reproduces the main effects. However, here we will simply pick the right model immediately, unfairly giving you the impression that this is simple in practice. It is not.

2.3.1 Model of atoms on the left side

Let us try to analyze the core features of the system: What is the system? It is a closed box with an imaginary wall in the middle. Initially, there are N atoms on one side of the wall. On the other side of the wall there are no atoms. As the atoms move about they get mixed, so that the distribution of atoms becomes more homogeneous after some time.

We could make this system more idealized by introducing an actual wall in the middle of the system. A wall with a small hole. This would also slow down the equilibration process, since it would take some time for the atoms to hit the hole and thus move to the other side to homogenize the distribution of atoms. However, we expect this system to retain many of the features of the original system: All the atoms starts on one side and as time progresses we expect the system to become more homogeneous. We could also model this system using molecular dynamics. But it would take a long time to find its behavior, since it would take a long time for an atom to actually hit the hole. We would spend a lot of time modeling atomic motion as the atoms bounced around inside the box, and we would spend little time modelling the actual process of interest, where an atom moves from one side of the box to the other. Instead, we should simplify the process and introduce an *algorithmic model* that only captures the most important part of the process — the motion of an atom from one box to another.

2.3.2 Simplifying the system — part 1

First, let us make a few simplifying assumptions about the system. Let us assume that all the atoms are independent. (This turns out to be a reasonable assumption and indeed the assumption we will use to define an ideal gas). With this assumption we get a new picture of what happens in the system. Each of the two boxes consists of many atoms that bounces around independenty. This means that each atom will have a given probability to go through the hole every time it hits the dividing wall. Another way to formulate this is that for each atom there is a certain probability per unit time that it will pass through the hole and move to the other side. Since the atoms are independent and identical, this probability must be the same for all the

atoms. Now we have transformed the problem into a rule-based model, which we can use to find the behavior of the system:

- The system consists of N indepedent atoms seperated by a wall with a small hole. The position of each atom is given by s_i, so that it atom i is in the left box, $s_i = 1$, and if it is in the right box $s_i = 0$.
- For each atom, there is a probability $p = R\Delta t$ that the atom will move through the hole from one box to the other during a time interval Δt.
- There are $n(t) = \sum_i s_i$ atoms on the left side and $N - n(t)$ on the right side.

This means that we can formulate a simple rule for how the system develops in time. At a time t, the state of the system is given by the positions $s_i(t)$ of all the N atoms. In the short time interval Δt from t to $t + \Delta t$, each atom i has a probability $p = R\Delta t$ to move from the box it currently is in and into the other box[10], that it, each atom has a probability p to change its state s_i into the opposite state.

2.3.3 Simplifying the system — part 2

However, this model is still more complicated than necessary, because for a small time step Δt, we will spend a lot of time not doing anything. Instead, we could divide time into time intervals Δt, which corresponds to the typical time it takes for an atom to pass through the hole. Then, the rule would be:

- At each time interval Δt, corresponding to the typical time it takes for an atom to pass through the hole, we choose an atom at random and move it to the other side.

This is a possible model, but it is *still* a bit too elaborate. We can make the model even simpler and more to the point, because we are not interested in where each atom is – we are only interested in how many atoms there are at each side. We are only interested in n: the number of atoms on the left side, and $N - n$, the number of atoms on the right side. If we pick an atom at random an move it to the other side, the probability that we pick an atom on the left side is n/N. (And the probability that we pick an atom on the right side is $(N - n)/N$.) The probability that we move an atom from the left to the right hand side is therefore n/N. We therefore have arrived at a simple algorithm:

- Draw a random number r between 0 and 1.
- If $r \leq n/N$ we move an atom from left to right ($n \to n - 1$), otherwise we move an atom from right to left ($n \to n + 1$).

[10] You may ask whether these probabilities always must be the same: Does the probability for an atom to move from the left to the right have to be the same as the probability to move from the right to the left? We will return to this question later. For now we will assume that the system is symmetric, and that there is no difference between the two sides.

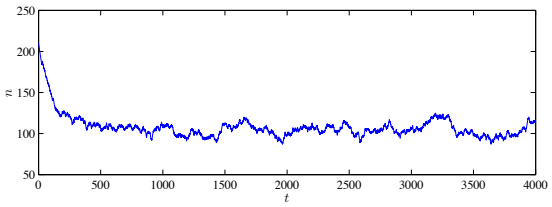

Fig. 2.7 Plot of the number of atoms on the left half (n) as a function of time (t) for 4000 timesteps of the discrete algorithm.

- Increase time by a unit Δt corresponding to the typical time it takes for an atom to pass through the hole.

We use this algorithm to study the approach towards equilibrium. The algorithm is implemented in the following program:

```
from pylab import *
N = 210 # Number of particles
nstep = 4000 # Number of steps
n = zeros(nstep)
n[0] = 210 # Initial number on left side
for i in range(1,nstep):
    r = rand(1)
    if (r<n[i-1]/N):
        n[i] = n[i-1] - 1 # Move atom from left to right
    else:
        n[i] = n[i-1] + 1 # Move atom from right to left
plot(range(0,nstep),n/N)
xlabel('t'),ylabel('n/N')
show()
```

If we run this program for $N = 210$, which corresponds to the number of atoms in the molecular dynamics simulations, we get the $n(t)$ curve shown in Fig. 2.7.

2.3.4 Fluctuations and the number of atoms, N

This simplified model allows us to study the approach to equilibrium and the fluctuations at equilibrium efficiently, for example by addressing the dependency on the number of atoms, N. What happens as we increase the number of atoms? Based on our discussions boave, we expect the system to become more irreversible, even if we still do not know what we precisely mean by that. Let us therefore look at the data. We implement a short script to plot the behavior for various values of N:

```
from pylab import *
N = array([16,32,64]) # Number of particles
nstep = 4000 # Number of steps
for j in range(0,size(N)):
    NN = N[j]
    n = zeros(nstep)
    n[0] = NN # Initial number on left side
```

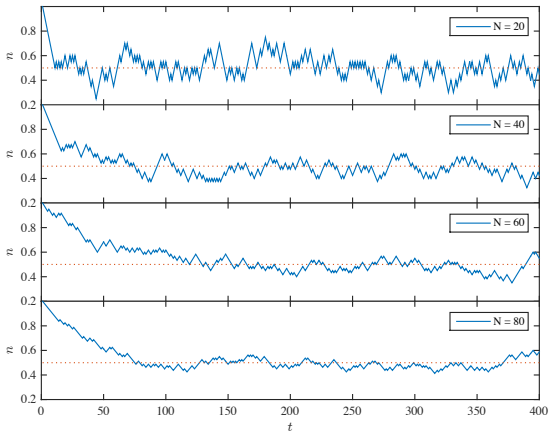

Fig. 2.8 Plot of $n(t)/N$ for various system sizes N.

```
    for i in range(1,nstep):
        r = rand(1)
        if (r<n[i-1]/NN):
            n[i] = n[i-1] - 1 # Move atom from left to right
        else:
            n[i] = n[i-1] + 1 # Move atom from right to left
    plot(range(0,nstep),n)
show()
```

The resulting behaviors for $N = 20, 40, 60$ and 80 are shown in Fig. 2.8. We see that the fluctuations becomes smaller when N increases. This means that the system stays closer to the equilibrium value for n — corresponding to the average n over long times. The probability for a large excursion becomes smaller when the number of particles N becomes larger. This also means that when the number of particles is very small, it is fully possible to have a fluctuation where all the particles are on the left side ($n = 1$), but as N becomes larger, this becomes more and more unlikely. This illustrates the principle of the physical law we are exploring here: A behavoir far from the average becomes less and less probable — not impossible, only improbable — as the number of atoms increase. The physical laws for macroscopic systems are therefore a consequence of the many number of atoms in a typical macroscopic system. In a liter of gas (at room temperature and atmospheric pressure) there are typicall 10^{23} gas molecules. A very large number. And it is the largeness of this number that ensures the regularity of the physical laws. With $N = 10^{23}$ the fluctuations becomes very, very small indeed.

We also observe a clear transition from the initial state at $t = 0$ to the equilibrium state where there are (small) fluctuations around an average value of n. And we see that the transition time becomes longer for a larger system. For now, this will only be an interesting observation, but we may return to such transient effect later on.

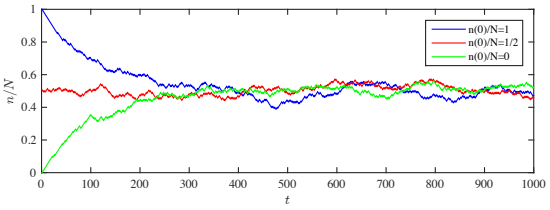

Fig. 2.9 Plot of $n(t)/N$ for various initial conditions $n(0)$: $n(0) = 1$ (*blue*), $n(0) = 1/2$ (*red*), $n(0) = 1/2$ (*green*).

2.3.5 *Average value of* n

What is the equilibrium value of n in this system? This is not a clear question, since we have not defined what *equilibrium* means. However, from our data in Fig. 2.8 it is clear that $n(t)$ approaches some value after a time, although it continues to fluctuate around this value. What is this value? We could find this value as the average value of n, after an initial transient time of 1000 steps, using the function `average`,

```
>> average(n[1000:])
0.50569523809523831
```

This is very close to $n(t)/N = 1/2$, which is what we would expect theoretically. Why? Because there is nothing special about the left hand side compared to the right hand side: They are equally large boxes and otherwise similar. We therefore expect the number of atoms on the left and on the right hand side to be equal, on average, and hence $n/N = (N-n)/N$, which gives $n/N = 1/2$. In practice, we are not going to measure exacly the same number of atoms on each side, since there are small fluctuations.

Test your understanding: What do you think will happen if we gradually increase N? Make a short program to plot the standard deviation of $n(t)$ as a function of N and comment on the results.

2.3.6 *Dependence on initial conditions*

Fig. 2.8 shows that the system approaches an equilibrium state after a short transition time, where the transition time appears to increase with the number of atoms, N, in the system. After this transition time, the system appears to have stabilized, with small fluctuations around an average value, $\langle n \rangle / N = 1/2$. During the short transition time, the initial conditions influence the state of the system: The value of $n(t)$ depends on the initial value, $n(0)$, over some interval τ_0. This is also illustrated in Fig. 2.9, which shows the behavior for three different initial conditions, $n(0) = 1$, $n(0) = 1/2$ and $n(0) = 0$.

We could call this transition time the memory extent for the system: The system remembers its initial conditions for some time interval τ_0, and after this time, it has reached the equilibrium state. We expect that the behavior of the system does not depend on the initial state after this transition time. This means that when the

system reaches its equilibrium state, the behavior does no longer depend on the initial condition — the behavior is instead general, depending on the properties of the system when it is in the equilibrium state.

2.4 Approach to equilibrium — theoretical model

From these models we observe that macroscopic quantities, such as n, may become independent of time and of the initial state: If only the number of particles N is large enough the fluctuations will be very small and the quantity will be approximately independent of time. In the example above, we found that the gas system approaches $n(t) = 1/2$ independently of the initial condition of the gas. However, this prompts us to return to our main question: Why do these systems have a particular direction of time? In what way is the situation where all the atoms are on one side ($n(0)/N = 1$) different from the states with $n/N \simeq 1/2$ observed later, when the system has reached what we call equilibrium?

First, let us introduce a way to describe the state of the system. We use the term *microstate* to describe a particular arrangement of the atoms. This could be the positions of all the atoms, $\mathbf{r}_i$, or we could simplify it a bit further, and only specify if an atom is on the left or on the right side of the box. We can denote this by introducing $s_i = 1$ when atom i is on the left side, and $s_i = 0$ when atom i is on the right side. We can describe the microstate of the system by listing all the s_i for all the i atoms in the system. Using this notation we can write a microstate as

$$(s_1, s_2, s_3, \ldots, s_N)\,, \tag{2.5}$$

For the $N = 2$ system there are 4 possible microstates:

Microstate	n	Description
$(1,1)$	2	(atom 1 on the left, atom 2 on the left)
$(1,0)$	1	(atom 1 on the left, atom 2 on the right)
$(0,1)$	1	(atom 1 on the right, atom 2 on the left)
$(1,1)$	0	(atom 1 on the right, atom 2 on the right)

Of these microstates, there is one microstate with $n = 2$, one microstate with $n = 0$ and two microstates with $n = 1$.

Now, if we assume that the system cycles through all the microstates and spends the same amount of time in each microstate, we would assume that the probability for each of the microstates are the same. Each microstate is equally probable. (This is a fundamental assumption. We will return with more arguments as to why this is true later).

This assumption would mean that the system will take the value $n = 2$, $n = 1$, $n = 1$, and $n = 0$ with the same frequency and the same probability, which will mean that the value $n = 1$ is twice as likely as the states $n = 2$ and $n = 0$. For the $N = 2$ the difference between the average value $n = 1$ and the initial condition where

all atoms are on one side, $n = 2$ or $n = 0$, is not that large. But as N increases we will see that the state with $n = N$ becomes increasingly unlikely. For large N almost all the states occur near $n = N/2$ and there are very few states with $n = N$ or $n = 0$. That is what we are going to mean by equilibrium: It is the most probably state. But not the most probably microstate, because any microstate is equally probable, but the most probably macrostate: The most probably value of n in this case. We will dive deeper into these arguments and these distinctions in the next chapters.

(You may already now realize that the probability distribution for n is given by the binomial formula, $P(n,N) = N!/(n!(N-n)!)(1/2)^N$, which is very sharp around $n = N/2$ as N becomes large. We will address these issues in detail in the following chapters).

Summary

- On an atomic level all laws are reversible in time.
- On a macroscopic level, time has a specific direction.
- Energy conservation is not a sufficient principle. We need new principles describing the distribution of energy.
- We can understand the direction of time as a development towards a most likely state in the system.

Exercises

Discussion questions

Exercise 2.1. Direction of time for a bouncing dimer

Fig. 2.10 shows a bouncing dimer ($N = 2$). However, in one of the figures the time axis has been reversed. Which figure shows the correct progression of time and what arguments can you use to find out?

Exercise 2.2. Time development for gas with two atom types

A system consists of two types of atoms, A and B, in gas form in a rectangular box. Initially, all the A atoms are on the left side of the box and all the B atoms are on the right side. Which of the figures in Fig. 2.11 represent realistic time developments of the system? Explain your answer.

Exercise 2.3. One-way hole

A box consists of atoms of one type. Initially, all the atoms are on the left side. There is a hole separating the left from the right side and the atoms can only progress from the left to the right through this hole. Sketch the time development of $n(t)$. Does this behavior seem realistic to you?

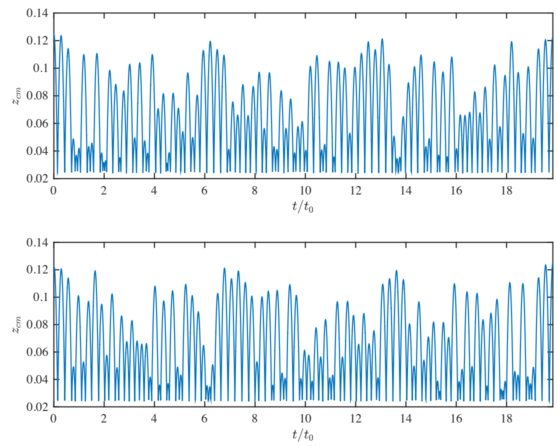

Fig. 2.10 Plot $z_{\mathrm{cm}}(t)$ for a bouncing dimer, that is a mass-spring system with two ($N = 2$) masses. Which figure shows the correct progression of time?

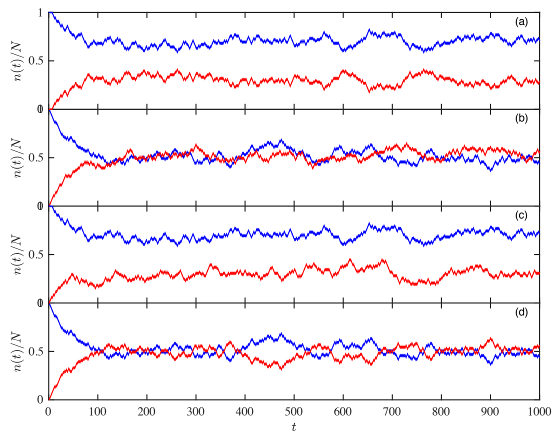

Fig. 2.11 Plot $n_A(t)$ (*blue*) and $n_B(t)$ (*red*) for a two-atom system. However, only one of the figures are from a realistic simulation.

Exercise 2.4. Two-size particles and one-size hole

A box consists of two atoms, A and B, where atom A is smaller than atom B. Initially, the atoms are uniformly distributed throughout the box. Suddenly, we place a sieve in the middle of the box, so that only the A atoms may pass through the sieve, but not the B atoms. Sketch the time development of the number of particles $n_A(t)$ ($n_B(t)$) of type A (type B) on the left side as a function of time.

Exercise 2.5. Irreversible processes

List a few processes from your kitchen that clearly are not reversible and explain why you think the processes are irreversible.

Problems

Exercise 2.6. Bouncing point masses

Discuss the difference between one bouncing point mass and a system consisting of two bouncing point-particles connected by a spring. Calculate the time development for a one-dimensional system and comment on the behavior.

Exercise 2.7. Example of irreversible processes

List more examples of processes that only can go in one direction. Try to articulate why the reverse process does not happen spontaneously.

Exercise 2.8. Lifting a ball

If you take a ball from a table and place it on the floor – can this process spontaneously occur in reverse? If you take a ball from the floor and place it on the table – can this process spontaneously occur in reverse? Explain.

Exercise 2.9. Atomic modeling of mixing

We start a molecular dynamics simulation from a gas state where all the atoms are distributed uniformly in space with random initial velocities. Color the left half red and the right half blue. What is the time development of the number of red particles in the left side? Will you ever return to the starting position?

Chapter 3
Getting started with molecular dynamics modeling

Abstract In this chapter we provide a quick introduction to molecular dynamics modeling. In molecular dynamics the motion of a set of atoms is determined from a model for the inter-atom interactions. We demonstrate the basic physical formulation for a Lennard-Jones model for a gas and provide a Python implementation of the molecular dynamics algorithm. This Python implementation is too slow for any practical application, and we therefore introduce an open-source integrator to determine the motion of all the atoms. Installation and use of the LAMMPS simulator is described in detail. The simulation produces a set of trajectories for all the atoms in the model, and we also demonstrate how to read these trajectories into Python and use this data-set to characterize the behavior of realistic systems.

Statistical mechanics allows us to go from the atomic hypotesis to theories for macroscopic behavior. We will do this by developing strongly simplified models of the physical systems we are interested in: the ideal gas and the ideal crystal models. However, we can also study the behavior of a system in detail by calculating and following the motion of each atom in the system. In this book we will use a modeling tool called molecular dynamics. This is a useful tool that you may indeed use in your professional career as a researcher. Also, molecular dynamics allow you to pose and answer many questions about both atomic scale systems and many-particle systems — questions with non-trivial answers as well as real research questions. The tools we introduce here are indeed the tools that are used to understand nano-scale structures of materials, to understand and engineer chemical processes such as catalysis, to model and understand biological processes and to understand the interaction between drugs and living organisms. Molecular dynamics modeling is an important field of research and engineering, and our quick introduction here is in no way exhaustive, but should give you a quick introduction for the capabilities of the tool.

This chapter starts by motivating the use of classical mechanics to understand atomic motion. We introduce the Lennard-Jones model and standard boundary and initial conditions for atomic modeling studies. We introduce standard integration methods and a very simple illustrative implementation in Python. A more practical approach is then introduced using the LAMMPS software package, provide instal-

lation and simulation instructions for modeling the Lennard-Jones system using this package. We also demonstrate how to visualize and analyse the results of a simulation. Finally, we also demonstrate how LAMMPS can be used to model more realistic systems, such as water, biological and mineral systems. (These finals parts are interesting, but not central to exposition of this book, and can be skipped on a first reading).

3.1 Atomic modeling basics

How do we determine the motion of a system of atoms? This is really a problem in quantum mechanics — the motion and interaction of atoms, including the full motion of the electrons, needs to be determined from the basic equations of quantum mechanics. This is fully possible for small systems with present technology. However, to address the behavior of macroscopic systems, we need thousand, millions, or billions of atoms to get representative results from our calculations. However, if we want to model thousands to millions of atoms we need a different and more computationally efficient approach.

The use of the classical approximation to describe the motion of a system of atoms is, fortunately, able to capture many of the features of macroscopic systems. In this approximation, we use Newton's laws of motion and a well-chosen description of the forces between atoms to find the motion of the atoms. The forces are described by the potential energies of the atoms given their positions. The potential energy functions can, in principle, be determined from quantum mechanical calculations because the forces and the potential energies depend on the states of the electrons — it is the electrons that form the bonds between atoms and are the origin of the forces between atoms. However, in the classical approximation we parametrize this problem: We construct simplified models for the interactions that provide the most important features of the interactions.

3.1.1 Lennard-Jones potential

These parametrized models can be simple or complicated. They can include only pair interactions, three-particle interactions, or even many-particle interactions. Here, we will primarily use simple models, since the statistical and thermal effects we are interested in do not depend strongly on the details of the system. One of the simplest models for atom-atom interactions is a representation of the interactions between nobel-gas atoms, such as between two Argon atoms. For the interaction between two noble-gas atoms we have two main contributions:

- There is an attractive force due to a dipole-dipole interaction. The potential for this interactions is proportional to $(1/r)^6$, where r is the distance between the

atoms. This interaction is called the *van der Waals* interaction and we call the corresponding force the *van der Waals* force.
- There is a repulsive force which is a quantum mechanical effect due to the possibility of overlapping electron orbitals as the two atoms are pushed together. We use a power-law of the form $(1/r)^n$ to represent this interaction. It is common to choose $n = 12$, which gives a good approximation for the behavior of Argon.

The combined model is called the **Lennard-Jones potential**:

$$U(r) = 4\varepsilon \left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right) . \tag{3.1}$$

Here, ε is a characteristic energy, which is specific for the atoms we are modeling, and σ is a characteristic length.

The Lennard-Jones potential and the corresponding force $F(r)$ is illustrated in Fig. **??**. We see that the Lennard-Jones potential reaches its minimum when

$$F(r) = -\frac{d}{dr}U(r) = 0 , \tag{3.2}$$

which occurs at

$$F(r) = 24\,\varepsilon_0 \left((\sigma/r)^6 - 2\,(\sigma/r)^{12} \right) = 0 \;\Rightarrow\; \frac{r}{\sigma} = 2^{1/6} . \tag{3.3}$$

We will use this potential to model the behavior of an Argon system. However, the Lennard-Jones potential is often used not only as a model for a noble gas, but as a fundamental model that reproduces behavior that is representative for systems with many particles. Indeed, Lennard-Jones models are often used as base building blocks in many interatomic potentials, such as for the interaction between water molecules and methane and many other systems where the interactions between molecules or between molecules and atoms is simplified (coarse grained) into a single, simple potential. Using the Lennard-Jones model you can model 10^2 to 10^6 atoms on your laptop and we can model 10^{10}-10^{11} atoms on large supercomputers. However, if you are adventurous you may also model other systems, such as water, graphene, or complex biological systems using this or other potentials as we demonstrate in Sect. 3.5

3.1.2 Initial conditions

An atomic (molecular) dynamics simulation starts from an initial configuration of atoms and determines the trajectories of all the atoms. The initial condition for such

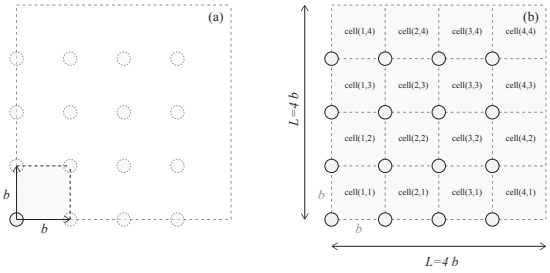

Fig. 3.1 **a** Illustration of a unit cell for a square lattice. **b** A system consisting of 4×4 unit cells, where each of the cells are marked and indexed for illustration.

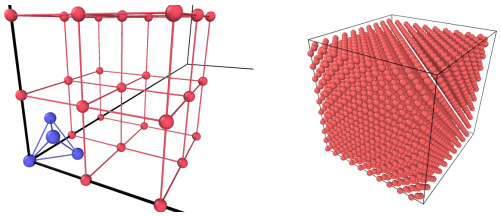

Fig. 3.2 *Left* Illustration of a unit cell for a face centered cubic lattice. Unit cell atoms illustrated in blue and the base position of other cells shown in red. *Right* A system consisting of $10 \times 10 \times 10$ unit cells.

a simulation consists of all the positions, $\mathbf{r}_i(t_0)$ and velocities $\mathbf{v}_i(t_0)$ at the initial time t_0. In order to model a realistic system, it is important to choose the initial configuration with some care. In particular, since most potentials such as the Lennard-Jones potential increase very rapidly as the interatomic distance r goes to zero, it is important not to place the atoms too close to each other. We therefore often place the atoms regularly in space, on a lattice, with initial random velocities.

We generate a lattice by first constructing a *unit cell* and then copying this unit cell to each position of a lattice to form a regular pattern of unit cells. (The unit cell may contain more than one atom). Here, we will use cubic unit cells. For a cubic unit cell of length b with only one atom in each unit cell, we can place the atom at $(0,0,0)$ in the unit cell and generate a cubic lattice with distances b between the atoms by using this cubic unit cell to build a lattice. This is illustrated for a two-dimensional system in Fig. 3.1. Such a lattice is called a *simple cubic lattice*.

However, for a Lennard-Jones system we know (from other theoretical, numerical and experimental studies) that the equilibrium crystal structure is not a simple cubic lattice, but a face centered cubic lattice. This is a cubic lattice, with additional atoms added on the center of each of the faces of the cubes. The unit cell for a face centered cubic lattice is illustrated in Fig. 3.2. We will use this as our basis for a simulation and then select a lattice size b so that we get a given density of atoms. The whole system will then consist of $L \times L \times L$ cells, each of size $b \times b \times b$ and with 4 atoms in each cell.

3.1.3 Boundary conditions

A typical molecular model of a liquid of Argon molecules is illustrated in Fig. 3.3a. In this case, we have illustrated a small system of approximately $10 \times 10 \times 10$ atom diameters in size. Below, you will learn how to set up and simulate such systems on your laptop. Unfortunately, you will not be able to model macroscopically large systems — neither on your laptop nor on the largest machine in the world. A liter of gas at room temperature typically contains about 10^{23} atoms, and this is simply beyond practical computational capabilities.

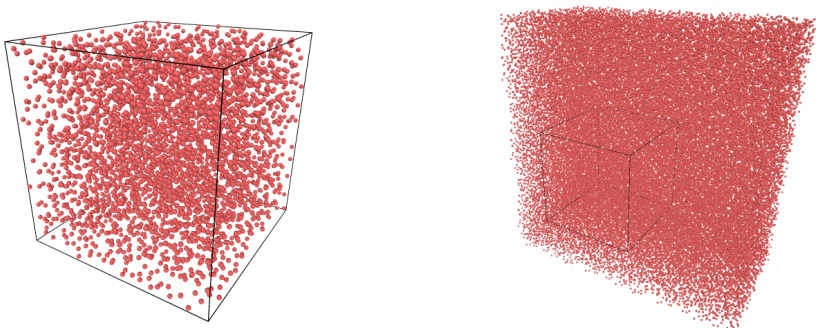

Fig. 3.3 **a** Visualization of a $10 \times 10 \times 10$ simulation of an Argon liquid. **b** Visualization of a 10×10 simulation of an Argon gas, showing the actual simulation area in the center box, and 8 copies of the center box illustrating the principles of the periodic boundary conditions.

But it is possible with a small system to gain some insights into how very large, even infinite, systems behave? One of the problems with the $10 \times 10 \times 10$ system above is the external boundaries. But we can fool the system into believing it is infinite by applying what we call periodic boundary conditions. If the atoms on the left of the system do not see emptiness to their left, but instead see the right hand side of the system, as if the system is wrapped around a cylinder, the system will look like it is inifinite. This is illustrated in Fig. 3.3b. This convention of periodic boundary conditions is usually applied in all simulations in order to avoid dealing with boundary conditions. (There may be some possibly problematic aspects of periodic boundaries, but we will not address or worry about these here).

3.1.4 Integrating the equations of motion

How do we determine the behavior of the system? We solve the equations of motion. For molecular dynamics simulations we usually use an algorithm called the Velocity-Verlet, which is approximately like the forward Euler method, but it is very well suited for conservative forces. The velocity is calculated at both time t and at

intermediate times $t + \Delta t/2$, where Δt is the time-step, whereas the forces are only calculated at the full time-steps, t, $t + \Delta t$, $t + 2\Delta t$ etc. The most time-consuming part of the calculation is the calculation of the forces. We therefore want to limit the number of times we calculate the forces and still have as high precision as possible in our calculations. The Velocity Verlet algorithm ensures a good trade-off between precision in the integration algorithm and the number of times forces are calculated.

In the Velocity-Verlet algorithm the positions $\mathbf{r}_i(t)$ and velocities $\mathbf{v}_i(t)$ of all the atoms, $i = 1, \ldots, N$, are propagated forward in time according to the algorithm:

$$\mathbf{v}_i(t + \Delta t/2) = \mathbf{v}_(t) + \mathbf{F}_i(t)/m_i \Delta t/2 \tag{3.4}$$

$$\mathbf{r}_i(t + \Delta t) = \mathbf{r}_(t) + \mathbf{v}_i(t + \Delta t/2) \tag{3.5}$$

$$\mathbf{F}_i(t + \Delta t) = -\nabla V(\mathbf{r}_i(t + \Delta t)) \tag{3.6}$$

$$\mathbf{v}_i(t + \Delta t) = \mathbf{v}_(t + \Delta t/2) + \mathbf{F}_i(t + \Delta t)/m_i \Delta t/2\ ,, \tag{3.7}$$

This method has very good properties when it comes to energy conservation, and it does, of course, preserve momentum perfectly.

3.2 Simple implementation of a molecular dynamics simulator

How can we implement the full simulation procedure in Python? We need to set up the initial configuration of atoms, integrate the motion for a given number of time-steps, and then output the results to a file that can be read by standard visualization programs. Here, we provide a full implementation to show how a molecular dynamics simulation is implemented. However, this is mainly for illustrational purposes so that you can see how it is done. We will actually use a state-of-the-art open-source simulation package to perform our simulations.

3.2.1 Non-dimensional equations of motion

However, all the quantities in a molecular dynamics simulations are very small. It is therefore usual to introduce measurement units that are adapted to the task. For the Lennard-Jones model we usually use the intrinsic length and energy scale of the model as the basic units of length and energy. This means that we measure lengths in units of σ and energies in units of ε_0. A vector $\mathbf{r}'_i$ in the simulation is therefore related to the real-world length $\mathbf{r}_i$ through

$$\mathbf{r}_i = \sigma\, \mathbf{r}'_i \;\Leftrightarrow\; \mathbf{r}'_i = \mathbf{r}_i/\sigma\ . \tag{3.8}$$

Similarly, we can introduce a Lennard-Jones time, $\tau = \sigma\sqrt{m/\varepsilon}$, where m is the mass of the atoms, and the Lennard-Jones temperature $T_0 = \varepsilon/k_B$. Using these notations, we can rewrite the equations of motion for the Lennard-Jones system using the non-

dimensional position and time, $\mathbf{r}'_i = \mathbf{r}_i/\sigma$ and $t' = t/\tau$:

$$m\frac{d^2}{dt^2}\mathbf{r}_i = \sum_j 24\varepsilon_0 \left((\sigma/r_{ij})^6 - 2\,(\sigma/r_{ij})^{12} \right) \left(\mathbf{r}_{ij}/r_{ij}^2 \right) \ , \tag{3.9}$$

to become

$$\frac{d^2}{d(t')^2}\mathbf{r}'_i = \sum_j 24 \left(r_{ij}^{-6} - 2r_{ij}^{-12} \right) \left(\mathbf{r}'_{ij}/r'^2_{ij} \right) \ . \tag{3.10}$$

Notice that this equation is general. All the specifics of the system is now part of the characteristic length, time and energy scales σ, τ, and ε_0. For Argon $\sigma = 0.3405\mu$m, and $\varepsilon = 1.0318 \cdot 10^{-2}$eV, and for other atoms you need to find the corresponding parameter values.

Quantity	Equation	Conversion factor	Value for Argon
Length	$x' = x/L_0$	$L_0 = \sigma$	0.3405μm
Time	$t' = t/tau$	$\tau = \sigma\sqrt{m/\varepsilon}$	$2.1569 \cdot 10^3$ fs
Force	$F' = F/F_0$	$F_0 = m\sigma/\tau^2 = \varepsilon/\sigma$	$3.0303 \cdot 10^{-1}$
Energy	$E' = E/E_0$	$E_0 = \varepsilon$	$1.0318 \cdot 10^{-2}$ eV
Temperature	$T' = T/T_0$	$T_0 = \varepsilon/k_B$	119.74 K

3.2.2 Initialization of system

First, we need to initialize the system by generating a lattice of atoms with random velocities. We write a small script to generate a lattice and write it to disk. The lattice consists of units cells of size b (measure in units of σ, as are everything in the simulation). If a unit cell starts at $\mathbf{r}_0$, then the 4 atoms in the units cell are at positions $\mathbf{r}_0$, $\mathbf{r}_0 + (b/2, b/2, 0)$, $\mathbf{r}_0 + (b_2, 0, b_2)$, and $\mathbf{r}_0 + (0, b/2, b/2)$. If a system consists of $L \times L \times L$ such cells it is simple to create the system: We simply loop through all L^3 positions of $\mathbf{r}_0$ and for each such position we add atoms at the four positions relative to $\mathbf{r}_0$. This is done using the following script, which is explained in more detail below:

```
L = 5; % Lattice size
b = 2.0; % Size of unit cell (units of sigma)
v0 = 1.0; % Initial kinetic energy scale
N = 4*L^3; % Nr of atoms
r = zeros(N,3);
v = zeros(N,3);
bvec = [0 0 0; b/2 b/2 0; b/2 0 b/2; 0 b/2 b/2];
ip = 0;
% Generate positions
for ix = 0:L-1
    for iy = 0:L-1
```

```
        for iz = 0:L-1
            r0 = b*[ix iy iz]; % Unit cell base position
            for k = 1:4
                ip = ip + 1; % Add particle
                r(ip,:) = r0 + bvec(k,:);
            end
        end
    end
end
% Generate velocities
for i = 1:ip
    v(i,:) = v0*randn(1,3);
end
% Output to file
writelammps('mymdinit.lammpstrj',L*b,L*b,L*b,r,v);
```

In addition, we need the following function to write the data to file:

```
function writelammps(filename,Lx,Ly,Lz,r,v)
%WRITELAMMPS Write data to lammps file
fp = fopen(filename,'w');
s = size(r);
ip = s(1);
fprintf(fp,'ITEM: TIMESTEP\n');
fprintf(fp,'0\n');
fprintf(fp,'ITEM: NUMBER OF ATOMS\n');
fprintf(fp,'%d\n',ip); % Nr of atoms
fprintf(fp,'ITEM: BOX BOUNDS pp pp pp\n');
fprintf(fp,'%f %f\n',0.0,Lx); % box size, x
fprintf(fp,'%f %f\n',0.0,Ly); % box size, y
fprintf(fp,'%f %f\n',0.0,Lz); % box size, z
fprintf(fp,'ITEM: ATOMS id type x y z vx vy vz\n');
for i = 1:ip
    fprintf(fp,'%d %d %f %f %f %f %f %f \n',...
            i,1,r(i,:),v(i,:));
end
fclose(fp);
end
```

Notice that we use the vectors $\mathbf{b}_k$ to describe the four positions relative to the origin, $\mathbf{r}_0$ of each cell:

```
bvec = [0 0 0; b/2 b/2 0; b/2 0 b/2; 0 b/2 b/2];
...
r0 = b*[ix iy iz]; % Unit cell base position
for k = 1:4
    ip = ip + 1; % Add particle
    r(ip,:) = r0 + bvec(k,:);
end
```

Also notice that we use a variable `ip` to keep track of the particle index for the atom we are currently adding to the system. This is simpler than calculating the particle number each time. That is, we can simply use

```
    ip = ip + 1; % Add particle
```

instead of the more elaborate and less transparent

```
    ip = ix*4*L*L + iy*4*L + iz
```

to calculate the current particle number each time a new atom is added. Finally, we add a random velocity vector, with a normal (Gaussian) distribution of velocities with a standard deviation of v_0 using the `randm` function. This can be done using a loop or with a single, vectorized command:

```
for i = 1:ip
    v(i,:) = v0*randn(1,3);
end
#
```

or

```
v = v0*randn(ip,3);
```

Finally, the state is saved to the file `mymdinit.lammpstrj`. The resulting file, which contains the complete atomic state of the system, looks like this:

```
ITEM: TIMESTEP
0
ITEM: NUMBER OF ATOMS
500
ITEM: BOX BOUNDS pp pp pp
0.000000 10.000000
0.000000 10.000000
0.000000 10.000000
ITEM: ATOMS id type x y z vx vy vz
1 1 0.000000 0.000000 0.000000 -0.306633 -2.732455 1.612753
2 1 1.000000 1.000000 0.000000 0.099804 0.487968 -1.545902
3 1 1.000000 0.000000 1.000000 -0.500267 0.777696 0.028699
4 1 0.000000 1.000000 1.000000 -0.404407 -0.867741 0.626161
...
```

where we only have included the first four of 500 atoms. This file can then be used as a starting point for a simulation. The output from a simulation will have a similar format, providing the state of the atomic system which we can then analyze in details.

3.2.3 Integration of the equations of motion

Starting from the initial configuration, we integrate the equations of motion to find the particle trajectories and output them to file for further analysis. First, we write a short function to read the initial configuration from file, and then we integrate and write results to a file.

The function readlammps[1] used to read from a LAMMPS trajectory file will be reused many times throughout this book, hence we strive to make it sufficiently general and sophisticated. You can find the listing and discussion of this program in the Appendix (Chap. **??**). Here, we simply notice that we can call it as

```
[Lx,Ly,Lz,r,v]=readlammps('mymdinit.lammpstrj');
```

In order to read in the initial state. The following program integrates the equation of motion:

```
[Lx,Ly,Lz,r,v]=readlammps('mymdinit.lammpstrj');
L = [Lx Ly Lz]; s = size(r); N = s(1);
t = 3.0; dt = 0.001; n = ceil(t/dt);
a = zeros(N,3);  % Store calculated accelerations
for i = 1:n-1 % Loop over timesteps
    a(:,:) = 0;
    for i1 = 1:N
        for i2 = i1+1:N
            dr = r(i,i1,:) - r(i,i2,:);
            for k = 1:3  % Periodic boundary conditions
                if (dr(k)>L(k)/2) then
                    dr(k) = dr(k) - L(k);
                end
                if (dr(k)<-L(k)/2) then
                    dr(k) = dr(k) + L(k);
                end
            end
            rr = dot(dr,dr);
            aa = -24*(2*(1/rr)^6-(1/rr)^3)*dr/rr;
            a(i1,:) = a(i1,:) + aa(1); % from i2 on i1
            a(i2,:) = a(i2,:) - aa(2); % from i1 on i2
        end
    end
    v(i+1,:,:) = v(i,:,:) + a*dt;
    r(i+1,:,:) = r(i,:,:) + v(i+1,:,:)*dt;
    % Periodic boundary conditions
    for i1 = 1:N
        for k = 1:3
            if (r(i+1,i1,k)>L(k) then
                r(i+1,i1,k) = r(i+1,i1,k) - L(k);
            end
             if (r(i+1,i1,k)<0 then
                r(i+1,i1,k) = r(i+1,i1,k) + L(k);
            end
        end
    end
end
writelammps('mymddump.lammpstrj',Lx,Ly,Lz,r,v);
```

Notice how the periodic boundary conditions are implemented. They need to be included both when the relative distance $\Delta \mathbf{r}_{ij}$ between two atoms are calculated and when the positions of the atoms are updated during the integration step.

[1] http://folk.uio.no/malthe/fys2160/readlammps.m

The main idea here is to show you the structure and principles of a molecular dynamics program, as well as how data is handled and stored in files. This Python program is not practically useful because Python is too slow, in particular for a program using many nested loops. We will therefore not use this code, but instead use a professionally developed open-source molecular dynamics integrator to find the time development of the atomic system. The principles are the same, but the computational efficiency is many orders of magnitude greater for a professional code.

3.3 Running a molecular dynamics simulation

There are several efficient packages that solves the equations of motion for a molecular dynamics simulation. The packages allow us to model a wide variety of systems, atoms and molecules, and are efficienty implemented on various computing platforms, making use of modern hardware on your laptop or desktop or state-of-the-art supercomputing fascilities. We use a particular tool developed at Sandia National Laboratories called LAMMPS[2].

3.3.1 Installation of LAMMPS

If you want to be able to reproduce the simulations performed here you will need to install LAMMPS on your computer. This is very simple if you have a Mac or an Ubuntu system — for a windows system you will have to follow the installation instructions found at the web-site for LAMMPS. You can find all the documentation of LAMMS here[3].

Mac installation. On a mac you should be able to install LAMMS using Homebrew or Macports.

Using **Homebrew** (Homebrew[4]) LAMMPS is installed with:

```
brew tap homebrew/science
brew install lammps
```

If you want to use the parallell implementation of LAMMPS you can install this version using

```
brew tap homebrew/science
brew install lammps --with-mpi
```

[2] `http://lammps.sandia.gov/`

[3] `http://lammps.sandia.gov/`

[4] `http://brew.sh/`

Using **MacPorts** (MacPorts[5]) LAMMPS is installed with:

```
port install lammps
```

Ubuntu installation. You can install a recent version of LAMMPS with:

```
sudo apt-get install lammps
```

Python interface installation. In addition you should install the module `Pizza.py` which we use to read simulation data into Python, and you need a Python installation that includes `pylab`. I recommend Anaconda[6], but the Enthought version also works fine. Download Pizza.py[7] and follow the installation instructions.

3.3.2 Starting a simluation in LAMMPS

If you have successfully installed LAMMPS, you are ready to start your first molecular dynamics simulations. The LAMMPS simulator reads its instructions on how to run a simulation from an input file with a specific syntax. Here, we will set up a two-dimensional simulation of a Lennard-Jones system using the file `in.myfirstmd`:

```
# 2d Lennard-Jones gas
units lj
dimension 2
boundary p p p
atom_style atomic

lattice hex 0.75
region simbox block 0 20 0 10 -0.1 0.1
create_box 1 simbox
create_atoms 1 box

mass 1 1.0
velocity all create 2.5 87287

pair_style lj/cut 2.5
pair_coeff 1 1 1.0 1.0 2.5

neighbor 0.3 bin
neigh_modify every 20 delay 0 check no

fix 1 all nve

dump 1 all custom 10 dump.lammpstrj id type x y z vx vy vz
thermo 100
run 5000
```

[5] `https://www.macports.org/`

[6] `http://continuum.io/downloads`

[7] `http://pizza.sandia.gov/`

The simulation is run from the command line in a Terminal. Notice that the file `in.myfirstmd` must be in your current directory. I suggest creating a new directory for each simulation, copying the file `in.myfirstmd` into the directory and modifying the file to set up your simulation, before starting the simulation with:

```
lammps < in.myfirstmd
```

This simulation should only take a few seconds. It produces output in the Terminal and two files: `dump.lammpstrj`, which contains all the data from the simulation, and `log.lammps`, which contains a copy of what was output to your terminal. Fig. 3.4 illustrates the positions of the atoms initially and at the end of the simulation.

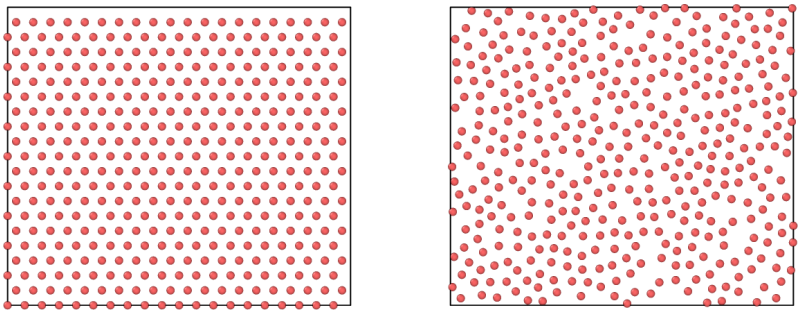

Fig. 3.4 (*Left*) Illustration of the initial hexagonal lattice generated in the simulation. (*Right*) Illustration of the final positions of the atoms after 5000 timesteps.

The input file `in.myfirstmd` consists of a series of commands to be interpreted by LAMMPS. Here, we look at what these do in detail. (You can skip this at first reading, and return when you wonder what the parameters actually do).

```
# 2d Lennard-Jones gas
```

This line starts with a # and is a comment that is ignored by the program.

```
units lj
dimension 2
boundary p p p
atom_style atomic
```

This block describes general features of the simulation:

The `units lj` command selects Lennard-Jones units, which were introduced in Sect. 3.2.1. This means that lengths are measured in units of σ, energies in units of ε_0, time in units of $\tau = \sigma\sqrt{m/\varepsilon}$, and temperature in terms of

$T_0 = \varepsilon/k_B$. For Argon $\sigma = 0.3405\mu$m, and $\varepsilon = 1.0318 \cdot 10^{-2}$eV. Other atomic models will have other parameters.

The `dimension` command specifies the dimensionality of the simulation: 2 or 3. Here we run a 2d simulation.

The `boundary` command specifies boundary conditions to be applied. Here we have periodic boundaries in the x-, y-, and z- directions.

The `atom_style` command specifies the complexity of the description of each atom/particle. Here, we will use the simplest description, `atomic`, which is used for noble gases and coarse-grained simulation models.

```
lattice hex 0.75
region simbox block 0 20 0 10 -0.1 0.1
create_box 1 simbox
create_atoms 1 box
```

This block sets up the dimensions of the 20×10 simulation box and fills the box with atoms with a given packing fraction.

The `lattice` command generates a lattice of points. This does, surprisingly enough, not actually generate any atoms, it only generates a set of positions in space where atoms will be generated when we generate atoms. The type `hex` specifies a two-dimensional lattice of hexagonal shape, so that each atom has six nearest neighbors. And the number `0.75` is called the scale and is the reduced density, ρ', when we have chosen LJ units for the simulation. (Notice that the scale is interpreted differently if we do not use LJ units, see the LAMMPS documentation for more information).

The `region` command defines a region which is a `block` extending over $0 < x < 20$, $0 < y < 10$, $-0.1 < z < 0.1$ We give this region the name `simbox`.

The `create_box` command now actually creates the simulation box based on the spatial region we called `simbox`. The simulation box will only contain 1 (one) type of atoms, hence the number `1`.

The `create_atoms` finally fills the simulation box we have defined using the lattice we have defined with atoms of type `1`.

```
mass 1 1.0
velocity all create 2.5 87287
```

This block defines the material properties of the atoms and defines their initial velocities.

The `mass` command defines that atoms of type `1` will have a mass of `1.0` relative to the mass of the Lennard-Jones model. This means that all atoms have mass 1 in the Lennard-Jones units. This means that the masses of all the atoms are the same as the mass m used in the non-dimensionalization of the Lennard-Jones model.

The `velocity` command generates random velocities (using a Gaussian distribution) so that the initial temperature for `all` atom types in the system is `2.5` in the dimensionless Lennard-Jones units. The last, strange integer number `87287` is the seed for the random number generator used to generate the

random numbers. As long as you do not change the seed number you will always generate same initial distribution of velocities for this simulation.

```
pair_style lj/cut 2.5
pair_coeff 1 1 1.0 1.0 2.5
```

This block specifies the potential between the atoms.

The `pair_style` command specifies the we want to use a Lennard-Jones potential with a cut-off that is of length `2.5`. What does this mean? It means that since the Lennard-Jones potential falls so quickly to zero as the distance between the atoms increase, we will approximate interaction to be zero when the atoms are further than 2.5 away from each other (measured in Lennard-Jones units, that is in units of σ). The simulator ensures that both the potential and the force is continuous across the transition. There are many other types of force fields that you may use — take a look at the documentation of LAMMPS for ideas and examples.

The `pair_coeff` command specifies the parameters of the Lennard-Jones model. The two first numbers, `1 1`, specifies that we describe the interaction of atoms of type 1 with atoms of type 1. And the parameters of the Lennard-Jones model are `1.0 1.0 2.5`. This means that The interaction between an atom of type 1 with an atom of type 1 has a σ-value corresponding 1.0 times the the general σ-value (hence the first number 1.0), and a ε_0-value corresponding to 1.0 times the overall ε-value (hence the second number 1.0). The cut-off for this interaction is 2.5 — the same value as we specified above.

```
neighbor 0.3 bin
neigh_modify every 20 delay 0 check no
```

This block contains information about how the simulator is to calculate the interaction between the atoms using list of neighbors that are updated at regular intervals. You do not need to change these parameters, but changing them will typically not have any effects on the simulation results, only on the efficiency of the simulations.

```
fix 1 all nve
```

This one-line block specifies what type of simulation we are performing on the atoms. This is done by one or more `fix` commands that can be applied to regions of atoms. Here, the `fix`, which we call `1` (you can choose numbers or names for identity), is applied to `all` particles and specifies that the simulation is run at constant `nve`, that is, at constant number of particles (`n`), constant volume (`v`, meaning that the simulation box does not change during the simulation), and constant energy (`e`). You may be surprised by the constant energy part. Does the integration algorithm ensure that the energy is constant. Yes, it does. However, there can be cases where we want to add energy to a particular part of the system, and in that case the basic interation algorithm still conserves energy, but we add additional terms that may change the total energy of the system.

```
dump 1 all custom 10 dump.lammpstrj id type x y z vx vy vz
thermo 100
run 5000
```

This block specifies simulation control, inclusing output and the number of time-steps to simulate.

The `dump` command tells the simulator to output the state. The `1` is the name we give this dump — it could also have been given a name such as `mydump`. We specify that `all` atoms are to be output using a `custom` output format, with output every `10` time-steps to the file `dump.lammpstrj`, and the 'id type x y z vx vy vz' list specifies what is output per atom.

The `thermo` command specifies that output to the Terminal and to the log file, `log.lammps`, is provided every `100` timesteps.

The `run` command starts the simulation and specifies that it will run for `5000` timesteps.

3.3.3 Visualizing the results of a simulation

It is good practice to look at the results of the simulation. Use for example VMD[8] or Ovito[9] to visualize the results. Here, let us demonstrate the use of VMD. First, open VMD. Then open the `File -> New Molecule` menu item. You find the `dump.lammpstrj` file that were generated by your simulation run, and press load. The whole time sequence of your simulation is now loaded in VMD. However, the default visualization mode is usually not that clear. You fix that by going to the main menu again, and select `Graphics -> Representations...` In the window that opens you change the `Drawing method` to `CPK` and you select `Bond Radis` to be 0.0. Now, your visualization should show the time dynamics of an ensemble of atoms. Enjoy!

3.4 Analyzing the results of a simulation

The results from the simulations can be analyzed using built-in tools from LAMMPS. We demonstrate these tools by measuring the number of particle on the left side of the box as a function of time, by extracting that data from the simulation files.

We have illustrated the text-format output files from LAMMPS above. The file may contain data from may timesteps, t. For each timestep, there are a few initial lines describing the system, followed by a list of properties for each atom. We have chosen the custom output form

[8] `http://www.ks.uiuc.edu/Research/vmd/`

[9] `http://www.ovito.org/`

```
dump 1 all custom 10 dump.lammpstrj id type x y z vx vy vz
```

Each line of atom properties contains data of the form `id type x y z vx vy vz`. This means that we will get access to both the atom position and the atom velocity for each of the atoms if we read the data in this file. Our strategy will then be to use the atom position x_i and compare it with $L_x/2$, where L_x is the dimensions of the simulation box, measured using the same units as used for the positions x_i of the atoms. How can we read the data from the LAMMPS dump files. We could read our own input functions, but it is simpler to use already developed code, which is distributed with the LAMMPS distribution.

3.4.1 Matlab implementation

For matlab, we use the tools found in the `tools/matlab` directory, which was part of the lammps installation. (If you cannot find these after installing LAMMS, you can always download LAMMS again as a tarball, extract the tarball, and find the programs in the resulting directory). You need to copy the file `readdump_all.m` and `readdump_one.m` to your current working directory — the directory where you ran the simulation and where the file `dump.lammpstrj` is located.

First, we read data from all the timesteps into matlab:

```
data = readdump_all('dump.lammpstrj');
```

We need the time values t_i at each of the timesteps and the number of time-steps, `nt`:

```
t = data.timestep;
nt = length(t);
```

We set up an array, `nleft`, to store $n(t)$, the number of atoms on the left side as a function of time:

```
nleft = zeros(nt,1);
```

The size of the simulation box is stored in the variable `box`, and we store $L_x/2$ in the variable `halfsize`:

```
box = data.x_bound;
halfsize = 0.5*box(:,2);
```

Now, we loop through all the timesteps. For each timestep we extract all the x-positions for all the atoms in a list `xit`. Next, we find a list of all the atoms that are on the left side, that is, all the atoms with an x-position smaller than $L_x/2$. This is done by the `find` command. Finally, we count how many elements are in this list. This is the number of atoms that are on the left side of the box. We store this number in the array `nleft` for this timestep, and reloop to address the next timestep:

```
for it = 1:nt
```

```
    xit = data.atom_data(:,3,it);
    jj = find(xit<halfsize(it));
    nleft(it) = length(jj);
end
```

The complete program reads:

```
%start1
data = readdump_all('dump.lammpstrj');
t = data.timestep;
nt = length(t);
nleft = zeros(nt,1);
box = data.x_bound;
halfsize = 0.5*box(:,2);
for it = 1:nt
    xit = data.atom_data(:,3,it);
    jj = find(xit<halfsize(it));
    nleft(it) = length(jj);
end
plot(t,nleft), xlabel('t'), ylabel('n')
```

A short and simple program to handle a complicated set of data. This should provide you with a good basis for measuring various properties of the simulated system using Python.

3.4.2 Python implementation

For Python we use the tool `Pizza.py`. You should then simply start your python session by calling `pizza.py` initially. If we assume that you have installed the program in `/pizza`, you can start your `ipython session` using `pizza.py` by typing

```
ipython -i ~/pizza/src/pizza.py ~/pizza/src/pizza.py
```

in your terminal. The you are ready to analyze data from the simulation.

First, we read data from all the timesteps into Python:

```
from pylab import *
data = dump("dump.lammpstrj") # Read all timesteps
```

We need the time values t_i at each of the timesteps and the number of time-steps, `nt`:

```
t = data.time()
nt = size(t)
```

We set up an array, `nleft`, to store $n(t)$, the number of atoms on the left side as a function of time:

```
nleft = zeros(n,float); # Store number of particles
```

The size of the simulation box is found in the variable `box`, and we store $L_x/2$ in the variable `halfsize`:

```
tmp_time,box,atoms,bonds,tris,lines = data.viz(0)
halfsize = 0.5*box[3] # Box size in x-dir
```

Now, we loop through all the timesteps. For each timestep we extract all the x-positions for all the atoms in a list `xit`. Next, we find a list of all the atoms that are on the left side, that is, all the atoms with an x-position smaller than $L_x/2$. This is done by the `find` command. Finally, we count how many elements are in this list. This is the number of atoms that are on the left side of the box. We store this number in the array `nleft` for this timestep, and reloop to address the next timestep:

```
#Get information about simulation box
tmp_time,box,atoms,bonds,tris,lines = data.viz(0)
halfsize = 0.5*box[3] # Box size in x-dir
for it in range(nt):
   xit = array(data.vecs(it,"x"))
   jj = find(xit<halfsize)
   nleft[it] = size(jj)
```

The complete program reads:

```
#start1
from pylab import *
data = dump("dump.lammpstrj") # Read all timesteps
t = data.time()
nt = size(t)
nleft = zeros(n,float); # Store number of particles
#Get information about simulation box
tmp_time,box,atoms,bonds,tris,lines = data.viz(0)
halfsize = 0.5*box[3] # Box size in x-dir
for it in range(nt):
   xit = array(data.vecs(it,"x"))
   jj = find(xit<halfsize)
   nleft[it] = size(jj)
plot(t,nleft), xlabel('t'), ylabel('N_p'), show()
```

A short and simple program to handle a complicated set of data. This should provide you with a good basis for measuring various properties of the simulated system using Python.

3.4.3 Results

The resulting plot is shown in fig. **??**. If you wonder why the number of atoms on the left is varying, how it is varying, and how to describe how it is varying — this indeed in the topic of this book so just read on!

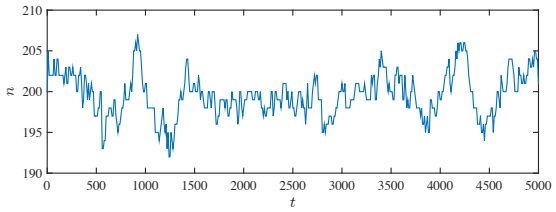

Fig. 3.5 Plot of $n(t)$, the number of atoms on the left side of the simulation box, as a function of time, where the time is measured in the number of timesteps in the molecular dynamics simulation.

3.5 Advanced models

Molecular dynamics simulation packages such as LAMMPS are professional tools that are used for research and development. Here, I provide a few examples of more advanced use of LAMMPS that may inspire you to try the tool also on your own.

3.5.1 Coarsening

Out first example is an extension of the first two-dimensional simulation you performed above. What happens if we start the system with a homogeneous distribution of atoms, but with a low initial energy? In addition, we keep the average kinetic energy in the system approximately constant. (This corresponds, as you will learn later, approximately to keeping the temperature in the system constant):

```
units lj
dimension 2
boundary p p p
atom_style atomic

lattice hex 0.5
region simbox block 0 80 0 40 -0.1 0.1
create_box 1 simbox
create_atoms 1 box

mass 1 1.0
velocity all create 0.05 87287

pair_style lj/cut 2.5
pair_coeff 1 1 1.0 1.0 2.5

neighbor 0.3 bin
neigh_modify every 20 delay 0 check no

fix 1 all nvt temp 0.25 0.25 1.0
dump 1 all atom 1000 dump.lammpstrj
thermo 100
run 50000
```

The resulting behavior shown in Fig. **??** is a phase separation: The system separates into a liquid and a gas phase, demonstrating a phenomenon called spinoidal decomposition. We will discuss the behavior of such systems using both molecular dynamics models and algorithmic models later.

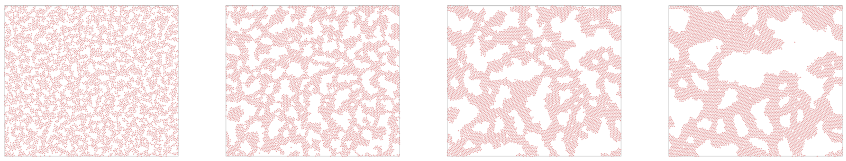

Fig. 3.6 Snapshots from a simulation of a Lennard-Jones liquid, with a low initial density. A slowening coarsening is observed.

3.5.2 Hemoglobin in water

3.5.3 Lipid bilayers

3.5.4 Fracture of Silicon crystal

3.5.5 Water permeation in a carbon nanotube

Summary

- Molecular dynamics simulations model how atomic systems propagate forward in time by integrating the equations of motion based on a given interaction potential between the atoms.
- We have introduced a commonly used interaction potential, the Lennard-Jones potential: $V(r) = 4\varepsilon\left((\sigma/r)^{-12} - (\sigma/r)^{-6}\right)$, where the behavior is determined by the energy scale ε_0 and the length scale σ.
- We run molecular dynamics simulations using custom code, public or commercial codes, which include efficient integrators.
- A typical simulation models a system with constant energy, volume, and number of particles.
- Molecular dynamics simulations are routinely used as part of research and development tasks across disciplines, and are important tools for applications such as the development of new materials, chemical catalysis optimization and drug development.

Exercises

Problems

Exercise 3.1. Evolution of density

We will here study a $10 \times 10 \times 10$ system of Argon atoms, as done above. Initiate the system with all particles in the left 1/3 of the system. We call the number of particles in the left, center and middle thirds of the system $n_1(t)$, $n_2(t)$, and $n_3(t)$ respectively.

a) How do you expect n_1, n_2, and n_3 to develop in time?

b) Measure $n_1(t)$, $n_2(t)$, and $n_3(t)$ and compare with your predictions.

Exercise 3.2. Temperature variation

a) Measure the maximum and minimum temperature, $T_{\max}$ and $T_{\min}$ respectively for a simulation of $L = 10 \times 10 \times 10$ atoms, and calculate $\Delta T = T_{\max} - T_{\min}$.

b) Measure ΔT for three different system size, $L = 4$, $L = 8$, and $L = 16$. Comment on the results.

Exercise 3.3. Coarsening

Start a LAMMPS simulation with the following init file:

```
# 3d Lennard-Jones melt
units       lj
atom_style  atomic
lattice     fcc 0.2
region      box block 0 10 0 10 0 10
create_box  1 box
create_atoms    1 box
mass        1 1.0
velocity    all create 0.1 87287
pair_style  lj/cut 2.5
pair_coeff  1 1 1.0 1.0 2.5
neighbor    0.3 bin
neigh_modify    every 20 delay 0 check no
fix 1 all nvt temp 0.05 0.05 1.0
dump        id all atom 50 dump.lammpstrj
thermo      100
run     10000
```

(This simulation is performed at constant temperature and not at constant energy). Visualize the time development of the system and comment on what you see.

Chapter 4
Probability and Statistics

Abstract This chapter will introduce you to some of the most common and useful tools from statistics, tools that you will need throughout your career. We will learn about statistics motivated by an introductory example — trying to understand the data you generated by running a molecular dynamics simulations. This is a good starting point, since statistics really is about describing real data. The data from a realistic molecular dynamics simulation can be quite overwhelming, the data-sets become extremely big, so we need effective tools to pick out the most important information. Here we introduce the concept of probabilities, the basic properties of probabilities, such as the rule of addition and the rule of multiplication. We introduce probability distributions and densities. We introduce the average and standard deviation, and discern between the underlying parameters of a probability distribution and the estimated parameters measured from observations. We introduce the biomial, normal and exponential distribution. We demonstrate that the distribution of a sum of independent variables usually are normally distributed, independently of the details of the distribution of each event in the sum. Finally, we use these tools to develop a simple theory for the number of particles on the left side of a gas.

Statistics provides us with the tools to describe real data, whereas probability theory provides us with the theoretical underpinning for statistics. Here, we will start with a practical approach to probability theory, based on the idea of frequencies of event, and use this to develop the fundamental laws of probabilities. This approach suits a physicists approach well and is also the most direct approach from a computational perspective. Again, we will introduce both theory and the computational tools needed to test the theory and make accurate measurements also for large data sets.

Probability theory is a classic field that has occupied many of the great mathematicians and physicists throughout history. It has been develop along with gambling, but this is just an unfortunate association. Probability theory is the tool we need to address a world of real data and unknown processes, and the tool we need to describe the behavior of systems with many particles, where the physical laws no longer are absolute, but only represent the overwhelmingly most probable outcome of an experiment or an observation. However, we will start our venture into statisti-

cal and probability motivated by observations and real data — in our case the data from measuring the number of atoms on the left side of a box filled with a gas.

4.1 Motivating example — an ideal gas

We have already seen that molecular dynamics simulations of a gas produces fluctuations, because the atoms are moving approximately randomly around. Here, we will use the data from this approximately random system as a model to analyze and understand random systems.

We introduced the measure $n(t)$ to characterize the number of atoms on the left side of the box — you can find this data in the datafile ndata.d[1].

We generate the data using the same approach as in Chap. 3, but we generate more data points here to have better statistics for our analysis. The data is generated by a molecular dynamics simulation using LAMMPS to model a two-dimensional gas system using the following LAMMPS input file:

```
# 2d Lennard-Jones gas
units lj
dimension 2
atom_style atomic
lattice hex 0.10
region box block 0 20 0 10 -0.1 0.1
create_box 1 box
create_atoms 1 box
mass 1 1.0
velocity all create 2.5 87287
pair_style lj/cut 2.5
pair_coeff 1 1 1.0 1.0 2.5
neighbor 0.3 bin
neigh_modify every 20 delay 0 check no
fix 1 all nve
dump 1 all custom 10 gasstat01.lammpstrj id type x y z vx vy vz
thermo 100
run 50000
```

which is run by typing

```
lammps < in.gasstatistics01
```

We then use the script we developed in Sect. 3.4 to extract $n(t)$, the number of atoms in the left half of the system ($x < L_x/2$) as a function of time t:

```
from pylab import *
data = dump("gasstat01.lammpstrj") # Read output states
t = data.time()
nt = size(t)
nleft = zeros(nt,float) # Store number of particles
```

[1] http://folk.uio.no/malthe/fys2160/ndata.d

```
# Get information about simulation box
tmp_time,box,atoms,bonds,tris,lines = data.viz(0)
halfsize = 0.5*box[3] # Box size in x-dir
for it in range(nt):
   xit = array(data.vecs(it,"x"))
   jj = find(xit<halfsize)
   numx = size(jj)
   nleft[it] = numx
plot(t,nleft), xlabel('t'), ylabel('n'), show()
np.savetxt('ndata.d',(t,nleft))
```

The result is $n(t)$ for a sequence of timesteps t_i, as illustrated in Fig. 4.1 for $N = 210$ atoms in the box. The resulting sequence t_i, n_i is also written to the file `ndata.d` using the command `dlmwrite`.

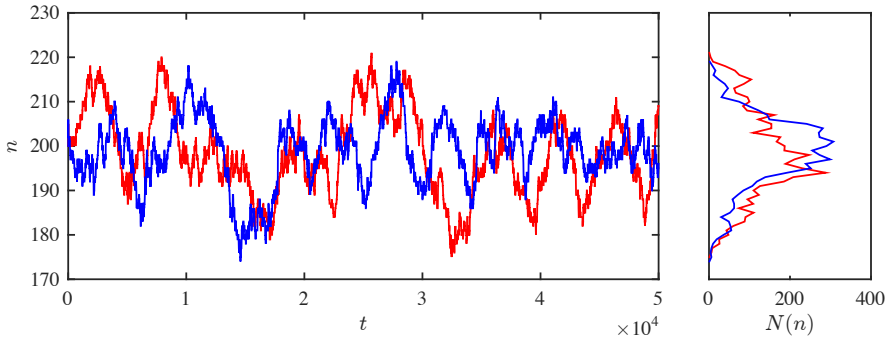

Fig. 4.1 Plot of the number of atoms on the left half (n) as a function of time (t) for 50000 timesteps for two runs with different random seeds for the initial velocity distribution.

Now, we do not need to perform the simulation many times. Instead, we can concentrate on analyzing the resulting data in data by reading and analyzing the data file `ndata.d`. The file is loaded using

```
import numpy as np
t,n = np.loadtxt('ndata.d')
```

Now, this gives us direct access to the data. But how can we understand what this data represents and what it tells us about the gas? In order to understand the data better we will develop several measurement techniques characterizing the data, and we will develop a model that may explain the data and our analysis of the data. Such a model may not reproduce all the features of the data exactly, but it should represent the main features in a good way. What do we mean by that? Hang on and you will find out! We notice that the data has some random element — at least it looks rather random. If we did another simulation, but with a different random seed for the initial velocities, we would expect the behavior to be different — at least not identical — but also to retain some of the features. This is illustrated in

Fig. 4.1, where two curves $n(t)$ are illustrated. The curves have similar features, but are clearly not identical.

What general features of $n(t)$ may we hope that a model would predict or explain? Since the curves are not identical, we must characterize properties of $n(t)$. For example, we could look at how often $n(t)$ visits a particular value of n: We count the number of times, $N(x)$, when there are x atoms on the left side, that is, how many times $n(t) = x$ for various values of x. This is done automatically by a histogram:

```
histogram(m)
```

(We will explain this method in detail below). The histogram is shown in Fig. 4.1, in the same figure as the data-set. This gives us a first insight into the frequency of occurrence of the various values of n. And we immediately observe that not all n-values occur equally frequent: values close to $n = 200$ occur more frequently that values further away from $n = 200$. Fortunately, there is a whole field of science dedicated to studying frequencies of occurrence and histograms of frequencies — the field of probability theory and statistics. We need tools from statistics to describe and model these data. This chapter therefore contains a brief introduction to statistics and probability theory illustrated by applications to the simulation data.

4.2 Probabilities

4.2.1 Statistical experiments

Measuring a sequence of number, n_i, has similarities to recording a sequence of die throws[2]. However, our knowledge of die problems is significant. Not only because may great mathematicians also were gambler, but also because a die introduces us to many fundamental questions in probability. What is a die throw? It is a **statistical experiment**:

> A **statistical experiment** is a trial with a number of possible outcomes or events. The result of an experiment is called an **outcome** or an **event** and the set of all possible outcomes is called the **sample space**.

Typically, we will consider a die to be fair, meaning that all outcomes are equally likely, and that two consequetive experiments are **uncorrelated**, meaning that if we know the result of the previous experiment, this will not help us know the result of

[2]There are also important differences between measuring $n(t_i)$ and recording dice throws: For the measurements subsequent elements may be *correlated*: If we know the value at i, the system may not have changed significantly to the measurement $i+1$. (Depending on the time between measurements). However, for a sequence of dice throws we expect the event to the *uncorrelated*.

the next experiment. (However, many physical systems, such as the gas, as **correlated**, because the state of the system may not have changed completely between two subsequent measurements).

We could consider each measurement of the number of atoms on the left side of the box, an experiment and each result of the measurement, $n_i = n(t_i)$, an outcome. Similarly, we could call throwing a die an experiment and the number appearing, m_i, an outcome. The simulation of the gas therefore results in a sequence of experiments resulting in a sequence of outcomes, n_i. How can we analyze such a sequence?

4.2.2 Generating a sequence of outcomes

We start from a simpler problem by generating a sequence of experiments, such as a die throw, numerically and then analyzing the sequence. We can think of this as performing a numerical statistical experiment using the random number generator in Python [3].

We generate a random number between 1 and 6 in Python by:

```
randint(0,6)
```

```
3
```

(Here, the first argument is the smallest random integer and the second argument is the largest random integer, and an optional third argument is the number of outcomes to produce). And we can use the same command to generate a sequence of 4 such experiments by:

```
randint(0,6,4)+1
```

```
array([1, 4, 4, 2])
```

The result is a sequence, m_i, $i = 1,\ldots,M$ of M outcomes from a numerical stastistical experiment. We are interested in understanding the experiment by analyzing the outcomes. How can we do that?

4.2.3 Measuring the frequency of occurrence

Our first approach to characterize the results from the experiment is to measure the frequency of occurrence of a particular outcome: We count how many times a given value x occurs in the sequence of number, m_i, by generating a histogram. A histogram records how many times, $N(x)$ a value x occurs in the interval from x_j to

[3] The random number generator only produces *pseudo-random* numbers based on a deterministic algorithm that generates a sequence of apparently random numbers. However, for all our purposes these numbers appear to be random.

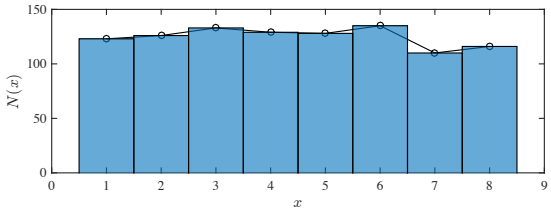

Fig. 4.2 Plot of the histogram of 1000 outcomes from an experiment where we choose a random number between 1 and 8.

$x_{j+1} = x_j + \Delta x$, where Δx is the bin width of the histogram. Python has automatic functions for finding the histogram for a data set. We generate 1000 random numbers between 1 and 8 and find the histogram using:

```
m = randint(0,8,1000)+1
[Nx,edges] = histogram(m,8,range=(0.5,8.5))
print Nx
```

```
array([127, 131, 128, 135, 116, 140, 127,  96])
```

```
print edges
```

```
[ 0.5  1.5  2.5  3.5  4.5  5.5  6.5  7.5  8.5]
```

(Notice the range used in order to ensure there are 8 bins of size 1 each). Here, the `edges` are the x_j values giving the edges of the bins used. We can find the centers of the bins and plot the results using:

```
x = (edges[1:]+edges[:-1])/2
print x
```

```
[ 1.  2.  3.  4.  5.  6.  7.  8.]
```

```
plot(x,Nx)
```

We can compare this with a traditional histogram plot using

```
hold('on')
hist(m,8,range=(0.5,8.5))
hold('off')
show()
```

The resulting plot is shown in Fig. 4.2. This provides us with our first characterization of the data from the experiments. We see that the frequencies are approximately, but not exactly, the same for all the outcomes. A similar illustration of the data from the simulation of the gas is shown in Fig. 4.1, but here the frequencies are not the same for all outcomes. How can we interpret these frequencies of occurrence?

4.2.4 Frequencies and probabilities

When we analyze a statistical experiment, or data from experiments or simulations, we will assume that the experiments reflects an underlying process which has given properties. For example, that each outcome has a given probability. The probability is a feature of the process, whereas the frequency is an observation of the outcomes from this process. We can relate frequency of occurrence and the probability of an outcome through a definition of the **the probability of an event** :

We observe a sequence of M outcomes, n_i, $i = 1,\ldots,M$, from a statistical experiment. The probability $P(n = x)$ for measuring n and finding the value x is defined as how many times, N_x, we have measured x in the sequence, divided by the number of elements, M, in the sequence:

$$P(n = x) = \frac{N_x}{M} , \tag{4.1}$$

As the number of elements grows to infinity, we expect this to approach a value which we **define** as the probability. We call this definition the **frequency definition** of probability.

Notice that this is not the only possible definition of a probability. In statistics you may meet other definitions. But it is a practical and useful definition for our purposes. Also, it reflects the important differene between the underlying properties of a process, given here by the probability for an outcome, and the observation of outcomes of that process.

4.2.5 Example: Probability of a dice

We can learn about the properties of the probability by looking at a specific example. We throw a fair dice with six sides. What is the probability to throw a 1?

Numerical experiments. First, we perform a numerical statistical experiment to gain intuition. We generate M throws of a die with a result between 1 and 6 and plot the histogram for various experiments as M becomes larger:

```
from pylab import *
Mval = array([10,100,1000,10000,100000])
nM = len(Mval)
for im in range(nM):
    M = Mval[im]
    m = randint(0,6,M)+1
    [Nx,edges] = histogram(m,6,range=(0.5,6.5))
    x = (edges[1:]+edges[:-1])/2
```

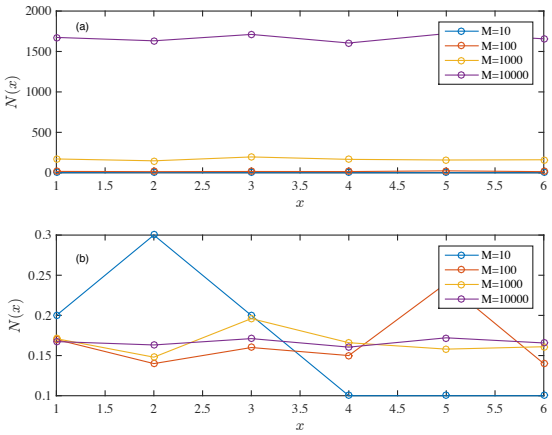

Fig. 4.3 **a** Plot of the number of occurrences, $N(x)$ for $x = 1,2,3,4,5,6$ for $M = 10, 100, 1000$, and 10000 thrwos. **b** Plot of the frequencies, $N(x)/M$ for the same data as in **a**.

```
    plot(x,Nx,'-o')
xlabel('x'), ylabel('N(x)'), show()
```

The results are shown in Fig. 4.3a.

Ooops. That did not work. This is because we have plotted $N(x)$, the number of occurrences, and this will increase when the number of experiments increase as well. Instead, we should plot the frequency of occurrence, $N(x)/M$. We do this by dividing by M: [Nx,edges] = histogram(m)/M; The resulting plot is shown in Fig. 4.3b. Now, we see that the frequencies, $N(x)/M$ clearly converges as M grows larger. We expect the frequencies to converge towards the underlying probabilities, $P(x)$. What are the values of $N(x)/M$ for $M = 10000$?

```
print Nx
```

```
array([0,1674, 0.1632, 0.1711, 0.1605, 0.1721, 0,1657])
```

Both these results, and the histogram plots in Fig. 4.3, indicate that the probability is a constant, the same, for all possible outcomes x. This is what we would expect from a *fair* dice. Otherwise, we would know that the dice preferred a particular outcome, and it would not be fair.

Theretical argument. We can also devise a theoretical argument for the value of the probability $P(x)$ for a throw of the die. From the definition of probability, the probability to get i is

$$P(i) = \frac{N_i}{M}\,, \tag{4.2}$$

where M is the number of throws and i may be 1,2,3,4,5 or 6 — valid in the limit when M becomes large. From this definition we see that probabilities are *normalized*:

$$\sum_i P(i) = \frac{\sum_i N_i}{M}\,, \tag{4.3}$$

where $\sum N_i = M$ because in each throw one and only one of the 6 possible outcomes are possible. This gives us *the normalization property of probabilities*:

$$\sum_i P(i) = 1 \,, \tag{4.4}$$

when any two outcomes i cannot occur at the same time and the sum is over all possible outcomes.

We can use this property to calculate the probability to throw a 1. This follows from two characteristics of the experiments. First, the dice is fair, which means that $P(i)$ are the same for all i, $P(i) = \text{const} = p$. Second, we use the normalization condition:

$$\sum_{i=1}^{6} P(i) = \sum_{i=1}^{6} P = p \sum_{i=1}^{6} = p \cdot 6 = 1 \;\Rightarrow\; p = \frac{1}{6} \,, \tag{4.5}$$

This example uses a standard set of tricks that you may want to remember: If all outcomes are equally probable we can assume they all have the same probability p, and the probabilities are normalized.

We can also illustrate another basic principle of probabilities that also serves as a useful trick: The probability of something *not happening* can be found from the normalization condition. For example, what is the probability of not throwing a one in a single throw?

Here, we use a trick and split all possible outcomes into $P(\text{one})$ and $P(\text{not one})$. Since one of these two outcomes must occur, they must be normalized, so that

$$P(\text{one}) + P(\text{not one}) = 1 \,, \tag{4.6}$$

where we know that $P(\text{one}) = p = 1/6$. We can therefore find $P(\text{not one})$ from

$$P(\text{not one}) = 1 - P(\text{one}) = 1 - p = 1 - \frac{1}{6} = \frac{5}{6} \,, \tag{4.7}$$

4.2.6 Properties of probabilities

From this example, we have learned several new properties of probabilities that are general:

Normalization: Probabilities are **normalized** so that the sum of the probabilities of all the possible outcomes is 1:

$$\sum_x P(x) = 1 \,, \tag{4.8}$$

when any two outcomes x cannot occur at the same time and the sum is over all possible outcomes x. This is called the **normalization condition of probabilities**.

This can be proven from the frequency definition and that the total number of observations, $\sum_x N(x)$ must be the same as the total number of experiments, M:

$$\sum_x P(x) = \sum_x \frac{N(x)}{M} = \frac{1}{M}\sum_x N(x) = \frac{M}{M} = 1 \, . \tag{4.9}$$

Inversion rule: We can always divide all the possible outcomes of an experiment into two parts, that an outcome A occurs and that the outcome A does not occurs. Either of these outcomes must occur, and they cannot occur at the same time. The normalization condition therefore gives:

$$P(\text{A}) + P(\text{not A}) = 1 \Rightarrow P(\text{A}) = 1 - P(\text{not A}) \, . \tag{4.10}$$

Sets of outcomes. Notice that outcomes from an experiment can be grouped in various ways. For example, when we throw a six-sided dice, we can group the set of all possible outcomes into two set: set A = 1,2, and set B = 3,4,5,6. These two sets are **independent** , meaning that set A cannot occur at the same time as set B, and they span the total **sample space** — the set of all possible outcomes. The normalization rule can therefore be applied to sets A and B: $P(A) + P(B) = 1$. (It is customary to write the probability for a set A as $P(A)$).

Sets do not need to be independent. For example, we could define three sets A = 1,2,3, B = 3,4 and C = 4,5,6. These sets span the sample space, but they are not independent. We can therefore not apply the normalization rule to these sets: $P(A) + P(B) + P(C) \neq 1$.

Random variable. If the outcome of an experiment is a number, such as for the throw of a die, we call the outcome a **random variable**, and usually denote it with a *capital symbol* for the random variable and a normal case sumbol for the outcome of an experiment. For example, for the throw of a die, we would use the symbol N for the number shown in the die, and the symbol n for a particle outcome. The probability to observe the outcome n when we measure the random variable N can therefore be written as the probability for N to be equal to n: $P(N = n)$.

Rule of addition. For two independent sets of outcomes, A and B, the probability that A or B occurs is given as the sum of their probabilities:

Addition rule: For two independent sets A and B, the probability for A or B is

$$P(\text{A or B}) = P(\text{A}) + P(\text{B}) \, . \tag{4.11}$$

This rule can be proven using the frequency definition of probabilities: If we perform M experiments, the number of results in set A is $N(A)$ and the number of results in set B is $N(B)$. The number of results in A or B is therefore $N(A)+N(B)$ and the probability for A or B is therefore $P(\text{A or B}) = (N(A)+N(B))/M = P(A)+P(B)$. We can extend this rule also to cases where A and B not are independent:

$$P(\text{A or B}) = P(\text{A}) + P(\text{B}) - P(\text{A and B}) \, . \tag{4.12}$$

Where the last term is zero if A and B are independent, because in that case they cannot both occur at the same time.

4.2.7 Probability for simultaneous outcomes

What would happen if we threw a die two times? How can we find the probability to throw a 1 in the first throw and a 6 in the second throw? This corresponds to asking questions about the outcomes from two experiments. We could write the probability for both of these events to occur at the same time as:

$$P\left(n_1 = 1 \text{ and } n_2 = 6\right) \, , \tag{4.13}$$

We could call the event that we get a 1 in the first throw A and the event that we get a 6 in the second throw B. We can then write the probability for both to occur as $P(A \text{ and } B)$. This is usually written at $P(A \cap B)$, using the intersection sign $\cap$ because the outcome is the intersection of A and B, the set of outcomes that include both A and B.

Numerical approach. First, we measure the probability for (A and B) numerically: We draw two random numbers, n_1 and n_2, many times, and count how many times, N, we get a $n_1 = 1$ and $n_2 = 6$ at the same time. First, we generate the numbers n_1 and n_2:

```
n1 = randint(0,6,1000)+1
n2 = randint(0,6,1000)+1
```

We can find all the values that are equal to 1 or 6 using the equal to operator

```
i1 = 1*(n1==1)
i2 = 1*(n2==6)
```

Where we have multiplied with one to ensure the result is an integer and not `True/False`. Then we find how many times $n_1 = 1$ and $n_2 = 1$ at the same time by summing `i1` and `i2`. Where this sum is equal to 2, both $n_1 = 1$ and $n_2 = 6$.

We then simply have to count how many times the sum is equal to 2 to find N, the number of times $n_1 = 1$ and $n_2 = 6$ at the same time:

```
m = 1*(i1+i2)==2
N = sum(m)
p = N/1000.0
print p
```

```
0.034
```

The result of this experiment is therefore $p = N/M = 0.034$.

Theoretical approach. How can we develop a theoretical answer for this problem? There are several approaches we can follow. We can use the same approach as we used numerically above: We can look at all the possible outcomes from the experiment which consists of throwing a dice two times. How many outcomes are there in total? The first die has 6 possible outcomes, and the second die has 6 outcomes. In total, there are $6 \times 6 = 36$ different outcomes. (Notice in particular, that the outcome where the first die is a 2 and the second is a 4 is different from the outcome where the first die is a 4 and the second die is a 2. We have to count both these). All these outcomes are equally likely, and will have the same probability, if the die is fair. This means that the probability p for any such outcome is found from the normalization rule:

$$\sum_{i=1}^{36} p = p\,36 = 1 \Rightarrow p = 1/36\,. \tag{4.14}$$

The probability for a particular outcome, $n_1 = 1$ and $n_2 = 6$ is the probability for one of these 36 outcomes, and hence the probability for this is $1/36$.

This approach is very robust. We can easily use the same approach to find the probability for the first die to be equal to 1 or 2 and the second die to be equal to 6. Can you find the argument and the answer?

Alternative theoretical approach. Notice that we could have used an alternative approach: In order to get a 1 in the first throw and then a 6 in the second throw, we see that there are 6 possibilities for the first, throw, and only 1 of these give a 1. That is, the probability to get a 1 in the first throw is $p_1 = 1/6$. Then, in the second throw, there are six possible outcomes, but only one of them give a 6. Thus the probability to get a 6 in the second throw is $p_2 = 1/6$. What is the probability to get both, that is a 1 in the first throw and a 6 in the second throw. The total number of outcomes is 6×6 and the number of outcomes that give $n_1 = 1$ and $n_2 = 6$ is 1×1. Hence, the probability for both is

$$p = \frac{1 \times 1}{6 \times 6} = \frac{1}{6}\,\frac{1}{6} = p_1\,p_2\,. \tag{4.15}$$

This indicates that the probability for (A and B) is the product of the probability for each of the events — given that the events cannot occur at the same time. (If they could occur at the same time, that is, that they are not independent, then we could not find the total number of outcomes by multiplying the number of outcomes in throw 1 and the number of outcomes in throw 2).

Rule of multiplication. This rule is indeed a general law in statistics, which we call the rule of multiplication:

> **Rule of multiplication:** For two *independent* events A and B, the probability to observe A and B at the same time is the product of the probability to observe A and the probability to observe B:
>
> $$P(\mathrm{A} \cap \mathrm{B}) = P(\mathrm{A})\, P(\mathrm{B})\,, \tag{4.16}$$

4.2.8 Example: Throwing two dice

Let us answer various questions for the results of two fair dice.

What is the probability to throw 1,2 or 3 in the first and 4,5 or 6 in the second throw?

We can use the rule of multiplication to find this. The probability to throw 1,2 or 3 in the first throw can be found from the addition rule. The outcomes 1,2 or 3 cannot occur at the same time. Hence the probability to observe 1,2 or 3 is the sum of the probability for each, $p_1 = P(1)+P(2)+P(3) = p+p+p = 1/6+1/6+1/6 = 3/6$. Similarly, the probability to throw 4,5 or 6 in the second throw is $p_2 = 3/6$. The rule of multiplication then gives that the probability for both is $p_1\, p_2 = (3/6)\,(3/6) = 9/36 = 1/4$.

What is the probability that the sum of the dice is 7?

We can use the rule of addition to find this. We need to find all the possible outcomes that add to 7. Each of the outcomes have the same probability (1/36) and they cannot occur at the same time. What outcomes add to seven? We write the outcome as two numbers where $(1,2)$ means that the first throw is a 1 and the second is a 2, whereas $(2,1)$ means that the first throw is a 2 and the second throw is a 1. The outcomes that add to 7 are $(1,6)$, $(2,5)$, $(3,4)$, $(4,3)$, $(5,2)$, and $(6,1)$, six outcomes in total. The probability that the sum is 7 is therefore $P(n_1+n_2=7) = 6\,(1/36) = 1/6$.

What is the probability that at least one of the dice show a 1?

This problem can be solved in several ways. We could count the number of outcomes (n_1,n_2) where at least one of the numbers is a 1: There are 11 such outcomes. This means that probability for at least one 1 is $12/36$. Can you think of a way to use the rule of multiplication? (Be careful to ensure that the events you are multiplying cannot occur at the same time!)

4.3 Average and standard deviation

We have now found that we can describe a random process by the probability for a given outcome. However, there are even simpler descriptions that can be measured directly from the sequence of measurements: The *average* and the *standard deviation* of the outcome of the experiment.

4.3.1 Average

For a sequence of outcomes n_i from a statistical experiment, we may ask what the typical or *average* value of the outcome would be. The most straight-forward approach is to define it as the arithmetic average, $\bar{n}$, of the outcomes:

$$\bar{n} = \frac{1}{M}\sum_{i=1}^{M} n_i \,, \tag{4.17}$$

This sum can be replaced by a sum over all possible outcomes, x, and the number of times each outcome x occurs, N_x:

$$\bar{n} = \sum_{i=1}^{M} \frac{n_i}{M} = \sum_x \frac{N_x x}{M} = \sum_x \left(\frac{N_x}{M}\right) x \,, \tag{4.18}$$

In the limit of infinitely many measurements, the value $N_x/M \to P(x)$, and we can replace the sum with a sum over probabilities

$$\langle n \rangle = \lim_{M\to\infty} \sum_x \left(\frac{N_x}{M}\right) x = \sum_x P(x)\, x \,, \tag{4.19}$$

where we have introduced the notation $\langle n \rangle$ for this asymptotic value of the average. This is indeed what we will use as the **definition of the average**:

Average of a random variable: The average $\langle n \rangle$ of a random variable n is defined as:

$$\langle n \rangle = \sum_n P(n)\, n \,, \tag{4.20}$$

where $P(n)$ is the probability to observe outcome n.

The best *estimator* for the average from a sequence of outcomes is:

Estimator for the average: The average $\langle n \rangle$ is best estimated by the estimator, $\bar{n}$:

$$\bar{n} = \frac{1}{M}\sum_{i=1}^{M} n_i \,, \tag{4.21}$$

where $\bar{n}$ approaches $\langle n \rangle$ as the number of measurements approaches infinity.

(We will discuss the difference between the *estimates* and the *real* values in Sect. 4.4 below.)

4.3.2 Average of a function $f(n)$

We can calculate the average of the outcome n directly, but we can also calculate the average of a function of n. For example, we may be interested in the average of n^2 or $\sin(n)$. In this case, we can find the average by

$$\bar{f(n)} = \frac{1}{M}\sum_{j} f(n_j) \,, \tag{4.22}$$

where we can simplify the expression by grouping outcomes into groups of values x and counting, N_x, the number of outcomes of value x. The sum can then be rewritten as:

$$\overline{f(n)} = \frac{1}{M}\sum_{x} N_x f(x) = \sum_{x} \frac{N_x}{M} f(x) \to \sum_{x} P(x)\, f(x) = \langle f(n) \rangle \,, \tag{4.23}$$

where the arrow indicates the limit when M goes to infinity. This will be our definition of the average of a function $f(n)$. While this definition may seem unmotivated now, you will see that this expression will be frequently used throughout this book.

4.3.3 Example: Average of a die throw

Let us use this to find the average value for the throw of a die. The result of a throw is the random variable n, and the average is therefore $\langle n \rangle$. We can find this from the probability $P(n)$ to observe the value n:

$$\begin{aligned} \langle n \rangle &= \sum_{n=1}^{6} P(n)\, n = \sum_{n=1}^{6} p\, n = \sum_{n=1}^{6} \frac{1}{6}\, n \\ &= \frac{1}{6}(1+2+3+4+5+6) = \frac{21}{6} = \frac{7}{2} \,. \end{aligned} \tag{4.24}$$

Notice that this is an exact result — it is not a measurement.

For a sequence of observations, n_j, $j = 1, \ldots, M$, we can estimate the average value of n by:

$$\bar{n} = \frac{1}{M}\sum_{j=1}^{M} n_i \,, \tag{4.25}$$

where the sum is over all the measurements M. For example, we can use Python to estimate the average of the sequence of die throws found in Sect. 4.2.5 using using the formula in (4.25):

```
myavg = sum(m)/len(m)
print myavg
```

```
3.5044
```

The average is implemented as the standard function `average` in Python:

```
myavg2 = average(nd)
print myavg2
```

```
3.5044
```

Notice that this measured value is close, but not exactly equal to, the theoretical value. We will return to this difference between *measured* and *exact* values in Sect. 4.4 below.

4.3.4 Standard deviation

Similarly, we may want to characterize the *variation* in the outcomes. The simplest approach would be to find the average deviation from the estimated average value:

$$\frac{1}{M}\sum_{i=1}^{M}(n_i - \bar{n}) = \left(\frac{1}{M}\sum_{i=1}^{M} n_i\right) - \bar{n} = \bar{n} - \bar{n} = 0 \,. \tag{4.26}$$

Ok, so this did not work, since the result is always zero. Instead, we may use the absolute value of the deviation:

$$\frac{1}{M}\sum_{i=1}^{M}|n_i - \bar{n}| \,. \tag{4.27}$$

This does not become zero. However, it is more usual to characterize the deviation not by the average of the deviation, but by the average of the square of the deviation, which we call the *variance*. We use the symbol $\bar{Var}(n)$ or $\bar{\sigma}^2$ for this:

$$\bar{\sigma}^2 = \frac{1}{M}\sum_{i=1}^{M}(n_i - \bar{n})^2 \,. \tag{4.28}$$

We can simplify the expression by expanding the paranthesis:

$$\begin{aligned}\bar{\sigma^2} &= \frac{1}{M}\sum_{i=1}^{M}(n_i-\bar{n})^2 = \frac{1}{M}\sum_{i=1}^{M}\left(n_i^2-2n_i\bar{n}+\bar{n}^2\right)\\ &= \frac{1}{M}\sum_{i=1}^{M}n_i^2 - 2\bar{n}\frac{1}{M}\sum_{i=1}^{M}n_i + \frac{1}{M}\sum_{i=1}^{M}\bar{n}^2 = \left(\frac{1}{M}\sum_{i=1}^{M}n_i^2\right)-\bar{n}^2\,.\end{aligned} \tag{4.29}$$

Here, we reqognize the first term as the estimate of the average of the square of n. We can rewrite this as a sum over the possible outcomes x by inserting N_x, the number of outcomes of value x. The sum is then:

$$\frac{1}{M}\sum_{i=1}^{M}n_i^2 = \frac{1}{M}\sum_x N_x x^2 = \sum_x \frac{N_x}{M}x^2 \;\rightarrow\; \sum_x P(x)x^2\,, \tag{4.30}$$

where the arrow indicated the behavior when M goes to infinity. (This is exactly the same as we would get if we calculated the average of the function $f(n)=n^2$ using the definition of the average from (4.23)).

The definition of the variance σ^2 and the standard deviation, σ, for a statistical experiment is therefore

Standard deviaton: The **variance**, σ^2, of a random variable n is defined as

$$\sigma^2 = \langle n^2\rangle - \langle n\rangle^2 = \sum_x P(x)x^2 - \left(\sum_x P(x)x\right)^2\,. \tag{4.31}$$

We can also introduce an estimator to calculate the standard deviation from on a sequence of outcomes based on (4.113). However, for subtle statistical reasons, it turns out that we should divide by $M-1$ and not by M in this sum if we use the estimated value $\bar{n}$ instead of the exact value $\langle n\rangle$ in this sum. The estimator for the standard deviation is therefore:

Standard deviation: We measure the standard deviation of a sequence of M outcomes n_i using:

$$\bar{\sigma} = \sqrt{\frac{1}{M-1}\sum_{i=1}^{M}(n_i-\bar{n})^2}\,. \tag{4.32}$$

The estimate of the variance is given as $\bar{\sigma}^2$.

4.3.5 Example: Standard deviation of a die throw

We found that for a single die the probability is $P(i) = p = 1/6$ for $i = 1, \ldots, 6$. We can find the standard deviation σ for a single throw of a die through

$$\begin{aligned}\sigma^2 &= \langle i^2 \rangle - \langle i \rangle^2 \\ &= \sum_{i=1}^{6} P(i) i^2 - \left(\sum_{i=1}^{6} P(i) i \right)^2 \\ &= \frac{1}{6}(1^2 + 2^2 + 3^2 + 4^2 + 5^2 + 6^2) - \left(\frac{7}{2} \right)^2 \\ &= \frac{91}{6} - \frac{49}{4} = \frac{35}{12} ,\end{aligned} \tag{4.33}$$

This is again an exact result. We can *measure*, that is estimate, the standard deviation directly from the 10000 dice throws directly using the formula in (4.29):

```
myavg = average(m)
mystd = sqrt(sum(m*m)/(len(m)-1.0)-myavg*myavg)
print mystd
```

```
1.7072142923487961
```

Or you can use the built-in function `std` directly

```
std(m)
```

```
1.7072142923487961
```

4.4 Difference between exact and estimated values

In statistics we often discern between the underlying, or *real*, parameters in a model and the *estimated* values of these parameters.

These concepts become clear for the throw of a six-sided die. If we throw M times and record the throws as n_i for $i = 1, \ldots, M$, we can estimate the average using the estimator:

$$\bar{n} = (1/M) \sum_{i=1}^{M} n_i \, . \tag{4.34}$$

However, the estimated value is in general not equal to the exact value for the average, which we found to be:

$$\langle n \rangle = \sum_{n} n P(n) = 7/2 \, , \tag{4.35}$$

It is good practice to discern between estimated and exact values, and we will here use a bar over the expression as a notation for estimated values.

In statistics we characterize the quality of estimators using a scheme called the maximum likelihood estimator. For the the average the maximum likelihood estimator is:

$$\bar{n} = \frac{1}{M}\sum_{j=1}^{M} n_i \ . \tag{4.36}$$

Similarly, the maximum likelihood estimator for the variance σ^2 is:

$$\bar{\sigma^2} = \frac{1}{M-1}\sum_{i=1}^{M}\left(x_i^2 - \bar{i}^2\right)^2 \ , \tag{4.37}$$

We already mentioned the $1/(M-1)$ factor in front of this equation, which appears because we are estimating two quantities, both σ^2 and $\bar{i}$, from the same set of outcomes. If we knew the exact value of the average, we would use this value in the sum and divide by M in front of the sum instead.

4.5 Toward a theory for $n(t)$ for the MD model

Now, let us use what we have learned so far to develop a theory that can describe the probability $P(n)$ to observe a particular value n in the molecular dynamics simulation of the gas/liquid system.

4.5.1 One atom model

Let us start by simplifying the system. First to one atom. In that case we guess that the atom has equal probability to be in the left or the right hand half, since the system is symmetric and we have no reason to prefer the left to the right side. Hence we would expect there only to be two possible outcomes for n, $n = 0$ and $n = 1$ with equal probability. This is just like flipping a coin (throwing a 2-sided dice). From our discussions above, we know that the probability for a fair coin to show the number n corresponds to the probability of a fair, two-sided die to show the number n:

$$P(n) = 1/2, n = 0, 1 \ , \tag{4.38}$$

where we have already seen that this theory fits well with a numerical simulation. However, this model is too simple to predict $n(t)$ for a many-particle gas. Let us instead address a two-atom system.

4.5.2 Two-atom model

We make the model slightly more advanced by looking at two *independent* atoms, 1 and 2. We will assume that each atom is described by the random variables n_1 and n_2, which is 1 if the atom is on the left side and 0 is the atom is on the right side. We assume that these two values are independent and that their probabibilities for each of the atoms are the same as for a single atom in (4.38).

Using this notation, the total number of atoms on the left side is $n = n_1 + n_2$. How can we find the probability for the possible values of n? We address this by a numerical experiment and by a theoretical argument:

Numerical approach. We study the distribution of n by performing a numerical statistical experiment. We generate two sets of random numbers, $n_1(i)$ and $n_2(i)$, for $i = 1,\ldots,M$, and add them to get n. We can then find the probabilities for various values of n using the same method as we did above. First, we generate 2 arrays n_1 and n_2 each with M elements. We do this by generating a variable with $M \times 2$ random numbers so that the first column equals n_1 and the second column equals n_2:

```
M = 1000
m = randint(0,2,(M,2))
print m[0:5,:]
```

```
[[0 1]
 [1 0]
 [0 0]
 [1 1]
 [1 0]]
```

Where we have shown the first 5 elements for n_1 and n_2. We find n by adding the columns corresponding to n_1 and n_2. That is, we want to generate a new array with M elements, where each element corresponds to the sum along each row in the `m`-matrix. This is done by the `sum` command, but we must specify that the sum should be done along the second dimension of the array, so that we sum along the rows:

```
n = sum(m,axis=1)
print n[0:5]
```

```
[1 1 0 2 1]
```

We can now find and plot the histogram:

```
M = 10000
N = 2
m = randint(0,2,(M,N))
n = sum(m,axis=1)
hist(n), show()
```

The resulting plot is shown Fig. 4.4.

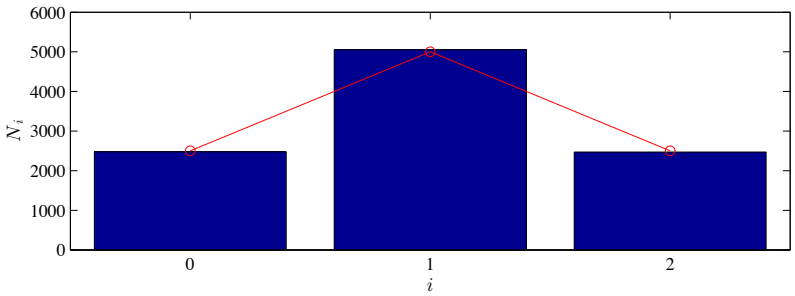

Fig. 4.4 Plot of the histogram for 10000 realizations of the sum of two coin tosses: $n = n_1 + n_2$, where $n_1 = 0, 1$ and $n_2 = 0, 1$.

Theoretical approach. We can find the probability $P(n)$ from a theoretical argument: There are four possible outcomes of (n_1, n_2), but only three possible outcomes of $n = n_1 + n_2$, as illustrated in the following table:

n_1	n_2	$n = n_1 + n_2$
0	0	0
1	0	1
0	1	1
1	1	2

If all the 4 outcomes of (n_1, n_2) are equally likely, with probability $p = 1/4$, then the probability for an outcome n is given as the number, $g(n)$ of (n_1, n_2)-states that give the same n-value, multiplied with the probability, p, per state, where $p = 1/M$, and $M = 2^2$ is the total number of (n_1, n_2)-states.

$$P(n) = g(n)p = \frac{1}{2^M} g(n) \,, \tag{4.39}$$

where the **multiplicity** $g(n)$ is read from the table above, and are tabulated in the following table:

n	$g(n)$	$P(n)$
0	1	1/4
1	2	2/4
2	1	1/4

We can compare these results directly with the histogram in Fig. 4.4. The histogram records how many experiments, N_n, that resulted in outcome n, when the total number of experiments were $M = 10000$. The probability $P(n)$ is approximately equal to N_n/M, hence we must compare $N_n = P(n)M$ with the values in the histogram, as done with the following code:

```
nn = array([0 1 2])
Pn = array([0.25, 0.5, 0.25])
Nn = Pn*M
plot(nn,Nn)
```

The resulting correspondence is excellent, as seen in Fig. 4.4.

4.5.3 Many-atom system

How can we generalize this theoretical approach to many atoms? For the two-atom system we counted the number of possible outcomes, that is, all the possible (n_1, n_2) values: $(0,0)$, $(1,0)$, $(0,1)$, $(1,1)$. There are four outcomes in total. We call such an outcome a **microstate**. There are four microstates. Each such microstate is equally probable — just like when we threw two dice. The possible states for n are $n = 0$, $n = 1$, and $n = 2$. We call these states **macrostates**. The macrostates do not have the same probability, because there may be a different number of microstates in each macrostate. To find the probability of a macrostate, we have to count how the number of microstates in a particular macrostate.

For two atoms there are four microstates in total, each with probability $1/4$. The macrostate $n = 0$ can only result from one microstate, $n = 1$ from two, and $n = 2$ from one.

This method can easily be generalized to any number of atoms. For N atoms, each microstate can be described as a sequence of N numbers, $(n_1, n_2, \ldots, n_N)$, for example $(1,0,0,1,0,1,1,0,0,0,\ldots,1)$, and n will be the sum, $n = \sum_i n_i$, of the values in the sequence. To find the probability of a macrostate, we must count how many microstates results in a particular macrostate. To do this, we use a clever trick: We realize that the value of n only depends on how many of the n_i in the microstate are 1. It does not matter where they are in the sequence. We can therefore find all microstates that give a value n by finding out how many ways we can place n ones in a sequence of N numbers. (The remaining numbers in the sequence will be zeros).

We have now mapped the problem onto a well know problem in combinatorics with a simple solution[4]. We have n ones (and $N - n$ zeros). Let us decide where to place the ones. The first one can be placed in N positions in the sequence, the second in $N - 1$ positions and so on until we reach $N - n + 1$. This results in $N!/(N - n)!$ different ways to place the ones.

However, using this method we have counted many microstates several times, because the ones are identical and therefore the order in which they are placed into the sequence is inconsequential. We can show this in the two-atom system. With the proposed counting method, we count one state where we first place a 1 at the first position ($n_1 = 1$), and then we place a one at the second position ($n_2 = 1$). This is the state $(1,1)$. However, we also count a state where we first place a 1 at the second position ($n_2 = 1$) and then we place a one at the first position ($n_1 = 1$),

[4]This is a common approach in physics that you will meet many times in your career as a physicist.

giving the state $(1,1)$. But these two states are identical! And here we have counted these states twice. We must therefore correct for this by dividing by the number of times we have counted the same state. For a sequence with n ones, such as for this sequence with 4 ones: $(1,0,1,1,0,1)$, we must divide by all the possible ways we could have generated this state, which corresponds to all the possible ways we can organize n numbers, which is $n!$. This means that the total number of microstates is

$$\Omega(n,N) = \frac{N!}{(N-n)!n!} = \binom{N}{n} , \tag{4.40}$$

where we have introduced the multiplicity $\Omega(n,N)$ and the usual notation used for this combinatorical result.

The total number of microstates is 2^N, and all these microstates are equally likely, so that the probabilty for n is

$$P(n) = \binom{N}{n} \frac{1}{2^N} = \frac{N!}{n!(N-n)!} \frac{1}{2^N} . \tag{4.41}$$

4.5.4 Comparing theory and simulation

We have now developed a theoretical description of the system, and the theory provides a prediction for the distribution of n — the number of atoms in the left half of the system — given the assumption that the position of each atom is independent of all the position of all the other atoms.

Reading data from simulations. Let us compare the theory directly with the measurements. First, we compare the simulated result from 5001 i-values, n_i, (from a simulation of 50000 time-steps in in.gasstatistics01[5]) and $N = 400$ atoms. The resulting trajectory gasstat01.lammpstrj[6] is analyzed by the script gasplotnt01.py[7]. You can find the data in the file ndata01.d[8], which was generated by the script. We read this data, extract the n-values, and plot $n(t)$

```
from pylab import *
t,n=loadtxt('ndata01.d');
plot(t,n), show()
```

The resulting plot of $n(t)$ is shown in the top of Fig. 4.5.

Estimating the probability distribution $P(n)$. We estimate the probability distribution $P(n)$ for the n using the histogram function: We count the number of times, N_n, that $n(t) = n$, and then divide by the total number of observations, M (called `nt`

[5] http://folk.uio.no/malthe/fys2160/in.gasstatistics01

[6] http://folk.uio.no/malthe/fys2160/gasstat01.lammpstrj

[7] http://folk.uio.no/malthe/fys2160/gasplotnt01.py

[8] http://folk.uio.no/malthe/fys2160/ndata01.d

in the code):

$$P(n) \simeq \frac{N_n}{M} \,. \tag{4.42}$$

The results depend on the number of bins used for the histogram. We can display the histogram with 10 and 20 bins using

```
hist(n,bins=10)
show()
hist(n,bins=20)
show()
```

The resulting histograms are shown in Fig. 4.5. Notice that the number of counts in each bin goes down when the number of bins is increased. These histograms only measure the number of outcomes in each bin, where the bin may span over a size Δn corresponding to many n-values. However, we would like to estimate $P(n)$ as N_n/M, where N_n is the number of outcomes that are equal to n. This can be done either by ensuring that the bin width is n, or we need to divide by the bin size in order to find the estimate for the probability $P(n)$ across the bin: $P(n) \simeq N(n,n+\Delta n)/(M\Delta n)$. Why do we need to divide by Δn? If we count $N(n,n+\Delta n)$, we have really counted

$$N(n,n+\Delta n) = N(n)+N(n+1)+N(n+2)+\ldots+N(n+\Delta n) \,. \tag{4.43}$$

If we now divide by M, the total number of outcomes, we get

$$\begin{aligned}\frac{N(n,n+\Delta n)}{M} &= \frac{N(n)}{M}+\frac{N(n+1)}{M}+\frac{N(n+2)}{M}+\ldots+\frac{N(n+\Delta n)}{M} \\ &\simeq P(n)+P(n+1)+P(n+2)+\ldots+P(n+\Delta n) \\ &= \Delta n\,\frac{1}{\Delta n}\left(P(n)+P(n+1)+P(n+2)+\ldots+P(n+\Delta n)\right) = \Delta n\,\overline{P(n+\Delta n/2)} \,.\end{aligned} \tag{4.44}$$

We therefore need to divide by Δn to find the average probability over the interval from n to $n+\Delta n$, and we need to find the centers of the bins, $n+\Delta n/2$, in order to plot the data in this point. We can calculate the probabilities and the bin centers directly from the output from the `histogram`-function:

```
Nn,edges=histogram(n,bins=10)
ncenter = 0.5*(edges[1:]+edges[:-1])
dn = edges[1:]-edges[:-1]
plot(ncenter,Nn/dn/nt)
```

Comparison with theory — numerical estimates. We now have the tools needed to compare the observed results from the molecular dynamics simulations with our simple theory for $n(t)$. The number of observations from the molecular dynamics simulations is 5001. To compare with the observed results, we generate set of possible values of n for $N = 400$ atoms using the theory we have developed so far: Each of the N atoms can either be on the left side ($X_i = 1$) or on the right side ($X_i = 0$) with equal probability, and we count the total number of atoms, n, on the left side

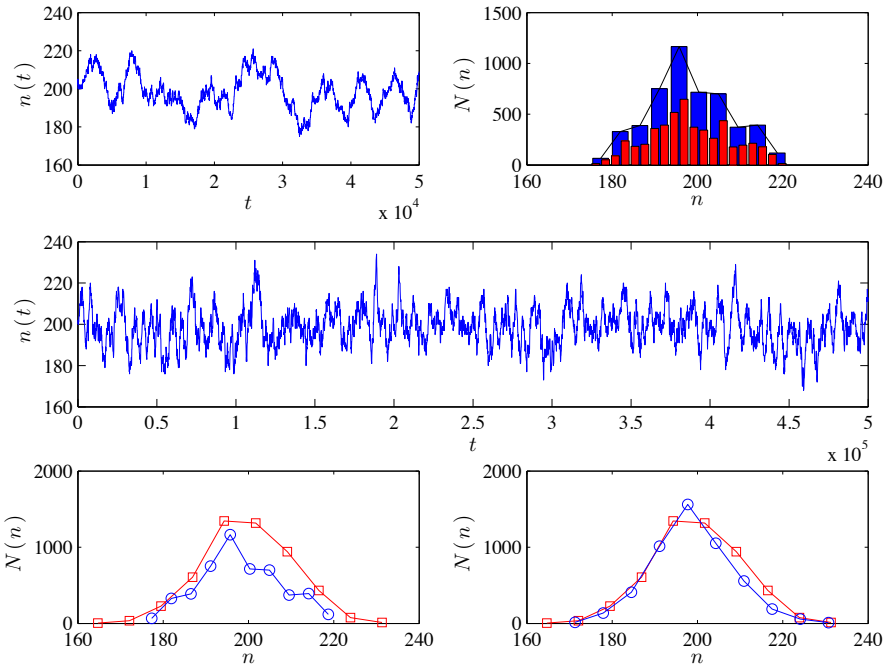

Fig. 4.5 Plot of $n(t)$ from the simulation and n_i from 5001 random microstates from 50,000 timesteps (top left) and 500,000 timesteps (middle). (Top right) Histogram of $n(t)$ for 50,000 timesteps with 10 and 20 bins. (Bottom left) Histogram of $n(t)$ for 50,000 timesteps (blue) and for a simulation with 5001 random outcomes using the binomial model (red). (Bottom right) Histogram of $n(t)$ for 500,000 timesteps (blue) and for a simulation with 5001 random outcomes using the binomial model (red).

$$n = \sum_{i=1}^{N} X_i \,, \tag{4.45}$$

We generate 5001 such values randomly and compare the two histograms:

```
ni = randint(0,2,(5001,400))
nt = sum(ni,axis=1)
hist(nt), hist(n), show()
```

The resulting plot is shown in Fig. 4.5. Hmmm. This did not look like a very good fit. What has gone wrong? It may be that the initial correlations in the data were too strong, so that not all the atoms really are active and can change places between each measurement from the simulation. Let us increase the time interval between measurements of n_i, but keep the number of observations the same. We rerun with

the input file in.gasstatistics02[9] resulting in gasstat02.lammpstrj[10], producing the data file ndata02.d[11], which we plot using

```
from pylab import *
t,n=loadtxt('ndata02.d');
ni = floor(rand(5001,400)*2)
nt = sum(ni,axis=1)
hist(nt), hist(n), show()
```

The resulting sequence $n(t)$ of n-values is shown in Fig. 4.5. We see that the correlations in the $n(t)$ signal are now less prominent, although there are still some correlations present. However, the histograms in the bottom right part of Fig. 4.5 now show much better correspondence, indicating that the theory we have developed provides a good explanation of the behavior. Indeed, the correspondence between observations and theory is surprisingly good given the simplicity of the model. We may therefore conclude that the model captures the most important features of the behavior of $n(t)$.

Comparison with theory — calculated probabilities. Now, let us also compare directly with the probabilities we have calculated for n:

$$P(n) = \frac{N!}{n!(N-n)!} 2^{-N} , \tag{4.46}$$

We use the `scipy.stats.binom.pmf` function to calculate $P(n)$ as a function of n. (This uses the scipy[12] package). In order to compare directly with the histogram, we notice that $P(n) = N_n/M$, and therefore $N_n = MP(n)$. We also choose many bins (301) to ensure that each value of n falls into one and only one bin, so that $P(n) = N_n/M$, where N_n is the number of outcomes in the bin that includes n. This is done by the following script:

```
import scipy, scipy.stats
M = 5001;
Nn,edges=histogram(n,bins=10)
ncenter = 0.5*(edges[1:]+edges[:-1])
dn = edges[1:]-edges[:-1]
xn = scipy.linspace(0,400,401)
pmf = scipy.stats.binom.pmf(xn,400,0.5)
plot(ncenter,Nn/dn/M,xn,pmf), show()
```

The resulting plot is shown in Fig. 4.6. The correspondence is not spectacular, but decent, showing that the theory we have developed gives a reasonably good description of $n(t)$ from the molecular dynamics simulations.

[9] http://folk.uio.no/malthe/fys2160/in.gasstatistics02

[10] http://folk.uio.no/malthe/fys2160/gasstat02.lammpstrj

[11] http://folk.uio.no/malthe/fys2160/ndata02.d

[12] http://www.scipy.org

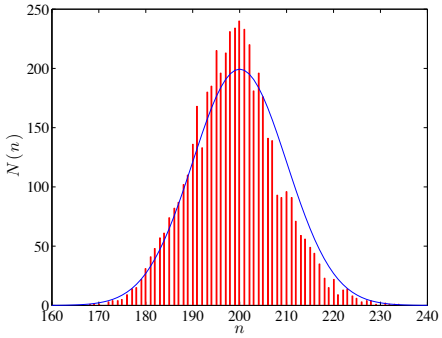

Fig. 4.6 Plot of the histogram for $n(t)$ from the simulation and n_i from a binomial distribution.

4.6 The binomial distriubtion

The binomial distribution fits well to the observed data for the molecular simulation. Indeed, the binomial distribution provides a good description for any sum of independent events, such as the sum of several dice, the results of many flips of a coin, or the behavior of the number of atoms in a part of a gas.

In general, the binomial distribution describes the case where we have a sequence of N *identical*, *independent* trials with discrete outcomes X_i, $i = 1,\ldots,N$ – often called a *Bernoulli process*. The simplest case is when X_i is binary, such as either 0 or 1. We have looked at a fair coin, but even when the coin is not fair, such as if the case if we have probability p for $X_i = 1$ and $q = 1 - p$ for $X_i = 0$, the results are described by the binomial distribution.

Binomial distribution: For N independent random variables, X_i, with the same distribution of outcomes, the sum $Z = \sum_{i=1}^{N} X_i$, is described by a binomial distribution, $P(z)$:

$$P(Z = z) = \binom{N}{z} p^z q^{N-z} , \tag{4.47}$$

This formula reproduces our results from above for a fair dice with $p = q = 1 - p = 1/2$.

4.6.1 Example: Coin toss

We can use the binomial distribution to answer questions about coin tosses. Let us assume we flip a fair coin, so that $p = q = 1/2$.

(a). If we flip the coin N times, what is the probability that all the flips are ones?

Getting all the flips to be ones corresponds to $n = N$. We can use the formula to find the probability for this:

$$P(N,N) = \frac{N!}{N!(N-N)!}2^{-N} = 2^{-N} \,. \tag{4.48}$$

(b). If we flip the coin $N = 5$ times, what is the probability to get exactly 3 heads? Again, we can apply the formula directly, now with $n = 3$ heads and $N = 5$ flips:

$$P(N = 5, n = 3) = \frac{5!}{3!(5-3)!}2^{-5} = \frac{5}{16} \,. \tag{4.49}$$

4.6.2 Properties of the binomial distribution

We can gain intuition about the binomial distribution by studying numerical experiments of $N = 10$, 20, 40, and 80 trials when X_i is 0 or 1 with equal probability, and then measuring $Z = \sum_i X_i$. This is done by generating a set of M random sequences of N steps:

$$X_{i,j} \begin{cases} 0 & (1-p) \\ 1 & p \end{cases} \quad i = 1,\ldots,N \,,\; j = 1,\ldots,M \,, \tag{4.50}$$

and

$$z_j = \sum_{i=1}^{N} X_{i,j} \,,\; j = 1,\ldots,M \,, \tag{4.51}$$

This gives M outcomes z_j, and we can then collect statistics about these outcomes. We generate M outcomes z_j of length N in the array `z` using

```
xi = randint(0,2,(M,N))
z = sum(xi,axis=1)
```

Again, we use histograms to estimate $P(z) \simeq N_z/M$, where N_z is how many times we observed z in M outcomes:

```
from pylab import *
Nvalues = array([10,20,40,80]) # Number of flips
M = 100000 # Number of samples
for ival in range(0,size(Nvalues)):
    print ival
    N = Nvalues[ival]
    ni = randint(0,2,(M,N))
    nt = sum(ni,axis=1)
    y,binEdges=histogram(nt,bins=100)
    bincenters = 0.5*(binEdges[1:]+binEdges[:-1])
    j = find(y>0)
    plot(bincenters[j],y[j]/(1.0*M),'-o')
xlabel('n')
ylabel('P(n)')
show()
```

We have used 100 bins. This is more than the number of possible values, which span from 0 to 80 for the case where $N = 80$ and over a smaller range when N is smaller. This was done to ensure that each bin only contains one possible outcome. Now, if we use 100 bins for the case when $N = 20$, there are only 21 possible outcomes, and many of the bins will be empty. We therefore only plot the values in the bins that contain at least one outcome. This is done by the `find` function, which returns an array with indicies for all the element where the histogram is non-zero. The resulting plots of the estimated probabilities for $N = 10, 20, 40, 80$ are shown in Fig. 4.7.

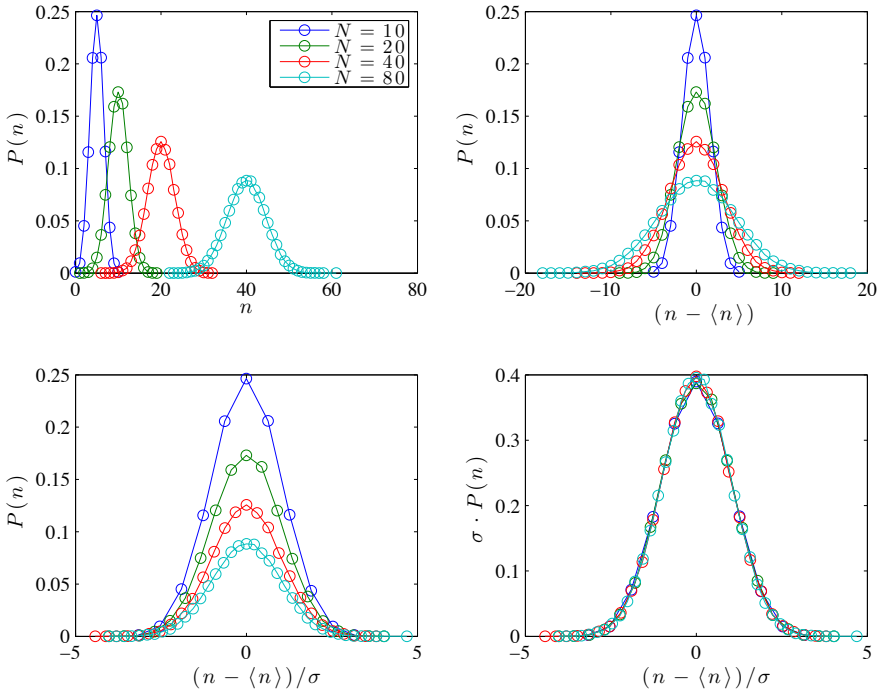

Fig. 4.7 Plot of the $P(n)$ as a function of n for $N = 10, 20, 40$, and 80.

4.6.3 Average and variation of the binomial distribution

From the plots we see that both the average value and the width of $P(z;N)$ increases with N. Let us see how this compares with the theoretical values for the average and the standard deviation of the binomial distribution. We can calculate the theoretical average of the binomial probability distribution using the definition of the average:

$$\begin{aligned}\langle n\rangle &= \sum_{n=0}^{N} nP(N,n) = \sum_{n=0}^{N} \binom{N}{n} np^n q^{N-n} \\ &= p\frac{d}{dp}\sum_{n=0}^{N} \binom{N}{n} p^n q^{N-n} \\ &= p\frac{d}{dp}(p+q)^N = pN(p+q)^{N-1} = Np\ ,\end{aligned} \tag{4.52}$$

Similarly, you can use the same approach to show that the variation is:

$$\langle (n-\langle n\rangle)^2\rangle = Npq\ , \tag{4.53}$$

We test these theoretical predictions directly on the simulated results. First, let us measure the deviation from the average, $n-\langle n\rangle = n-Np$, and find the histogram as a function of the average. As we see in Fig. 4.7 the distribution now appears symmetric around $n-Np=0$, which validates the theory for the average.

What about the width of the distribution? The theoretical prediction is that the width is $\sigma = \sqrt{Npq}$. We can test this by rescaling the n-scale with σ by plotting $P(n)$ as a function of $(n-\langle n\rangle)/\sigma$ where $\sigma = \sqrt{Npq}$. This resulting plot is shown in Fig. 4.7. The widths in these figures are correct, but the heights are now different. Experimentation shows that we can rescale the height also by multiplying with σ as shown in Fig. 4.7.

Can we understand how we arrived at this particular way of rescaling the distributions? Yes – this we can understand by addressing the continuum limit of the distribution.

4.6.4 Continuum limit of the binomial distribution

The binomial distribution is valid for discrete values z. However, as N becomes large, the range of possible outcomes for the sum becomes wide and we may ask what the limit of the distribution is for large N?

In this case, we need an approximate expression for $N!$ when N is large. This is given by Stirling's approximation, which is

$$\ln N! \simeq \ln\sqrt{2\pi N} + N(\ln N - 1)\ , \tag{4.54}$$

The second term contains the factor N, which typically will be much larger than the $\ln N$ term in the first factgor. We will therefore often only include the second $N\ln N - N$ term, but here we will also include the smaller first factor.

Here, we will prove that when N becomes large, the binomial distribution approaches that of a Gaussian or a Normal distribution

$$P(N,n) = \frac{N!}{n!(N-n)!}2^{-N} \simeq \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{((N/2)-n)^2}{2(N/4)}}\ . \tag{4.55}$$

We will provide you with two ways to approach this derivation, using purely analytical techniques and using a combination of analytical and symbolic methods.

4.6.5 Continuum limit — Analytical derivation

We will study the binomial distribution when n is close to its average value $\langle n\rangle = Np = N/2$. To simplify the notation we will here use $\langle n\rangle = \bar{n}$. We introduce the relative deviation from the average h, defined as

$$h = \frac{n-\bar{n}}{\bar{n}} \,, \tag{4.56}$$

where we will look at the case when n is close to its average so that h is small. We solve for n:

$$\bar{n}h = n-\bar{n} \;\Rightarrow\; n = \bar{n}+\bar{n}h = \bar{n}(1+h) \;. \tag{4.57}$$

Similarly, we can express $N-n$ in terms of $\bar{n}$ and h:

$$N-n = N-\bar{n}(1+h) = 2\bar{n}-\bar{n}-\bar{n}h = \bar{n}(1-h) \;. \tag{4.58}$$

We can use these terms to simplify the binomial distribution

$$P(N,n) = \frac{N!}{n!(N-n)!}2^{-N} = \frac{N!}{(\bar{n}[1+h)]![\bar{n}(1-h)]!}2^{-N} \,. \tag{4.59}$$

Let us now look at $\ln P(N,n)$ and gradually introduce Stirling's formula for the faculty functions

$$\ln P(N,n) = \ln(2\pi N)^{1/2} + N\ln N - N \tag{4.60}$$

$$-\ln(2\pi n)^{1/2} - n\ln n + n \tag{4.61}$$

$$-\ln(2\pi(N-n))^{1/2} - (N-n)\ln(N-n) + (N-n) \tag{4.62}$$

$$-N\ln 2 \,, \tag{4.63}$$

where we notice that the terms

$$-N+n+(N-n) = 0 \,, \tag{4.64}$$

cancel. As you do more of these approximations by youself, you will notice that this is always the case when you apply Stirling's formula to the binomial formula.

Let us look at the terms that include the 2π factor, and introduce $n = \bar{n}(1+h)$ and $(N-n) = \bar{n}(1-h)$. Notice that we have still not made any other approximations than to apply Stirling's formula.

$$\ln(2\pi N)^{1/2} - \ln(2\pi n)^{1/2} - \ln(2\pi(N-n))^{1/2} \tag{4.65}$$

$$= \frac{1}{2}\ln(2\pi N) - \frac{1}{2}\ln(2\pi\bar{n}(1+h)) - \frac{1}{2}\ln(2\pi\bar{n}(1-h)) \tag{4.66}$$

$$= \frac{1}{2}\ln\frac{2\pi N}{2\pi\bar{n}(1+h)2\pi\bar{n}(1-h)} \tag{4.67}$$

$$= \frac{1}{2}\ln\frac{N}{2\pi\bar{n}^2(1-h^2)} \,. \tag{4.68}$$

Now we assume that $h \ll 1$ and we insert that $\bar{n} = Np = N/2$, getting

$$\frac{1}{2}\ln\frac{N}{2\pi\bar{n}^2(1-h^2)} \simeq \frac{1}{2}\ln\frac{N}{2\pi\bar{n}^2} = \frac{1}{2}\ln\frac{N}{2\pi(N/2)^2} \,, \tag{4.69}$$

where we will not insert $\sigma^2 = Npq = N/4$:

$$\frac{1}{2}\ln\frac{1}{2\pi(N/4)} = \ln\frac{1}{\sqrt{2\pi\sigma^2}} \,. \tag{4.70}$$

We have then been able to simplify $\ln P(N,n)$ significantly:

$$\ln P(N,n) = \ln\left(2\pi\sigma^2\right)^{\frac{1}{2}} + N\ln N - n\ln n - (N-n)\ln(N-n) - N\ln 2 \,. \tag{4.71}$$

Now, let us simplify further by introducing $n = \bar{n}(1-h)$ and $(N-n) = \bar{n}(1+h)$. Let us first simplify the $-n\ln N - (N-n)\ln(N-n)$ terms:

$$-n\ln n - (N-n)\ln(N-n) \tag{4.72}$$

$$= -\bar{n}(1+h)\ln\bar{n}(1+h) - \bar{n}(1-h)\ln\bar{n}(1-h) \tag{4.73}$$

$$= -\bar{n}(1+h)\left(\ln\bar{n} + \ln(1+h)\right) - \bar{n}(1-h)\left(\ln\bar{n} + \ln(1-h)\right) \tag{4.74}$$

$$= -2\bar{n}\ln\bar{n} - \bar{n}(1+h)\ln(1+h) - \bar{n}(1-h)\ln(1-h) \tag{4.75}$$

$$= -2\frac{N}{2}\ln\frac{N}{2} - \bar{n}(1+h)\ln(1+h) - \bar{n}(1-h)\ln(1-h) \tag{4.76}$$

$$= -N\ln N + N\ln 2 - \bar{n}(1+h)\ln(1+h) - \bar{n}(1-h)\ln(1-h) \,. \tag{4.77}$$

Here, we see that the terms $-N\ln N + N\ln 2$ cancel the terms $N\ln N - N\ln 2$ in $\ln P(N,n)$. We are left with the last two terms. Now, we assume that $h \ll 1$, and we can therefore use that $\ln(1+h) \simeq h$ and $\ln(1-h) \simeq -h$, getting:

$$-\bar{n}(1+h)\ln(1+h) - \bar{n}(1-h)\ln(1-h) \tag{4.78}$$

$$\simeq -\bar{n}(1+h)(h) - \bar{n}(1-h)(-h) \tag{4.79}$$

$$= -2\bar{n}h^2 \,. \tag{4.80}$$

We can now put all this together into $\ln P(N,n)$, getting

$$\ln P(N,n) = \ln\frac{1}{\sqrt{2\pi\sigma^2}} - 2\bar{n}h^2 \,, \tag{4.81}$$

where we insert $h = (n-\bar{n})/\bar{n}$ in the last term:

$$-2\bar{n}h^2 = -2\bar{n}\left(\frac{n-\bar{n}}{\bar{n}}\right)^2 = -2\frac{(n-\bar{n})^2}{\bar{n}} \,. \tag{4.82}$$

We insert for $\bar{n} = Np = (N/2) = 2(N/4) = 2\sigma^2$:

$$\ln P(N,n) = \ln\frac{1}{\sqrt{2\pi\sigma^2}} - 2\frac{(n-\bar{n})^2}{2\sigma^2} \,, \tag{4.83}$$

If we now insert $N/4 = Npq = \sigma^2$, we see that the result of all of these calculations are that:

$$P(N,n) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{s^2}{2(N/4)}} = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{s^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{1}{2}\left(\frac{s}{\sigma}\right)^2} \,, \tag{4.84}$$

where

$$s = \frac{N}{2} - n = \bar{n} - n \,. \tag{4.85}$$

We can therefore rewrite the result in an even more general form:

$$\boxed{P(N,n) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{1}{2}\left(\frac{n-\bar{n}}{\sigma}\right)^2} \,.} \tag{4.86}$$

We now use this result to understand the scaling we found to work in fig. 4.7. We see that if we rewrite the equation as

$$P(N,n)\sigma = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{n-\bar{n}}{\sigma}\right)^2} \,, \tag{4.87}$$

we would expect all the data to fall onto a common, universal curve – given as the Gaussian distribution – as we indeed observe in Fig. 4.7.

4.6.6 Continuum limit — Symbolic derivation

Also when we use symbolic tools, we need both to have a good mathematical intuition, and we often also need to rewrite and simplify expressions ourselves. In particular, we often need to interpret limits, such as the limit of large N, ourselves using our mathematical experience and intuition.

First, we realize that the general expression for $P(N,n)$ is difficult to work with. It is simpler to work with some variable s, which is close to zero, instead of the variable n which is close to $N/2$, since we know from experience that this makes the algebra simpler when we derive a Taylor expansion around $s = 0$. We therefore

introduce $s = (N/2) - n$, and rewrite the binomial probability as

$$P(N,n) = \frac{N!}{(N/2-u)!\,(N/2+u)!} 2^{-N} \,. \tag{4.88}$$

Also, we know from experience that with all these factors, it may be simpler to work with the logarithm of the probability, $\ln P(N,n)$. We introduce these quantities symbolically

```
from sympy import *
u = symbols('u')
N = symbols('N')
P = factorial(N)/(factorial((N/2)-u)*factorial((N/2)+u))*2**N
lnP = log(P)
```

We are then ready to start symbolic manipulation. First, we would like to Taylor expand `lnP` around $u = 0$. We start from the lowest orders (second order):

```
TlnP = series(lnP,u,n=2)
print TlnP
```

```
log(2**N*factorial(N)/(factorial(N/2))**2) + O(u**2)
```

Ooops. There is no u-dependence! We know from experience that this typically means that the first order terms cancel. We need higher order terms:

```
TlnP = series(lnP,u,n=3)
print TlnP
```

```
log(2**N*factorial(N)/(factorial(N/2))**2) + ...
u**2*(-gamma(N/2 + 1)*polygamma(0, N/2 + 1)**2/factorial(N/2) - ...
gamma(N/2 + 1)*polygamma(1, N/2 + 1)/factorial(N/2) + ...
gamma(N/2 + 1)**2*polygamma(0, N/2 + 1)**2/(factorial(N/2))**2) + ...
O(u**3)
```

A bit messy. Let us simplify:

```
simplify(TlnP)
```

```
log(2**N*factorial(N)/(factorial(N/2))**2) - ...
u**2*polygamma(1, N/2 + 1) + O(u**3)
```

That is more like it! Now, we use our mathematical insight to simplify further. We realize that N is a large number and that $N/2+1 \simeq N/2$. The `polygamma` is the derivative of the digamma function. We can look up the digamma-function, $\Psi(x)$, finding that it can be approximated as

$$\Psi(x) \simeq \ln x - (1/2x) - (1/12x^2) + \ldots \tag{4.89}$$

In the limit of large x we can use $\Psi(x) \simeq \ln x$. The notation `polygamma(1,N/2+1)` means the first (hence the number 1) derivative of the digamma-function. The

`polygamma(1,N/2+1)`-term is therefore approximately $d\ln x/dx = 1/x$. We insert $x = N/2$, getting

$$\ln P = \ln(1/2)^N + \ln\frac{N!}{(N/2)!\,(N/2)!} - \frac{u^2}{(N/2)}\ . \tag{4.90}$$

This expression already shows us how P depends on u:

$$P(N,u) = C(N)\,e^{-\frac{u^2}{(N/2)}}\ . \tag{4.91}$$

Where the prefactor $C(N)$ is

$$\ln C(N) = \ln(1/2)^N + \ln\frac{N!}{(N/2)!\,(N/2)!} = -N\ln 2 + \ln N! - 2\ln(N/2)!\,. \tag{4.92}$$

In order to sort out the prefactor, $C(N)$, we need to use Stirling's approximation $\ln x! \simeq \ln\sqrt{2\pi x} + x\ln x - x$, which is valid for large x. We apply Stirling's approximation to the prefactor:

$$\begin{aligned}\ln C(N) &= -N\ln 2 + \ln N! - 2\ln(N/2)! \\ &\simeq -N\ln 2 + \ln\sqrt{2\pi N} + N\ln N - N \\ &\quad - 2\left(\ln\sqrt{2\pi(N/2)} + (N/2)\ln(N/2) - (N/2)\right)\ .\end{aligned} \tag{4.93}$$

This is again well suited for symbolic simplification. Notice the use of `sym(2)` to ensure that Python does not insert an approximate value for $\log(2)$:

```
lnC= -N*log(sym(2))+log(sqrt(2*pi*N))+N*log(N)-N- ...
      2*(log(sqrt(2*pi*(N/2)))+(N/2)*log(N/2)-(N/2))
simplify(lnC)
```

```
-log(sqrt(N)) - log(pi)/2 + log(2)/2
```

This is as good as it gets. We see that $\ln C = -(1/2)\ln(\pi N/2) = -\ln\sqrt{2\pi N/4}$. We insert this prefactor into $P(N,u)$ in (4.91), getting

$$P(N,u) = \frac{1}{\sqrt{2\pi N/4}}e^{\frac{-u^2}{2(N/4)}}\ . \tag{4.94}$$

This result can be generalized by realizing that $\bar{n} = N/2$ and $\sigma^2 = N/4$. We can therefore write u as $u = \bar{n} - n$, and $P(N,n)$ as

$$P(N,n) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{1}{2}\left(\frac{\bar{n}-n}{\sigma}\right)^2}\ . \tag{4.95}$$

This is exactly the same result as we found above, but now derived with a combination of analytical and symbolic tools.

4.6.7 Probabilities and probabilities densities

There is a small, but very important, distinction between the $P(N,n)$ we introduced for the bionomial distribution, and the $P(N,n)$ we have introduced now, because the new version is defined for all possible values of n, not only for discrete ones. This means that the probability to observe a specific value for n is really zero. We can only specify the probability to observe n in some interval from n to $n+dn$:

$$P(n \text{ is between } n \text{ and } n+dn) = P(N,n)dn \,, \tag{4.96}$$

We now realize that our notation is confusing and we should clean it up. Let us introduce the notation that the **random variable** Z is the result of a binomial experiment. The probability for the observed value of Z, z, to be in the range from n to $n+dn$ is then written as

$$P(n < Z < n+dn) = f_Z(n)dn \,, \tag{4.97}$$

where we call $f_Z(z)$ the **probability density** for the random variable Z.

Notice that it is the probability density $f_Z(z)$ that is normalized, but instead of summing we need to calculate the integral:

$$\int_{-\infty}^{\infty} f_Z(z)dz = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2} du = 1 \,, \tag{4.98}$$

where normalization is ensured by the prefactor.

Notice that $f_Z(z)$ is a function of $\overline{n}$ and σ and not of N anymore.

4.7 Universal distributions

The result we have found now for the Binomial distribution is general. Indeed, the sum of any set of independent, random variables will always follow a normal distribution, independently of the underlying distribution of ench individual part of the sum. This is a fascinating result with wide ramifications: The behavior of the sum is independent of the behavior of the individual components. The collective behavior is general and not dependent on the details. This is a general feature in statistical systems — that behaviors are universal and independent of the details — and a central part of physics is to discover what types of universal exist and what universality class a given process belongs to. Here, we will look at two types of universal behaviors: The normal distribution and extreme value statistics.

4.7.1 Central limit theorem

The result we found for the binomial distribution is very general — it is a result of a general theorem called the Central Limit Theorem: For any sequence of independent, identically distributed random variables X_i, the sum of the variables

$$Z = \sum_{i=1}^{N} X_i \,, \tag{4.99}$$

is described by a Gaussian or **Normal** distribution when N becomes large as long as the standard deviation σ_x of each X_i if finite. The random variable Z has an average $\mu = N\overline{X_i}$ and a standard deviation $\sigma = N\sigma_x$ and its probabilty density is

$$f_Z(z) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{1}{2}\left(\frac{Z-\mu}{\sigma}\right)^2} \,. \tag{4.100}$$

This result is valid for all possible values of the underlying variables X_i as long as they are independent and have a finite standard deviation.

4.7.2 Extreme value statistics

The central limit theorem describes the behavior of the average value of a sequence of identical variables, X_i. However, we can also ask other questions, such as what is the minimal or the maximum value of the sequence of N elements. This would for example be relevant if we were interested in the stength of a chain — the strength would be determined by the element with the smallest strength. What is the distribution of such elements?

We could address this numerically. We could generate a set of elements that are uniformly distributed between 0 and 1, find the minimum of this, and then perform this experiment many times, and characterize the resulting distribution.

(Not yet written)

4.8 Common probability distributions

4.8.1 The probability density and the cumulative distribution

The probability desity $f_Z(z)$ for a random variable Z gives the probability for Z to be in the range from z to $z+dz$:

$$P(z < Z < z+dz) = f_Z(z)dz \,. \tag{4.101}$$

This means that the probability for Z to be smaller than z, $P(Z \leq z) = F_Z(z)$ is given by the sum of all possible z values smaller than z:

$$P(Z \leq z) = F_Z(z) = \int_{-\infty}^{z} f_Z(z)dz \,. \tag{4.102}$$

The function $F_Z(z) = P(Z \leq z)$ is called the **cumulative distribution function** of Z. We can find the probability density $f_Z(z)$ from the derivative of the cumulative distribution function:

$$f_Z(z) = \frac{d}{dz} F_Z(z) \,. \tag{4.103}$$

The probability for Z to be larger than z is then

$$P(Z > z) = 1 - P(Z \leq z) = 1 - F_Z(z) \,. \tag{4.104}$$

In many cases it can be simpler to first calculate either $F_Z(z)$ or $1 - F_Z(z)$ and then find the probability density $f_Z(z)$ through derivation.

4.8.2 Uniform distribution

The uniform distribution describes a random process with values between a and b so that the probability $P(z, z+dz) = f_Z(z)dz$ does not depend on the value of z as long a z is between a and b.

This means that $f_Z(z) = c$ is a constant. We can find the constant from the normalization condition of the density:

$$\int_a^b f_Z(z)dz = c(b-a) = 1 \;\Rightarrow\; c = \frac{1}{b-a} \,. \tag{4.105}$$

This distribution:

$$f_Z(z) = \begin{cases} 0 & z < a \\ \frac{1}{b-a} & a \geq z \leq b \\ 0 & b < z \end{cases} \,, \tag{4.106}$$

is called the **uniform distribution**.

The average value a random variable Z which is uniformly distributed from a to b is:

$$\langle Z \rangle = \int_a^b z f_Z(z)dz = \int_a^b \frac{z}{b-a} dz = \frac{1}{2} \frac{(b^2 - a^2)}{b-a} = \frac{1}{2}(b+a) \,. \tag{4.107}$$

4.8.3 Exponential distribution

The distribution

$$f_Z(z) = \frac{1}{c} e^{-z/c} , \tag{4.108}$$

for positive z is called the **exponential distribution**. It is commonly used to describe waiting time statistics.

We check that the distribution is normalized:

$$\int_0^\infty \frac{1}{c} e^{-z/c} dz = \int_0^\infty e^{-u} du = 1 . \tag{4.109}$$

The average value of Z is

$$\langle Z \rangle = \int_0^\infty \frac{1}{c} z e^{-z/c} dz = c \int_0^\infty e^{-u} du = c . \tag{4.110}$$

We can understand the origin of the exponential distribution as a waiting time distribution from a simple example: A process occurs at a constant, but very low rate r, so that the probability for an event to occur in a time interval from t to $t + dt$ is a constant r multiplied with dt, rdt. What is the probability for the event to occur for the first time in the time interval from t to $t + dt$? For this to happen, the even must not occur for the time t and then occur in the time interval from t to $t + dt$. We can divide the time t into small units Δt. The probability for an event to occur in such a small time interval is $r\Delta t$, and there are $n = t/\Delta t$ such interval from 0 to t. The probability for no event to occur from 0 to t is then

$$P = (1 - r\Delta t)^n = \left(1 - \frac{r \cdot t}{n}\right)^n \to e^{-rt} , \tag{4.111}$$

when $\Delta t \to 0$ and $n = (t/\Delta t) \to \infty$. The probability for the event to occur for the first time in a time interval from t to $t + dt$ is P, which is the probability for no event to occur before t, multiplied by rdt, which is the probability that the event occurs in the interval from t to $t + dt$. This gives the probability for the event to occur for the first time in the inverval from t to $t + dt$ to be:

$$f_T(t)dt = e^{-rt} rdt . \tag{4.112}$$

We can therefore interpret $f_T(t)$ as the probability density for the waiting time T. We can therefore interpret the exponential distribution as a waiting time distribution with a rate r or a characteristic time $\tau = 1/r$.

We can test this theory by generating a simple data-set for waiting time statistics. Let us assume that a light bulb has a probability $p = 1/1000$ to fail during an hour. What is the distribution of the lifetime of such light bulbs?

Let us generate a numerical experiment to measure this, and then compare the results of the numerical simulation with the theory. First, how can we find the number of trials needed to observe the first event, when the probability of an even in each

trial is p? We could do this using a loop, which continues until we have achieved success, and then count how many attempts were needed:

```
n = 0
while (rand(1,1)>p);
    n=n+1
```

This is not a very efficient algorithm in a vectorized language such as Python, but we have chosen to implement it like this because the structure of the algorithm is very clear. We then need to repeat this experiment `nsamp` times and collect the results in a variable `tval`. This is implemented in the following program:

```
#start1
from pylab import *
p = 1.0/1000.0
nsamp = 10000
tval = zeros(nsamp,float)
for i in range(nsamp):
    j = 0
    while (rand(1)>p):
        j = j + 1
    tval[i] = j
```

How to estimate the probability density for T, the waiting time? We do this following the frequency definition of probability: By counting how many events, N_t, are in an interval from t to $t+dt$. The probability for T to be in the interval from t to $t+dt$ is then $f_T(t)dt = N_t/N$, where N is the total number of experiments. The number N_t are produced in a histogram, and we can then find the probability density through

$$f_T(t) = \frac{N_t}{N\,dt}\,. \tag{4.113}$$

Notice that the interval size, dt, appears in the denominator! This factor is often forgotten by students, but it is essential in order to find the probability *density*.

First, let us find the histogram using bins that are 1 second in width. We do this by choosing the number of bins used to correspond to the largest value of T observed using

```
n,t=hist(tval,bins=max(tval))
```

where the variable `n` now contains the counts, N_t, and t contains points representing the bins. Since the bins all have the same size, this is a feature of the `hist` function, we can find the bin size dt from the first two values of `t`:

```
dt = t[1]-t[0]
```

We can finally estimate the probability density using (4.113):

```
Pt = n/(nsamp*dt)
```

Notice that this is a vectorized command, which calculates the values for `Pt` for all the values in `n`. Essentially, a command like this corresponds to

```
for i in range(len(n)):
    Pt[i] = n[i]/(nsamp*dt)
```

We can then plot the calculated probability density and compare with the theoretical values found above:

$$f_T(t) = pe^{-pt} \,, \tag{4.114}$$

which is done by

```
theory = p*exp(-p*t)
```

which again is a vectorized command that calculates the values for `theory` for all values in the `t`-array. The results are plotted by

```
plot(t,Pt,'o',t,theory,'-'), show()
```

and the resulting distribution is shown to the left in Fig. 4.8.

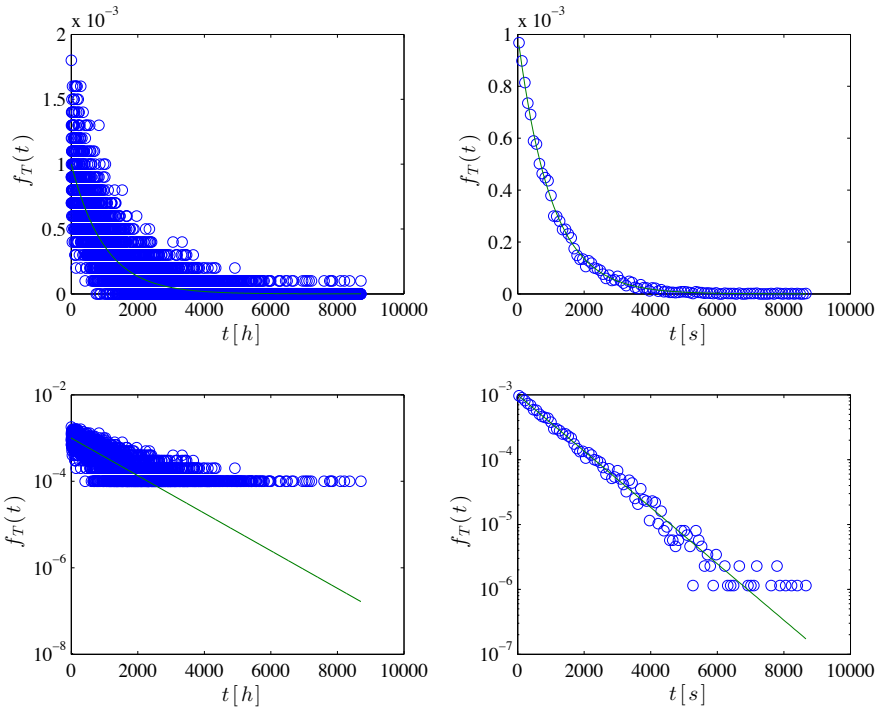

Fig. 4.8 Plot of the measured probability denstiy $f_T(t)$ for the waiting time distribution for a process occuring with a probabity $p = 1/1000$ per hour. (Left) Bin size correspoding to the smallest possible resolution in t. (Right) With 100 bins.

Unfortunately, this plot is really messy! It is not easy to observe the behavior because there are too many data points and too much noise. What can we do to improve this? We can broaden the bins, so that there are more points in each bin –

we need to change the bin size dt. For example, we can analyze the data using only 100 bins. However, we then need to be very careful to remember to include the bin size dt when we estimate the probability density. This was done using

```
n,t = hist(tval,bins=100)
dt = t[1]-t[0]
Pt = n/(nsamp*dt)
theory = p*exp(-t*p)
plot(t,Pt,'o',t,theory,'-')
```

And the resulting plots are shown to the right in Fig. 4.8. The results are now much clearer. And we can now also see that the theory fits perfectly. The data is indeed well described by the exponential distribution.

4.8.4 Poisson distribution

The Poisson distribution appears when we look at a process that occurs at a constant rate, so that over a time interval T, the expected number of events is λ. In this case, we can divide the time interval into N small intervals Δt, so that in each time interval the probability for the event to occur is $p = \lambda/N$. As we divide the time interval into more and more subintervals, the probability for an event to occur in a subinterval, p, becomes smaller and smaller, and we can therefore assume that the probability for more than one event to occur in a subinterval is negligible. What is then the distribution of n, the number of event occuring during the time interval T?

The number of events, n, occuring in N subintervals is the sum of the number of events, n_i, occuring in each subinterval i:

$$n = \sum_{i=1}^{N} n_i \,, \tag{4.115}$$

where the probability for n_i to be 1 is $p = \lambda/N$ and the probability for n_i to be 0 is $1-p$. This means that n is described by a binomial distribution:

$$P(n) = \frac{N!}{n!(N-n)!)} p^n (1-p)^{N-n} \,, \tag{4.116}$$

where we can now insert $p = \lambda/N$:

$$P(n) = \frac{N!}{n!(N-n)!} \left(\frac{\lambda}{N}\right)^n \left(1-\frac{\lambda}{N}\right)^{N-n} \tag{4.117}$$

$$= \frac{N!}{n!(N-n)!N^n} \lambda^n \left(1-\frac{\lambda}{N}\right)^N \left(1-\frac{\lambda}{N}\right)^n \tag{4.118}$$

$$= \frac{1}{n!} \frac{N!}{(N-n)!N^n} \lambda^n \left(1-\frac{\lambda}{N}\right)^N \left(1-\frac{\lambda}{N}\right)^n \,, \tag{4.119}$$

where we notice that when $N \to \infty$:

$$\frac{N!}{(N-n)!N^x} \to 1 \;\Rightarrow\; \left(1-\frac{\lambda}{N}\right)^N \to e^{-\lambda} \;\Rightarrow\; \left(1-\frac{\lambda}{N}\right)^n \to 1\,. \tag{4.120}$$

This gives

$$P(n) = \frac{\lambda^n e^{-\lambda}}{n!}\,, \tag{4.121}$$

for the probability for the number n of event in an interval with an average number of events λ, where the events are rare and independent.

The Poisson distribution describes processes where there are a large number of possible events (trials), but where the probability of each event is low. The distribution describes how many events occur in a given time interval. This can for example be used to describe the number of photons detected by a telescope or a camera in astronomy, the number of cars arriving at a traffic light, the decay of radiactive nuclei, or the number of mutations on a strand of DNA.

Summary

Statistical experiment. A **statistical experiment** is a trial with a given set of possible outcomes.

Outcome. A result n_i of a staticial experiment is called an **outcome**.

Probability. The frequency based definition of the probability p_i for the event that $n = x$ is that $P(n = x) = N_x/M$, where N_x is the number of times that $n = x$ was observed in M events, in the limit when M becomes large.

Random variable. A random variable Z describes the result of an experiment. The outcome of a particular experiment is z.

Probability density. The probability for a set of discrete outcomes is $P_Z(z)$. The probability for an outcome z in the range from z to $z+dz$ from a continous set of outcomes is $f_Z(z)dz$, where $f_Z(z)$ is called the probability density.

Average. The **average** of Z is $\langle Z\rangle = \sum_z z P_Z(z)$ for a discrete set of outcomes and $\langle Z\rangle = \int z f_Z(z)dz$ for a continuous set of outcomes.

Standard deviation. The variation in Z is characterized by the variance σ^2 (or the standard deviation σ), which is $\sigma^2 = (1/M)\sum(z_i - \langle z\rangle)^2$.

Exercises

Exercise 4.1. Deck of cards

We will address various probabilities for drawing cards from a deck of cards.

a) What is the probability to draw an ace of spades?

b) What is the probability to draw a two of diamonds?

c) What is the probability to draw a black card?

d) What is the probability to draw a spade?

e) What is the probability to draw a card that is not a spade?

f) What is the probability to draw two spades from a full deck?

g) What is the probability to draw five cards from a full deck and get four cards of equal value?

Exercise 4.2. Dice

We are using normal, fair, six-sided dice:

a) What is the probability to get a 1?

b) What is the probability to not get a 1?

c) If we throw two dice, what is the probability that both dice show 6?

d) If we throw two dice, what is the probability that the sum is 3?

e) If we throw two dice, what is the probability that the sum is j for all j?

f) If we throw two dice two times, what is the probability to get two six'es at least once?

g) If we throw four dice, what is the probability to get at least two six'es?

h) If we throw two dice, what is the probability that both dice show the same?

i) If we throw two dice, what is the probability that the sum is odd? And even?

j) If we throw one dice six times, what is the probability that we get at least one six?

Exercise 4.3. Casino in St. Petersburg

In the Casino in old St. Petersburg they had a special game. You paid a fee to participate, and then you started to flip a coin, and you continued flipping until you got heads. If heads came up for the first time after N flips, you won 2^N rubles. You can assume that the probability to flip a head is p.

a) Find the probability $P(N)$ for the coin to show heads for the first time at flip N.

b) Show that the probability $P(N)$ is normalized.

c) Find the average number of flips needed to get the first heads. where we have introduced $q = (1-p)$. We use a common trick to find such sums, by realizing that the sum can be written as

$$\sum_{n=0}^{\infty} nq^n = \sum_{n=0}^{\infty} q\frac{d}{dq}q^n = q\frac{d}{dq}\sum_{n=0}^{\infty} q^n = q\frac{d}{dq}\frac{1}{1-q} = \frac{q}{(1-q)^2} = \frac{q}{p^2}\,, \qquad (4.122)$$

which we insert back to get

$$\langle N\rangle = 1 + p\frac{q}{p^2} = 1 + \frac{1-p}{p}\,. \qquad (4.123)$$

d) Find the average return for a player in this game. Show that the return approaches infinity unless p is larger than $1/2$.

e) If $p = 0.6$, what must the entry fee for the game be to ensure that the Casino does not loose money?

Exercise 4.4. Waiting time statistics

Many problems in physics are about waiting — waiting for an event to occur. For example, we often questions of the type — what is the probability for an event X to happen for the first time after N attempts. Concepts such as mean-free-path and mean-free-time are often associated with such processes.

As an example of a waiting time distribution problem, let us address electrons with a narrow distribution of kinetic energies around E ($E \gg kT$). These electrons are injected into a material, where they have a certain probability to the scattered. The probabilitiy for scattering depends on the number of scattering centers the electron passes. This gives a probability $p\Delta x$ to be scattered after a short distance Δx.

a) When the electron starts from $x = 0$, show that the probability, $P(x)$, for the electron to reach x without being scattered is e^{-px}.

b) What is the probability for the electron to be scattered in the interval from x to $x + dx$?

c) How far does the electron travel on average before it is scattered?

d) What is the probability for the electron to travel for a time t without getting scattered?

e) What is the average time the electron travels before being scattered? This is called the average-free-time between collisions.

Exercise 4.5. Poisson statistics

We assume that you are studying a rare event such as the chance of getting the number 1000 if you spin a wheel of fortune with 1000 numbers from 1 to 1000. To make our results more general we call the probability to get 1000 p.

a) If you spin N times, what is the probability to get 1000 exactly one time.

b) What is the general probability $P(N,n)$ to get 1000 n times in N spins. What is the average value of n, $\bar{n}$?

c) Show that the expression can be approximated as

$$P(N,n) = \frac{\bar{n}^n e^{-\bar{n}}}{n!} \,, \tag{4.124}$$

when N approaches infinity and $p \ll 1$.

Exercise 4.6. Gamma distribution

Let us address an unfair coin with a probability p for heads. We flip the coin until we have obtained n heads, which concludes the experiment. The result of the experiment is N, the number of flips.

a) Find the probability $P(N,n)$ that we need to flip N times to get n heads.

b) Show that the probability is normalized.

c) Find an expression for the average N and the standard deviation of N.

Exercise 4.7. Stirling's formula

We often use Stirling's formula as an approximation for the factorial. However, the formula is only supposed to be a good approximation for $N \gg 1$. Let us see how good the approximation is for smaller N.

Stirling's approximation gives

$$N! = N^N \sqrt{2\pi N} \exp\left(-N + \frac{1}{12N} + \mathcal{O}\left(\frac{1}{N^2}\right)\right) \,, \tag{4.125}$$

or

$$\ln N! = \frac{1}{2}\ln(2\pi N) + N(\ln N - 1) + \frac{1}{12N} + \mathcal{O}\left(\frac{1}{N^2}\right) \,, \tag{4.126}$$

However, we often use simpler versions of the formula, such as

$$S_1(N) = N^N e^{-N} \,,\; S_2(N) = \sqrt{2\pi N} S_1(N) \,,\; S_3(N) = e^{\frac{1}{12N}} S_2(N) \,. \tag{4.127}$$

a) Write a program to compare the values for $N = 1, 2, 4, 8, 16, 32, 64$. Plot the three resulting functions and the exact values.

b) Find the relative error from using $N! \simeq S_3(N)$ for $N = 10^{20}$.

c) How large should N be for $\ln S_1(N)$ to represent $\ln N!$ with a relative error of less than 10^{-6}?

Exercise 4.8. Conducting strip

An experiment may result in two possible outcomes, heads with probability p and tails with probability $q = 1 - p$.

a) What is the probability that N independent flips all results in heads?

b) What is the probability for n heads and $N - n$ tails in N flips?

c) From the result above, find the average number of heads, $\bar{n}$ and the variance

$$(n - \bar{n})^2 \,. \tag{4.128}$$

Now, let us consider a strip consisting of N squares as illustrated in fig. 4.9. The squares are electrically conductive with probability p. Two conducting squares next to each other provide a conductive path.

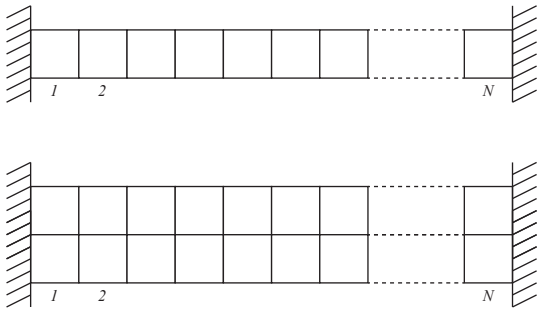

Fig. 4.9 Illustration of strip of conducting patches.

d) What is the probability for the strip to be conducting? What is the probability for one isolating square in the strip?

Now, we study two strips next to each other, that is, $2N$ squares. We assume that a conducting path can be formed both by two squares sharing a side and when touching at a corner.

e) What is the probability, P_c, for the double-strip to be conductive?

Let us assume that we have $2N$ squares, of which $N+M$ are conductive and differ from the others by a metallic surface. The $2N$ squares form a double-strip of length N.

f) What is the total number of different configurations Ω_{tot}?

g) What is the total number of conducting configurations, Ω_{cond}?

h) If we define

$$P_{\text{cond}} = \frac{\Omega_{\text{cond}}}{\Omega_{\text{tot}}} \; ; \tag{4.129}$$

and introduce $p = (N+M)/(2N)$, will we get the same situation as for P_c above or is the situation different? If this is the case, explain the difference.

Exercise 4.9. Velocity distribution in Argon gas

Let us use the molecular simulation approach to measure and characterize the velocity distribution in a gas.

a) The file in.gasexer010[13] can be used to simulate a two-dimensional gas. The resulting output file is found in gasexer010.lammpstrj[14]. Measure and plot the distribution of the x-component of the velocities of the atoms in the final time-step.

[13] `http://folk.uio.no/malthe/fys2160/in-gasexer010`

[14] `http://folk.uio.no/malthe/fys2160/gasexer010.lammpstrj`

b) What is the average velocity $\bar{v}_x$ and the standard deviation of the velocities, σ_v? Comment on the results.

c) Suggest a function $P(v_x)$ that can fit the data and check how well it fits the measured data.

d) Compare the distribution of v_x and the distribution of $v = \sqrt{v_x^2 + v_y^2}$. Comment on the results. (Difficult) Can you predict the functional form of $P(v)$?

e) Run a sequence of simulations for $N = 200, 400, 800, 1600$ atoms and find the standard deviation, σ_v, as a function of N. Find a reasonable functional form for $\sigma_v(N)$. What happens when N becomes realistically large, such as $N = 10^{23}$?

f) Run a sequence of simulations for various values of the initial velocity (energy) used in the simulations $v_0 = 1.5, 2.0, 2.5, 3.0$, and 3.5. Find the standard deviation σ_v as a function of K/N, the average kinetic energy per particle. Comment on the result. What does this tell you about the temperature-dependence of the velocity distribution?