Advanced Programming in Engineering
lecture notes

P.J. Visser, S. Luding, S. Srivastava

March 13, 2011

Changes

13/3/2011

- Added Section 7.4 to 7.7.
- Added Section 4.4

7/3/2011

- Added chapter Chapter 7

23/2/2011

- Corrected the equation for the average temperature in terms of microscopic quantities, Eq. 3.14.
- Corrected the equation for the speed distribution in 2D, Eq. 3.19.
- Added links to sources of 'true' random numbers available through the web as footnotes in Section 5.1.
- Added examples of choices for parameters of the LCG and LFG in Section 5.1.1 and 5.1.2, respectively.
- Added the chi-square test and correlation functions as ways for testing random numbers in Section 5.1.4.
- Improved the derivation of the average squared distance of particles in a random walk being linear with time, in Section 5.2.1.
- Added a discussion of the meaning of the terms in the master equation, in Section 5.2.5.
- Small changes.

19/2/2011

- Corrected the Maxwell-Boltzmann distribution for the speed, Eq. 3.19.

14/2/2011

- Added Chapter 6.

6/2/2011

- Verified Eq. 3.15 and added footnote discussing its meaning in case of periodic boundary conditions.
- Added formula for the radial distribution function in Section 3.4.2, Eq. 3.21.

31/1/2011

- Added Chapter 5.

9/1/2010

- Corrected labels on the axes in Fig. 3.10, changed $U \cdot \varepsilon$ and $r \cdot \sigma$ to U/ε and r/σ, respectively.
- Added truncated version of the Lennard-Jones potential in Section 3.3.1.
- Corrected speed distribution, Eq. 3.19 to be valid in 2D.
- Small changes.

3/12/2010

- Added section on simulating fluids, Section 3.3.
- Added section on statistical mechanics, Section 3.4.

13/12/2010 (2)

- Corrected that spring equilibrium lengths for N_p particles in 2D are fully determined by a particle pair rather than a particle. I.e. the equiblibrium lengths of Section 3.2.2 are $r_{\mathrm{eq},ij},\ i = 1..N_\mathrm{p}, j = 1..N_\mathrm{p}$, rather than $r_{\mathrm{eq},i},\ i = 1..N_\mathrm{p}$.
- Corrected that the outer loop of the force calculation, Algorithm 4, can run from 1 to $N_\mathrm{p} - 1$ rather than from 1 to N_p.
- Added section on the potential and kinetic energy in 2D, Section 3.2.3.
- Added exercise asking to add the computation of the potential and kinetic energy to the algorithm.
- Small changes.

13/12/2010

- Changed algorithms. Loop variables are now updated at the start of an integration step rather than the end. Positions at the current time step is stored rather than the position at the next time step. Loop runs from $n = 0$ to $n = N_\mathrm{t}$ rather than $n = 1$ to $n = N_\mathrm{t}$. It makes more sense this way.

- Corrected $\vec{r}_i = \begin{bmatrix} x_i & y_i \end{bmatrix}, \ i = 1..N_\mathrm{p}$ to $\vec{r}_i = \begin{bmatrix} x_i & y_i \end{bmatrix}^\mathrm{T}, \ i = 1..N_\mathrm{p}$ in Section 3.2.3.

- Corrected force calculation for particles connected with springs in 2D, Section 3.2; introduced spring compression δ and corrected that $\vec{n}_{ij}$ is the vector to particle i from particle j, not from particle i to particle j.

- Expanded section on solids, Section 3.2.4 a bit.

- Other small changes.

Preface

The present script attempts to describe into sufficient detail the material treated during the lectures. It is based mainly on handwritten notes made by the author during the 09/10 lectures. It therefore attempts to largely reflect the material as lectured by professor Luding (the topics of differential equations, numerical integration of ordinary differential equations, molecular dynamics, random numbers and fractals), Katia Bertoldi (the topic of finite element method) and Anthony Thornton (the topics of finite volume method and SPH method).

It is a work in progress, and as of yet only material covering the first few lectures has been written. The material that has been written will become available in parallel to the lectures covering it. The latest version will be available through Blackboard.

For certain topics, hyperlinks that point to further reading and other helpful places have been added to this document. Of course these will only be available when reading the digital version in a PDF reader supporting hyperlinks, such as Adobe Reader [1].

Throughout this document exercises are described. They are intended for self study purposes and are not mandatory.

Many thanks to all who have helped out in making this document, in particular to professor Luding, Saurabh Srivastava, Thomas Weinhart and Martin Robinson for their feedback on drafts of the document.

I hope you find this document a helpful addition to the lectures. I am sure that you will encounter many mistakes, be it in spelling or content. Any corrections and other comments would be greatly appreciated. Please send them to pieterjanv@gmail.com.

[1]available on http://get.adobe.com/reader/

Contents

Chapter 1

Numeral systems

Numeral systems are writing systems for expressing numbers. The most commonly used numeral system is the positional system. Here a numeral is written as a series of digits a_k in a base B, where $a_k = 0, 1, ..., B - 1$. The number N represented by the numeral is given by

$$N = \sum_k a_k B^k \tag{1.1}$$

The digits used to denote numbers are those of the arabic numerals for bases smaller or equal to ten. Thus the symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. For larger bases letters are used in addition to arabic numerals. For example, the digits used in hexedecimal notation ($B = 16$) are the arabic digits together with the symbols A, B, C, D, E and F.

In case when it is unclear from context in what base a numeral is written a subscript with the base number is usually added. For example, the numeral 3 in decimal base ($B = 10$) is written as 3_{10}.

1.1 Representation of positive integers and real numbers

To represent an integer $N_{\mathbb{Z}}$ using the positional system the index k ranges from zero to n, such that $B^n \leq N_{\mathbb{Z}} < B^{n+1}$, which leads to a sum with $n+1$ terms:

$$N_{\mathbb{Z}} = \sum_{k=0}^{n} a_k B^k. \tag{1.2}$$

As an example, the number represented by the numeral 11 in decimal ($B = 10_{10}$), binary ($B = 2_{10}$) and hexadecimal ($B = 16_{10}$) base are given. From the numeral we have $a_0 = 1,\ a_1 = 1$, and substituting this in Eq. (1.2) the number 11_B is thus $1 \cdot B^0 + 1 \cdot B^1$.

In decimal base the number is $1_{10} + 10_{10} = 11_{10}$.

In binary base the number is $1_{10} + 2_{10} = 3_{10}$.

In hexadecimal base the number is $1_{10} + 16_{10} = 17_{10}$.

To represent a real number requires representing the part smaller than unity. This is provided for by letting k take on values smaller than zero. For irrational numbers the representation of the part smaller than unity requires an infinite number of digits. A real number $N_{\mathbb{R}}$ can therefore be written as

$$N_{\mathbb{R}} = \sum_{k=-\infty}^{n} a_k B^k, \tag{1.3}$$

where n is such that $B^n \leq N_{\mathbb{R}} < B^{n+1}$.

As an example, the number represented by the numeral 3.14 in decimal ($B = 10_{10}$), binary ($B = 2_{10}$) and hexadecimal ($B = 16_{10}$) base are given. From the numeral we have $a_{-2} = 4$, $a_{-1} = 1$, $a_0 = 3$, and substituting this in Eq. (1.2) the number 3.14_B is thus $4_{10} \cdot B^{-2} + 1_{10} \cdot B^{-1} + 3_{10} \cdot B^0$.

In decimal base the number is $4_{10} \cdot 10_{10}^{-2} + 1_{10} \cdot 10_{10}^{-1} + 3_{10} \cdot 10_{10}^{0} = 3.14_{10}$.

In binary the number is undefined, since the numeral contains digits that aren't present in the binary numeral system.

In hexadecimal base the number is $4_{10} \cdot 16_{10}^{-2} + 1_{10} \cdot 16_{10}^{-1} + 3_{10} \cdot 16_{10}^{0} = 3.078125_{10}$.

1.2 Representation of negative numbers

To represent negative numbers in binary using digits only, the most commonly used method is the so-called two's complement. The method works by making the leftmost digit represent a negative number instead of a positive one. Suppose we want to denote a number N using m fractional digits and $n+1$ integer digits of base B. The number is then given by

$$N = a_n \cdot -B^n + \sum_{-m}^{n-1} a_k B^k. \tag{1.4}$$

So if the digit a_n is nonzero the number is negative, otherwise it is positive.

Example: (un)signed binary numerals

To clarify with an example, Tab. 1.1 gives all unsigned (only positive) and signed (both positive and negative) integers representable by three binary digits ($B = 2$). The second column is obtained from substiuting the numeral in Eq. (1.2), the third column is obtained from Eq. (1.4).

An unsigned integer binary numeral of $n+1$ bits can represent values on the interval $\left[0, 2^{n+1} - 1\right]$ and a signed one on the interval $[-2^n, 2^n - 1]$.

Ex. 1 — What are the smallest and largest values representable by the MATLAB datatypes `uint32` and `int32`?

Numeral	Unsigned value	Signed value
000	$0 \cdot \mathbf{4} + 0 \cdot 2 + 0 \cdot 1 = 0$	$0 \cdot -\mathbf{4} + 0 \cdot 2 + 0 \cdot 1 = 0$
001	1	1
010	2	2
011	3	3
100	$1 \cdot \mathbf{4} + 0 \cdot 2 + 0 \cdot 1 = 4$	$1 \cdot -\mathbf{4} + 0 \cdot 2 + 0 \cdot 1 = -4$
101	5	-3
110	6	-2
111	7	-1

Table 1.1: Signed and unsigned integers using three bits ($n = 2$, $B = 2$), thevalue of the leftmost digit is marked in boldface

Chapter 2

Ordinary differential equations

In molecular dynamics the motion of particles in time is computed. Since the motion of particles in time is described by ordinary differential equations (ODE's), the content of this chapter is preliminary for the next chapter on molecular dynamics.

In general ODE's do not allow an exact solution. Simulating this motion in time therefore requires integrating ODE's numerically. This chapter discusses some ordinary differential equations and methods used to numerically integrate them.

In learning to solve ODE's with numerical integration it is helpful to start with an ODE that allows an analytic solution, so that results obtained with numerical integration can be compared to the analytic results. To this end the harmonic oscillator and its analytic solution are discussed in Section 2.1.

Secondly, Section 2.2 discusses nondimensionalization, as it is a process which can get more out of a computer simulation.

Finally, Section 2.3 discusses several methods of numerically integrating ODE's. It covers to some detail the Euler, Euler-Cromer, Verlet and Runge-Kutta integration schemes.

2.1 The harmonic oscillator

In introducing the numerical methods that solve odinary differential equations in this course, the methods are applied to solving the differential equation of the harmonic oscillator. The reason is that we can obtain an analytic solution to this equation, allowing us to compare and judge the quality of the numerical solutions.

In this section the analytic solution of the equation of motion of the harmonic oscillator is derived.

The differential equation of the harmonic oscillator represented in Fig. 2.1 is given by

$$m\ddot{x}(t) = -k(x_\mathrm{m} - x_\mathrm{e}),$$

where m is the mass the spring acts on, k is the stiffness of the spring, x_m is the position of the mass and x_e is the equilibrium position of the mass.

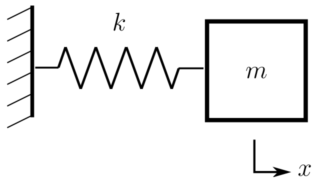

Figure 2.1: Schematic representation of the harmonic oscillator

Letting $x = x_\mathrm{m} - x_\mathrm{e}$ we can rewrite this to

$$\ddot{x}(t) = -\frac{k}{m}x, \tag{2.1}$$

which can be interpreted as a spring with an equilibrium length of zero.

2.1.1 Analytic solution

The solution to Eq. (2.1) is given by

$$x(t) = A\sin(\omega t + \delta), \tag{2.2}$$

where ω is the angular frequency of oscillation given by $\omega = 2\pi/T = \sqrt{k/m}$, where T is the period of oscillation.

In order to validate the numerical solutions the velocity is also required. The analytic solution for the velocity is given by the derivative of Eq. (2.2), equal to

$$\dot{x}(t) = \omega A\cos(\omega t + \delta). \tag{2.3}$$

The amplitude A and phase δ are determined by the intial conditions $x(0) = x_0$ and $v(0) = \dot{x}(0) = v_0$. Substituting these in Eqs. (2.2) and (2.3), respectively, and rewriting gives

$$\sin\delta = \frac{x_0}{A}, \tag{2.4}$$

$$\cos\delta = \frac{v_0}{A\omega}. \tag{2.5}$$

We can solve for the phase δ by dividing Eq. (2.4) by Eq. (2.5):

$$\tan\delta = \omega\frac{x_0}{v_0} \quad\Rightarrow\quad \delta = \arctan\left(\omega\frac{x_0}{v_0}\right). \tag{2.6}$$

We can solve for the amplitude A by squaring and adding Eqs. (2.4) and (2.5):

$$\sin^2\delta + \cos^2\delta = 1 = \frac{1}{A^2}(x_0^2 + (v_0/\omega)^2)$$
$$\Rightarrow \quad A = \sqrt{x_0^2 + (v_0/\omega)^2} \tag{2.7}$$

Ex. 2 — Write a small program that generates the analytical solution as time series and plot x vs. t.

2.2 Nondimensionalization

Using dimensions in our computations makes them easy to understand. Assigning dimensional units such as kilometers, kilograms and meters to the numbers we compute with provides an easy check for consistency. But it is not so that dimensions are necessary, and this is clear when one considers the computations performed by a computer. A computer just uses what we regard as pure numbers to perform its computations.

We can also perform our computations without the use of dimensions. The process of rewriting equations to a form that makes no use of dimensions is called 'nondimensionalization'. There are advantages to doing so.

One advantage of nondimensionalization is that the solution to the resulting nondimensional equation can be converted to the solution to a range of real world problems. I.e. one simulation provides the results that can be converted to multiple real world problems. Another advantage has to do with the limitations on the precision of numbers the computer can represent. When the numbers that appear in a problem range between orders of magnitude too far apart for a computer to represent simultaneously with the required accuracy, nondimensionalization might allow one to represent the problem on a scale that a computer can represent properly. Another advantage is that in some cases nondimensionalization reduces the number of parameters in the equation.

Any dimensional quantity Q can be thought of as being a product of a dimensionless quantity (a numerical value) $\{Q\}$ and a dimensional unit $[Q]$. The nondimensional number $\{Q\}$ is said to be measured in units of $[Q]$. To nondimensionalize an equation, the dimensional quantities in the equation can therefore be multiplied with an inverse dimensional unit $1/U$, such that $[U] = [Q]$ and therefore $Q/U = \{Q\}/\{U\}$, a dimensionless number. An interesting choice of U would be a unit that only depends on characteristics of the system under consideration. For example, given a characteristic length σ, we define a nondimensional variable x' to nondimensionalize the variable length x according to $x' := x/\sigma$.

In the following, the differential equation of the harmonic oscillator, Eq. (2.1), is nondimensionalized.

First we have to find enough characteristic quantities. For each unit in the equation we define a unit that is just a multiple of the SI unit, but depends on the characteristics of a harmonic oscillator. So we could define a unit of length σ representing the equilibrium length, a unit of mass m_0 representing the oscillating mass, and a unit of time τ representing the period of oscillation.

In order to nondimensionalize Eq. (2.1) we replace the dimensional quantities x, t, k and m with nondimensional quantities x', t', k' and m' by multiplying the dimensional quantities with an appropriate combination of the inverse of units σ, m_0 and τ. This can be achieved by choosing $x' = x/\sigma$, $t' = t/\tau$, $k' = k\tau^2/m_0$ and $m' = m/m_0$. Note that for this equation we have now defined enough units to nondimensionalize any dimensional quantity. For example, we could define a unit of density as $\rho_0 = m_0/\sigma^3$.

Instead of solving Eq. (2.1) we can now solve the nondimensional differential equation

$$\frac{d^2x'}{dt'^2} = -\frac{k'}{m'}x'. \tag{2.8}$$

To obtain a solution we set k' and m' to whatever we find appropriate. We also provide dimensionless initial conditions $x'_0 = (1/\sigma)x_0$ and $v'_0 = (\tau/\sigma)v_0$, which are determined by the particular problem to be solved. This results in the solution x' as a function of t'.

We can now obtain the solution to a real world dimensional problem by computing $x = x'\sigma$ and $t = t'\tau$. Here σ and τ can be varied to describe different real world problems.

The choice of reference units is somewhat arbitrary. One can use any combination of units that together provide all the physical dimensions that occur in the equation. Another set of reference units could be the position σ_0, the mass density ρ_0 and the system energy ϵ_0.

The number of quantities to keep in mind can grow quite large. One may find it convenient to give an overview of all the quantities in a table. An example of such an overview is given in Tab. 2.1. It lists all dimensional quantities we have in mind and their relation to the nondimensional quantities in the simulation.

Ex. 3 — Nondimensionalize Eq. (2.1) using the reference units σ_0, ρ_0 and ϵ_0 and fill in the table using these units.

2.3 Numerical integration of ODE's

Many ordinary differential equations do not allow for an analytic solution. To obtain a solution therefore requires an approximate approach, as provided by numerical integration.

The solution to Eq. (2.1) can be approached by integrating it numerically. To approach the exact solution, we estimate the value of x at discrete times

	Dimensional quantity	Quantity in simulation
reference units	position σ	$\sigma' = 1$
	time τ	$\tau' = 1$
	mass m_0	$m_0' = 1$
measurable quantities	position x	$x' = x/\sigma$
	time t	$t' = t/\tau$
	mass m	$m' = m/m_0$
	stiffness k	$k' = k\tau^2/m_0$
	angular frequency ω	$\omega' = \omega\tau$
	period T	$T' = T/\tau$
	density ρ	$\rho' = \rho\sigma^3/m_0$
	energy ϵ	$\epsilon' = \epsilon\tau^2/(m_0\sigma^2)$

Table 2.1: An overview of a nondimensionalization of the harmonic oscillator

$t_i = i\Delta t$, where Δt is the size of some arbitrary time interval. Four numerical integration schemes are introduced: the Euler, Euler-Cromer, Verlet and Runge-Kutta integration schemes.

2.3.1 Euler integration

For the Euler method the estimation goes as follows. Suppose we know the position and velocity at some time t_i, which we denote by x_i and v_i, i.e. we know $x_i = x(t_i)$ and $v_i = v(t_i)$. We can then estimate the position at the next time step $t_{i+1} = t_i + \Delta t$ by assuming the velocity stays constant during the time interval Δt. We thus estimate $x_{i+1} \approx x_i + \Delta t v_i$. Now, using this to compute the position at the next time step x_{i+2} would require the velocity v_{i+1}. Analogous to computing x_{i+1} this can be estimated by $v_{i+1} = v_i + a_i \Delta t$. Here a_i is the acceleration at time step t_i, which we can compute from Eq. (2.1): $a_i = \ddot{x}_i = -\omega^2 x_i$, where $\omega = \sqrt{k/m}$.

Thus, given an intial position x_0 and intial velocity v_0, we can now estimate the solution $x(t)$ at times t_i, $i = 0, 1, ..., M$, $M = t_{\max}/\Delta t$ by computing

$$x_{i+1} = x_i + \Delta t \; v_i, \tag{2.9}$$

$$v_{i+1} = v_i + \Delta t \; a_i, \tag{2.10}$$

where $a_i = -\omega^2 x_i$.

A single integration step using the Euler method is visualized in Fig. 2.2. Given the exact the solution $x_i = x(t_i)$ the approximation of the solution at the next time step $x_{i+1} = x(t_{i+1})$ is visualized.

There are two great disadvantages of the Euler integration scheme. One is that it 'overshoots' the analytic solution at every step, making the error grow relatively quick with time. The other is that it is only first order

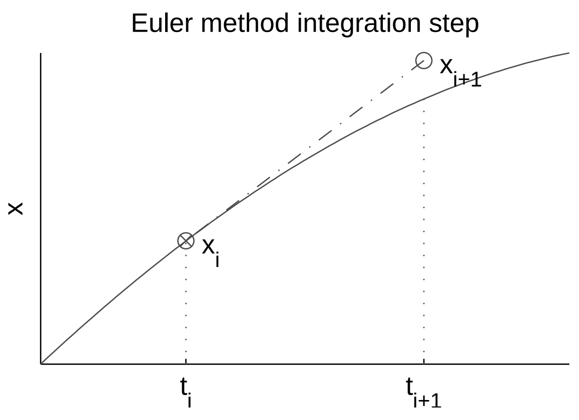

Figure 2.2: Visualization of an Euler method integration step

accurate. This means that decreasing the time step Δt by a factor h makes the error only a factor h smaller. Thus making your time step smaller is not that effective. Higher order schemes exist, for which the error would decrease by a factor h^n, $n > 1$.

2.3.2 Euler-Cromer integration

The Euler scheme has the disadvantage of constantly overshooting the exact solution. The Euler-Cromer scheme avoids this by not estimating the position x_{i+1} using the velocity at the current time v_i but the estimated velocity at the next time v_{i+1}. It is as if it anticipates where to go next, avoiding the constant overshooting. Of course this requires that we estimate v_{i+1} first.

Using the Euler-Cromer scheme, given the intial position and velocity x_0 and v_0, we can estimate the solution $x(t)$ at time $t_i,\ i = 0, 1, ..., M$ by computing

$$v_{i+1} = v_i + \Delta t a_i, \tag{2.11}$$

$$x_{i+1} = x_i + \Delta t v_{i+1}, \tag{2.12}$$

where for the harmonic oscillator $a_i = -\omega^2 x_i$, where $\omega = \sqrt{k/m}$.

A single integration step using the Euler-Cromer method is visualized in Fig. 2.3. Given the exact solution $x_i = x(t_i)$ the approximation of the solution at the next time step $x_{i+1} = x(t_{i+1})$ is visualized.

Even though the Euler-Cromer scheme performs better than the Euler scheme, the disadvantage of the scheme is that it is, just as the Euler scheme, only first order accurate.

2.3.3 Verlet integration

A higher order scheme is the Verlet integration scheme. It is derived by considering the Taylor series expansion of the position $x(t)$. The Taylor

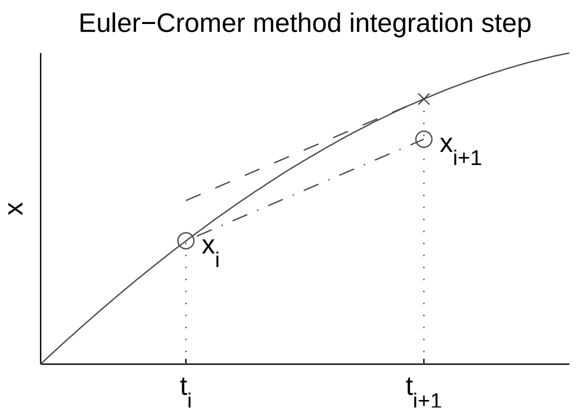

Figure 2.3: Visualization of an Euler-Cromer method integration step

series expansion of x_{i+1} truncated to second order is

$$x_{i+1} \approx x_i + \Delta t \dot{x}_i + \frac{\Delta t^2}{2}\ddot{x}_i. \tag{2.13}$$

Doing the same for x_{i-1} gives

$$x_{i-1} \approx x_i - \Delta t \dot{x}_i + \frac{\Delta t^2}{2}\ddot{x}_i. \tag{2.14}$$

Adding Eqs. (2.13) and (2.14) we get an approximation for x_{i+1} in terms of $\ddot{x}_i$, x_i and x_{i-1}:

$$x_{i+1} \approx 2x_i - x_{i-1} + \Delta t^2 \ddot{x}_i. \tag{2.15}$$

Given initial conditions $x(0) = x_0$ and $\dot{x}(0) = v_0$ we still need x_{-1} to obtain x_1. When this information is not available this can be done by using an Euler approximation; $x_{-1} = x(-\Delta t) \approx x_0 - v_0\Delta t$.

The advantages of the Verlet integration scheme are that it is rather simple and compact yet has an accuracy of order Δt^2. This means that reducing the step size by a factor h makes the error in the numerical solution h^2 times as small. This in contrast to the Euler and Euler-Cromer schemes, which are only first order accurate. Also, it is a symplectic integrator, meaning that the total energy only oscillates around the initial value. This is contrast to for example the Euler method and Runge-Kutta (treated below) methods.

The disadvantage of Verlet integration is that it only applies to differential equations of the form $\ddot{x} = f(x)$.

2.3.4 Runge-Kutta integration

The last integration scheme we discuss is the second order accurate Runge-Kutta scheme, also called the midpoint method. It is thus, as Verlet integration, second order accurate, but in contrast to the Verlet scheme it can be applied to differential equations of the form $\ddot{x} = f(x, \dot{x})$.

The Runge-Kutta scheme works by using the velocity and accelerations at intermediate times $t_{i+1/2}$ in estimating t_{i+1}, so

$$x_{i+1} = x_i + \Delta t \bar{v}_i \tag{2.16}$$

$$v_{i+1} = v_i + \Delta t \bar{a}_i, \tag{2.17}$$

where $\bar{v}_i = v_{i+1/2}$ and $\bar{a}_i = a_{i+1/2}$. To compute $\bar{v}_i$ and $\bar{a}_i$ we use the Euler scheme. For the harmonic oscillator the velocity at the intermediate timestep can be computed by

$$\bar{v}_i = v_i + \frac{\Delta t}{2} a_i,$$

where $a_i = -\omega^2 x_i, \omega = \sqrt{k/m}$ by Eq. (2.1). The intermediate acceleration can be computed by substituting the intermediate position in Eq. (2.1);

$$\bar{a}_i = -\frac{k}{m}\bar{x}_i,$$

where $\bar{x}_i = x_{i+1/2}$, approximated with the Euler method by

$$\bar{x}_i = x_i + \frac{\Delta t}{2} v_i.$$

A single integration step using the midpoint method is visualized in Fig. 2.4. Given the exact solution $x_i = x(t_i)$ the approximation of the solution at the next time step $x_{i+1} = x(t_{i+1})$ is visualized.

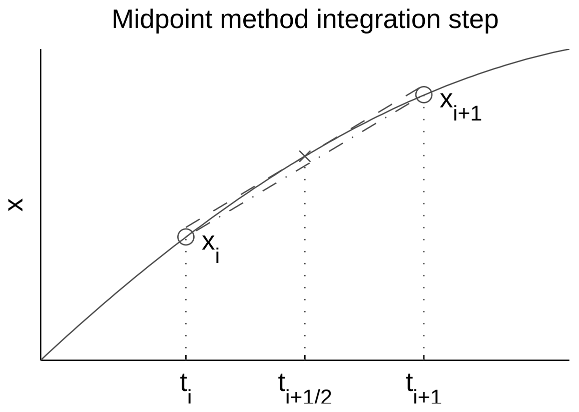

Figure 2.4: Visualization of a Midpoint method integration step

For a differential equation of the general form $\dot{\mathbf{x}} = f(t, \mathbf{x})$ where $\mathbf{x}$ denotes a vector, the Runge-Kutta scheme results in

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta t f\left(t_{i+1/2}, \mathbf{x}_i + \frac{\Delta t}{2} f(t_i, \mathbf{x}_i)\right). \tag{2.18}$$

It is now shown how Eq. (2.1) can be solved using the general form of the midpoint method, Eq. (2.18).

In order to get Eq. (2.1) into the form $\dot{\mathbf{x}} = f(t, \mathbf{x})$ we must introduce the extra equation

$$\dot{x} = v.$$

From Eq. (2.1) we can now write

$$\dot{v} = -\frac{k}{m}x.$$

To rewrite these coupled differential equations in a suitable form, we use the vector $\mathbf{x} = [x, v]^{\mathrm{T}}$. Now, the equation

$$\dot{\mathbf{x}} = f(t, \mathbf{x}),$$

with

$$f(t, \mathbf{x}) = \begin{bmatrix} v \\ -\frac{k}{m}x \end{bmatrix}, \tag{2.19}$$

is equivalent to Eq. (2.1).

Now suppose we have the initial conditions $x(0) = x_0$ and $v(0) = v_0$, or

$$\mathbf{x}_0 = [x_0, v_0]^{\mathrm{T}},$$

stiffness k and mass m, then we can compute $\mathbf{x}_1$ by substituting this into Eq. (2.18) with $i = 0$ and $f(t, \mathbf{x})$ given by Eq. (2.19), and so on to compute $\mathbf{x}_2, \mathbf{x}_3, \ldots$.

2.3.5 MATLAB ode45

In order to numerically solve a differential equation the above schemes can be implemented by hand. MATLAB comes with a family of implemented numerical solvers, however. The MATLAB function ode45 implements a different member of the Runge-Kutta family of ODE solvers, namely the Dormand-Prince method.

In the simplest case the function call is

```
[t, y] = ode45(odefun, tspan, y0);
```

We will now first discuss the input arguments and output arguments.

The argument `odefun` is a function handle to a function m-file implementing the function $f(t, \mathbf{x})$, which is Eq. (2.19) for the harmonic oscillator. Suppose this function is implemented in MATLAB as in Listing 2.1, then `odefun` should be assigned with

```
odefun = @harmonic_oscillator_rhs;
```

The second argument `tspan` is a vector specifying the time interval the simulation should cover. It could be assigned with

```
function dxdt = harmonic_oscillator_rhs(t, x)

k = 1;
m = 1;

dxdt = [x(2);
        -k / m * x(1)];
```

Listing 2.1: content of harmonic_oscillator_rhs.m

```
tspan = [t0, tf];
```

where `t0` and `tf` are the start and end times, respectively.

The third argument `y0` is a vector specifying the initial conditions, which in this case could be defined with

```
y0 = [x0; v0];
```

Here `x0` and `v0` are the initial position and velocity, respectively.

The call to `ode45` returns a column vector `t` with all the time points the solution was computed at and a matrix `y` whose rows correspond to the solution at a time point. The `y` is thus what $\mathbf{x}$ is in Eq. (2.19). We can now plot the position and velocity vs. time with the commands `plot(t, x(:, 1))` and `plot(t, x(:, 2))`, respectively. Figure 2.5 shows the plot resulting from the commands in Listing 2.2.

```
% Simulation parameters
t0 = 0;                             % start time
tf = 4 * pi;                        % end time
x0 = 0;                             % initial position
v0 = 1;                             % initial velocity

% Use ode45 to simulate
odefun = @harmonic_oscillator_rhs;  % function handle to f(t, x)
tspan = [t0, tf];                   % simulation time span
y0 = [x0; v0];                      % initial conditions
[t, x] = ode45(odefun, tspan, y0);  % call to ode45

% Plot result
plot(t, x(:, 1));
xlabel('time t'); ylabel('position x'); axis tight;
```

Listing 2.2: solving harmonic oscillator using `ode45` and plotting the result

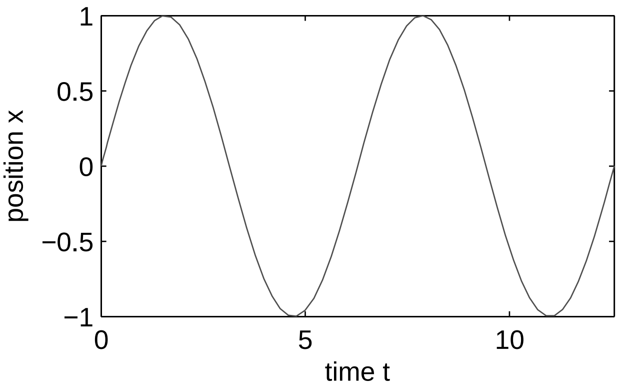

Figure 2.5: Position as a function of time of the harmonic oscillator as simulated with `ode45`

2.3.6 Other integration schemes

A lot more methods have been developed to numerically integrate ordinary differential equations. Important characteristics are of what order such a numerical integration scheme is, how stable it is and which type of equation it can solve.

Higher order schemes converge faster, and thus are cheaper to use when the problem is sufficiently complicated. Using a lower order scheme might take longer because although the computation per timestep is cheaper, the number of timesteps required to obtain the solution can be so much greater than it is for a method of higher order that the total amount of computation required is greater.

Higher order schemes are given by e.g. Runge-Kutta schemes, the most famous being the RK4 method, multistep methods and predictor-corrector methods. Both Runge-Kutta and multi-step methods use more and more information to determine the next time step, leading to a higher order of accuracy. The difference is that Runge-Kutta schemes discard the information used to compute the next time step for each time step, whereas multi-step methods reuse the same information for the computation of several time steps. Predictor-corrector methods are a class of methods in which first a prediction of the solution at the next time step is made, and next this prediction is used to compute a better approximation to the same solution. Many more methods exist and an overview can be found on Wikipedia by searching for 'template:numerical integrators' [1].

Stability is an issue when the size of the time step required by a method is determined by stability reasons instead of accuracy reasons. For many

[1]the 'template:' part is necessary because the page is in the template namespace on Wikipedia, not the default main namespace

algorithms a timestep that should lead to an accurate enough solution is not sufficient because it causes the solution to 'explode'. Equations whose numerical integration by certain methods causes this behaviour are called stiff. A stable algorithm allows the size of the timestep to be determined by accuracy reasons only and therefore leads to a smaller number of timesteps required for obtaining a solution. In general, implicit methods are more stable and computationally less expensive in solving stiff problems. For implicit methods the solution for the next time step can not be expressed explicitly in solutions at previous time steps, requiring extra computation to solve for the next time step.

In general, integration schemes are restricted to a specific class of equations. For example, the midpoint method discussed before can only integrate equations of the form $y'(t) = f(t, y(t)),\ y(t_0) = y_0$. Also, numerically integrating partial differential equations generally requires very different methods from the ones just discussed. Examples of methods to numerically integrate partial differential equations are finite difference methods, the finite element method and the finite volume method. The last two are discussed later in this course.

Chapter 3

Molecular Dynamics

Molecular dynamics (MD) attempts to model reality as a collection of particles obeying Newton's laws. It is therefore assumed that the motion of particles satisfies

$$m_i \ddot{\vec{r}}_i = \vec{f}_i, \tag{3.1}$$

where m_i is the mass of the ith particle with position $\vec{r}_i$ subject to force $\vec{f}_i$.

This chapter describes molecular dynamics for the case of point sized particles in one- and twodimensional space. Section 3.1 discusses particles connected with springs that are free to move in one dimension. Section 3.2 discusses particles connected with springs that are free to move in two dimensions. Section 3.3 discusses particles that interact like the molecules in fluids do. Finally, Section 3.4 discusses statistical mechanics, the theory that connects the microscopic scale with the macroscopic scale.

Given the initial positions, initial velocities and the masses of the particles of such a system we can compute everything there is to know about the system if we specify how to compute the forces on the particles.

3.1 Particles in one dimension connected with springs

We introduce MD with point particles restricted to 1D motion that are subject to the forces produced by linear springs connecting them. Section 3.1.1 discusses the simulation of the motion of two particles connected with a spring. Section 3.1.2 generalizes this to the motion of N_p particles conneceted with springs.

3.1.1 Two particles connected with a spring

Figure 3.1 displays two particles that can move in one dimension and are connected with a spring. The positions of the particles are x_1 and x_2, they both have mass m and the spring has stiffness k and equilibrium length x_e.

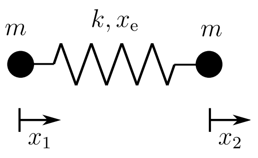

Figure 3.1: Two particles connected by a spring

We can write down Eq. (3.1) for each particle;

$$m\ddot{x}_i = f_i, \quad i = 1..2,$$

where

$$\begin{aligned} f_1 &= k\left[(x_2 - x_1) - x_e\right], \\ f_2 &= -k\left[(x_2 - x_1) - x_e\right]. \end{aligned}$$

Given initial positions $x_i^{(0)}$ and velocities $v_i^{(0)}, i = 1, 2$, we can use an integration scheme such as Verlet to obtain the positions and velocities of the two particles at discrete times $t^{(n)}, t^{(n)} = n\Delta t$. Algorithm 1 is a description of the solution process in pseudocode.

The results of a simulation are shown in Fig. 3.2. In this simulation the particles started at a distance equal to the equilibrium length from each other, and the rightmost particle was given an initial velocity to the right.

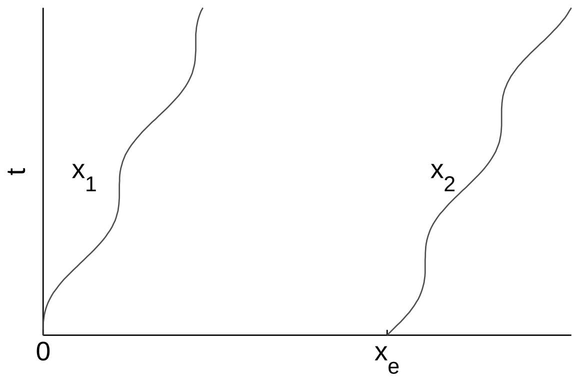

Figure 3.2: Positions of two particles connected by a spring as a function of time with $x_1^{(0)} = 0$, $x_2^{(0)} = x_e$, $v_1^{(0)} = 0$ and $v_2^{(0)} = 0.1$

3.1.2 N_p particles connected with a spring

We now generalize the system to contain N_p particles. Consider the system displayed in Fig. 3.3 consisting of N_p particles that can move in one dimension and are connected with springs. The positions of the particles

Algorithm 1 Simulate trajectory of the structure in Fig. 3.1 using Verlet

```
1: Set system parameters $k$, $m$ and $x_\mathrm{e}$.
2: Set initial conditions,
   $$\vec{x}^{(0)} \leftarrow \begin{bmatrix} x_1^{(0)} & x_2^{(0)} \end{bmatrix}^\mathrm{T} \quad \text{and} \quad \vec{v}^{(0)} \leftarrow \begin{bmatrix} v_1^{(0)} & v_2^{(0)} \end{bmatrix}^\mathrm{T}.$$
3: Set simulation parameters $t_\mathrm{f}$ and $\Delta t$
4: Compute number of time increments (here, the notation $\lceil x \rceil$ means to
   round $x$ upwards to the closest integer),
   $$N_\mathrm{t} \leftarrow \lceil t_\mathrm{f}/\Delta t \rceil$$
5: Allocate time vector $\mathbf{t}$, position solution matrix $\mathbf{x}$ and velocity matrix
   $\mathbf{v}$. $\mathbf{t}$ is of length $N_\mathrm{t}+1$ and $\mathbf{x}$ and $\mathbf{v}$ are of size $(N_\mathrm{t}+1) \times 2$ (since there
   are 2 particles).
6: Initialize the loop variables; the positions at the current timestep $\vec{x}^n$ and
   the positions at the next time step $\vec{x}^{n+1}$,
   $$\vec{x}^{(n)} \leftarrow \vec{x}^{(0)} - \Delta t \vec{v}^{(0)} \quad \text{and} \quad \vec{x}^{(n+1)} \leftarrow \vec{x}^{(0)}, \quad \text{respectively.}$$
7: for $n = 0$ to $N_\mathrm{t}$ do
8:     Update loop variables,
       $$x^{(n-1)} \leftarrow x^{(n)}, \quad x^{(n)} \leftarrow x^{(n+1)}.$$
9:     Compute forces at current time step,
       $$\vec{f}^{(n)} \leftarrow k(x_2^{(n)} - x_1^{(n)} - x_\mathrm{e}) \cdot \begin{bmatrix} 1 & -1 \end{bmatrix}.$$
10:    Compute accelerations at current time step,
       $$\vec{a}^{(n)} \leftarrow \vec{f}^{(n)}/m.$$
11:    Compute positions at next time step,
       $$\vec{x}^{(n+1)} \leftarrow 2\vec{x}^{(n)} - \vec{x}^{(n-1)} + \Delta t^2 \vec{a}^{(n)}.$$
12:    Compute velocities at current time step (derivation similar to that of
       Verlet, i.e. use Taylor series),
       $$\vec{v}^{(n)} \leftarrow (\vec{x}^{(n+1)} - \vec{x}^{(n-1)})/2\Delta t.$$
13:    Store positions and velocities at current time step.
14: end for
15: Write results to plot.
```

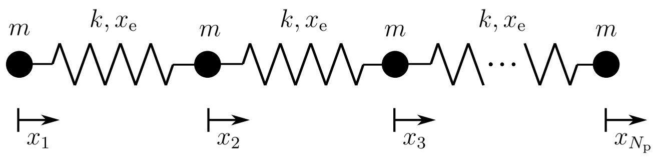

Figure 3.3: N_p particles connected by $N_p - 1$ springs

are $x_i, i = 1..N_p$, all particles have mass m and the springs have stiffness k and equilibrium length x_e.

Writing down the equations of motion for the particles gives

$$m\ddot{x}_i = f_i, \quad i = 1..N_p,$$

where

$$\begin{aligned} f_1 &= k\left[(x_2 - x_1) - x_e\right], \\ f_2 &= -k\left[(x_2 - x_1) - x_e\right] + k\left[(x_3 - x_2) - x_e\right], \\ f_3 &= -k\left[(x_3 - x_2) - x_e\right] + k\left[(x_4 - x_3) - x_e\right], \\ &\vdots \\ f_{N_p} &= -k\left[(x_{N_p} - x_{N_p - 1}) - x_e\right]. \end{aligned}$$

We can write the expressions for the forces more concisely as

$$f_i = \begin{cases} k\left[(x_{i+1} - x_i) - x_e\right], & i = 1 \\ k\left[x_{i+1} - 2x_i + x_{i-1}\right], & 1 < i < N_p \\ -k\left[(x_i - x_{i-1}) - x_e\right], & i = N_p \end{cases}, \qquad i = 1..N_p.$$

A result of a simulation with 5 particles is shown in Fig. 3.4. In this simulation all particles started at a distance equal to the equilibrium length from each other except the rightmost particle, whose distance from its left neighbour is larger than the equilibrium length. The initial velocity of all particles was zero.

3.1.3 Debugging using laws of physics

An MD simulation should always satisfy the laws of physics. This therefore provides a basic check for the correctness of your implementation; if a law isn't satisfied in your program, there is definitely something wrong.

Conservation of energy

In conservative systems such as the ones considered so far the law of conservation of energy should hold. The total energy present initially is preserved in time and changes form between potential and kinetic energy. For particles

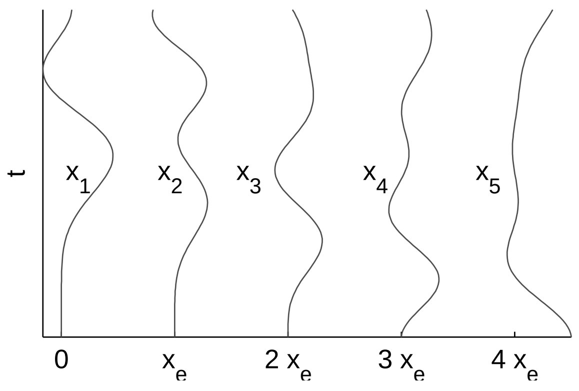

Figure 3.4: Positions of 5 particles connected by a spring as a function of time

connected with springs the potential energy is found by summing the potential energy stored in each spring. In the last case of N_p particles connected with springs this can be expressed as

$$U = \sum_{i=1}^{N_\mathrm{p}-1} \frac{1}{2} k \left[(x_{i+1} - x_i) - x_\mathrm{e} \right]^2 .$$

The kinetic energy can be computed by summing the kinetic energy of all particles, which for the case of 1D motion is expressed as

$$E_\mathrm{k} = \sum_{i=1}^{N_\mathrm{p}} \frac{1}{2} m v_i^2 .$$

The total energy E_tot is the sum of the kinetic and potential energy;

$$E_\mathrm{tot} = E_\mathrm{k} + U.$$

To check whether conservation of energy is satisfied one can compute the potential and kinetic energy at each time step, and after termination of the simulation plot the total energy as a function of time.

The lines to add are in pseudocode;

```
1: Allocate kinetic and potential energy solution vectors $\mathbf{E}_\mathrm{k}$ and $\mathbf{U}$, each
   of length $N_\mathrm{t} + 1$, the total number of time increments plus.
2: Compute kinetic energy at current time step
       $$E_\mathrm{k}^{(n)} \leftarrow \sum_{i=1}^{N_\mathrm{p}} \frac{1}{2} m \left( v_i^{(n)} \right)^2 .$$
```

```
3: Compute potential energy at current time step
       $$U^{(n)} \leftarrow \sum_{i=1}^{N_\mathrm{p}-1} \frac{1}{2}k\left[\left(x_{i+1}^{(n)} - x_i^{(n)}\right) - x_\mathrm{e}\right]^2.$$
4: Store kinetic and potential energy at current time step at $n+1$th position
   in solution vectors $\mathbf{E}_\mathrm{k}$ and $\mathbf{U}$.
```

A good place to add line 1 is between lines 5 and 6 in Algorithm 1. Lines 2..4 in the above code could be added between lines 12 and 13.

Figure 3.5 shows a plot of the kinetic, potential and total energy as a function time as computed in the simulation with N_p particles. As can be seen, the total energy remains constant.

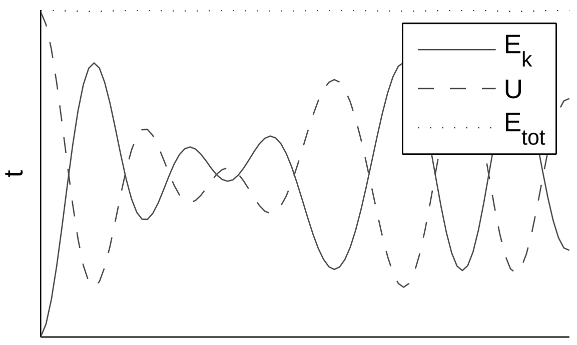

Figure 3.5: Kinetic, potential and total energy as a function of time

Newton's third law

Another helpful check is provided by Newton's third law; the action-reaction law. The forces from particle i on particle j should always be equal and opposite to that of particle j on particle i, $i \neq j$.

Ex. 4 — Implement steps to check if Newton's third law is satisfied.

3.2 Particles in two dimensions connected with springs

This section considers the motion of particles connected to each other with linear springs and free to move in two dimensions. It is discussed how to simulate this motion using molecular dynamics.

We start by showing how the forces on two particles connected with a spring and free to move in two dimensions can be calculated in Section 3.2.1.

This is generalized to the calculation of the forces on an arbitrary number of particles in Section 3.2.2. An algorithm for simulating the motion of an arbitrary number of particles is outlined in Section 3.2.3. Finally, Section 3.2.4 introduces the modeling of solids using molecular dynamics.

3.2.1 Two particles in two dimensions connected with a spring

Figure 3.6 displays two particles arbitrarily located in 2D space connected with a linear spring. It is shown how to compute the forces on particles i and j given their positions and the stiffness and equilibrium length of the spring.

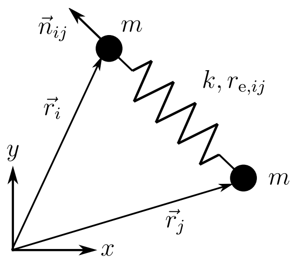

Figure 3.6: Two particles connected by spring in 2D

The magnitude of the force exerted by the spring on the particles, f_{ij}, can be computed by multiplying the spring stiffness, k, with the spring compression, δ. The compression is given by substracting the equilibrium length of the spring between particles i and j, $r_{\mathrm{e},ij}$, from the length of the vector pointing from particle i to particle j, $|\vec{r}_i - \vec{r}_j|$ and multiplying this with -1. This is stated as

$$f_{ij} = k\delta, \quad \text{with} \quad \delta = -\left(|\vec{r}_i - \vec{r}_j| - r_{\mathrm{e},ij}\right), \tag{3.2}$$

where $|\vec{x}| = \sqrt{(\vec{x})^{\mathrm{T}} \cdot \vec{x}}$ for a column vector $\vec{x}$. Note that a positive f_{ij} means the spring is in compression whereas a negative magnitude of the force means the spring is in tension.

The force exerted on particle i by the spring between particles i and j, $\vec{f}_{ij}$, can be calculated by multiplying the magnitude of the spring force, f_{ij}, with the unit vector pointing to particle i from particle j, $\vec{n}_{ij}$, or

$$\vec{f}_{ij} = f_{ij}\vec{n}_{ij}, \tag{3.3}$$

where the unit vector pointing to particle i from particle j is given by the vector pointing from particle i to particle j, divided by its length, or

$$\vec{n}_{ij} = \frac{\vec{r}_i - \vec{r}_j}{|\vec{r}_i - \vec{r}_j|}. \tag{3.4}$$

The force exerted on particle j by the spring between particles i and j, $\vec{f}_{ji}$ is equal in magnitude and opposite in direction to $\vec{f}_{ij}$, or

$$\vec{f}_{ji} = -\vec{f}_{ij}$$

.

3.2.2 N_p particles in two dimensions connected with springs

Figure 3.7 shows four out of N_p particles with masses m and positions $\vec{r}_i, i = 1..N_\mathrm{p}$, arbitrarily located in 2D space, connected with linear springs with stiffnesses k and equilibrium lengths r_eq.

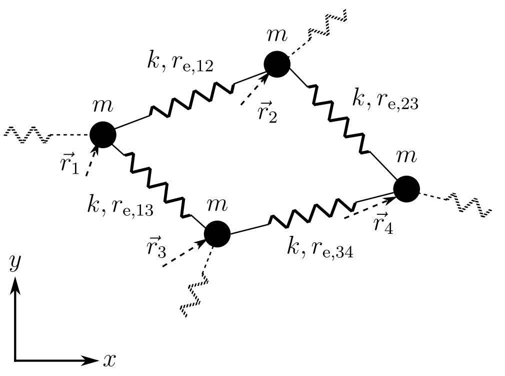

Figure 3.7: N_p particles connected by springs in 2D

In computing the total force on each particle, the difference with the case with 2 particles is that each particle can now be connected with more than one spring. To compute the total force on a particle, we must sum the forces exerted by each spring connected to the particle. We can express the force on particle i, $\vec{f}_i$, as

$$\vec{f}_i = \sum_{j=1}^{N_\mathrm{p}} C_{ij} \vec{f}_{ij}, \tag{3.5}$$

where $\vec{f}_{ij}$ is the force exerted by the spring connecting particle i and j on particle i, given by Eq. (3.3), and C_{ij} is given by

$$C_{ij} = \begin{cases} 1 & \text{if particles } i \text{ and } j \text{ are connected,} \\ 0 & \text{if particles } i \text{ and } j \text{ are not connected or } i = j. \end{cases} \tag{3.6}$$

Both the equilibrium lengths, $r_{\mathrm{eq},ij}$ and the connectivity coefficients, C_{ij}, $i = 1..N_\mathrm{p}$, $j = 1..N_\mathrm{p}$, are matrices. In the 'equilibrium length matrix' the cell in the ith row and the jth column contains the equilibrium length of the spring between particles i and j. In the 'connectivity matrix' the cell in the ith row and the jth column contains a 1 if particles i and j are connected and a 0 if they are not. Note that therefore $r_{\mathrm{eq},ij} = r_{\mathrm{eq},ji}$ and $C_{ij} = C_{ji}$, i.e. the equilibrium length matrix and connectivity matrix are symmetric.

3.2.3 An algorithm for 2D molecular dynamics

This section describes an algorithm for simulating the trajectories of N_p particles in 2D connected with linear springs. It starts by presenting the algorithm in pseudocode on a high level. This is preceded by some remarks on the notation of vectors describing positions, velocities, forces, etc. for all particles at once. Then a subsection follows describing the importance of the force calculation part in the algorithm, and a way to improve its efficiency. This section ends with a subsection containing remarks on satisfying constraints in an MD simulation.

In simulating the trajectories we solve Eq. (3.1) for the case of motion of N_p particles connected with linear springs in 2D. This can be stated as solving the system of differential equations

$$m\ddot{\vec{r}}_i = \vec{f}_i, \quad i = 1..N_\mathrm{p}, \tag{3.7}$$

where

$$\vec{r}_i = \begin{bmatrix} x_i & y_i \end{bmatrix}^\mathrm{T}$$

and $\vec{f}_i$ is given by Eq. (3.5).

These equations can be numerically integrated using e.g. the Verlet method discussed in Section 2.3.3, given the initial positions and velocities of the particles. Note that in order to compute the forces on the particles, $\vec{f}_i,\ i = 1..N_\mathrm{p}$, we also need to specify the spring stiffnesses, the equilibrium lengths and which particles are connected to each other.

In 1D we chose to collect all quantities such as positions x_i, velocities v_i and forces f_i, $i = 1..N_\mathrm{p}$ in single position, velocity and force vectors $\vec{x}$, $\vec{v}$ and $\vec{f}$, respectively. This was straightforward, since in the 1D case the positions and velocities are scalars. In 2D the positions, velocities and forces etc. are vectors $\vec{r}_i$, $\dot{\vec{r}}_i$ and $\vec{f}_i$, $i = 1..N_\mathrm{p}$, instead of scalars, which makes it less obvious to think of a vector describing the positions and velocities of all particles. In the following, two such vectors are discussed. One way of denoting the position vector is as

$$\vec{r} = \begin{bmatrix} x_1 & y_1 & x_2 & y_2 & \cdots & y_{N_\mathrm{p}} \end{bmatrix}^\mathrm{T}.$$

Here x and y components of position follow each other in order of the particle numbering. An equivalent notation, emphasizing it can be viewed as a collection of vectors, is

$$\begin{bmatrix} (\vec{r}_1)^\mathrm{T} & (\vec{r}_2)^\mathrm{T} & \cdots & (\vec{r}_{N_\mathrm{p}})^\mathrm{T} \end{bmatrix}^\mathrm{T},$$

where

$$\vec{r}_i = \begin{bmatrix} x_i & y_i \end{bmatrix}^\mathrm{T}, \quad i = i..N_\mathrm{p}.$$

The second way of writing such a vector would be to write

$$\vec{r} = \begin{bmatrix} x_1 & x_2 & \cdots & x_{N_\mathrm{p}} & y_1 & y_2 & \cdots & y_{Np} \end{bmatrix}^\mathrm{T}.$$

Here, first all the x positions are written in order of particle numbering, followed by all the y positions in order of particle numbering.

In this script we define the position, velocity and force vector in the first way.

An algorithm of simulating the trajectories of particles in 2D connected with linear springs is given in Algorithm 2 in pseudocode. Note that the integration loop is described in a separate part due to space limitations.

Efficient computation of forces

In an algorithm for molecular dynamics most of the computation is spent on computing the forces working on the particles. Making the force calculation efficient is therefore important. This section outlines a way of saving computation.

About half of the computations in the force calculation can be saved by making use of Newton's third law, the action-reaction law; we only have to evaluate the force for each unique particle pair, not each particle.

The force calculation described in Algorithm 4 describes a way of computing the forces on all particles by evaluating the force only for each unique particle pair. It works by looping over all particles but the last one, where for each particle only the interactions between it and the particles with higher particle numbers are considered. Now, not only is the interaction between particles i and j at the current time step, $\vec{f}_{ij}^{(n)}$, added to the total force on particle i, also the interaction with equal magnitude but opposite direction, $-\vec{f}_{ij}^{(n)}$, which is the interaction between particles j and i at the current time step, $\vec{f}_{ji}^{(n)}$, is added to the total force on particle j.

Computation of potential and kinetic energy

The potential energy is a sum over all unique particle pairs. It is given by

$$U = \sum_{i=1}^{N_\mathrm{p}-1} \sum_{j=i+1}^{N_\mathrm{p}} C_{ij} \frac{1}{2} k \left(|\vec{r}_i - \vec{r}_j| - r_{\mathrm{eq},ij} \right)^2,$$

where C_{ij} is 1 if particles i and j are connected and 0 otherwise. The kinetic energy is given by

$$E_k = \sum_{i=1}^{N_\mathrm{p}} \frac{1}{2} m \left| \vec{v}_i \right|^2.$$

Algorithm 2 Simulating the 2D motion of N_p particles connected with linear springs using Verlet

```
1: Define structure;
     • Initial positions,
         $$\vec{r}^{(0)} = \left[ \left(\vec{r}_1^{(0)}\right)^\mathrm{T} \quad \left(\vec{r}_2^{(0)}\right)^\mathrm{T} \quad \cdots \quad \left(\vec{r}_{N_\mathrm{p}}^{(0)}\right)^\mathrm{T} \right]^\mathrm{T}$$
       and initial velocities
         $$\dot{\vec{r}}^{(0)} = \left[ \left(\dot{\vec{r}}_1^{(0)}\right)^\mathrm{T} \quad \left(\dot{\vec{r}}_2^{(0)}\right)^\mathrm{T} \quad \cdots \quad \left(\dot{\vec{r}}_{N_\mathrm{p}}^{(0)}\right)^\mathrm{T} \right]^\mathrm{T},$$
       with
         $$\vec{r}_i^{(0)} = \left[ x_i^{(0)} \quad y_i^{(0)} \right]^\mathrm{T} \quad \text{and} \quad \dot{\vec{r}}_i^{(0)} = \left[ u_i^{(0)} \quad v_i^{(0)} \right]^\mathrm{T}, \quad i = 1..N_\mathrm{p}.$$
     • Particle connectivity, e.g. connectivity matrix by stating Eq. (3.6) for $i = 1..N_\mathrm{p},\ j = 1..N_\mathrm{p}$.
     • Spring properties; stiffness $k$ and equilbrium lengths $r_{\mathrm{e},ij},\ i = 1..N_\mathrm{p},\ j = 1..N_\mathrm{p}$.
2: Set simulation parameters, the end time $t_\mathrm{f}$ and the size of the timestep $\Delta t$.
3: Compute number of timesteps,
         $$N_\mathrm{t} \leftarrow \lceil t_\mathrm{f}/\Delta t \rceil.$$
4: Allocate time vector $\mathbf{t}$, position solution matrix $\mathbf{r}$ and velocity matrix $\dot{\mathbf{r}}$.
5: Initialize the loop variables; the positions at the current timestep $\vec{r}^n$ and the positions at the next time step $\vec{r}^{n+1}$,
         $$\vec{r}^{(n)} \leftarrow \vec{r}^{(0)} - \Delta t \dot{\vec{r}}^{(0)} \quad \text{and} \quad \vec{r}^{(n+1)} \leftarrow \vec{r}^{(0)}.$$
6: Integrate, see Algorithm 3.
7: Write results to plot.
```

Algorithm 3 The integration loop for simulating particles in on a 2D domain connected with linear springs

```
1: for $n = 0$ to $N_\mathrm{t}$ do
2:     Update loop variables,
           $$r^{(n-1)} \leftarrow r^{(n)}, \quad r^{(n)} \leftarrow r^{(n+1)}.$$
3:     Compute forces at current time step,
           $$\vec{f}^{(n)} \leftarrow \left[ \left(\vec{f}_1^{(n)}\right)^\mathrm{T} \quad \left(\vec{f}_2^{(n)}\right)^\mathrm{T} \quad \cdots \quad \left(\vec{f}_{N_\mathrm{p}}^{(n)}\right)^\mathrm{T} \right]^\mathrm{T},$$
       with
           $$\vec{f}_i^{(n)} \leftarrow \sum_{j=1}^{N_\mathrm{p}} C_{ij}\vec{f}_{ij}, \quad i = 1..N_\mathrm{p}, \quad \vec{f}_{ij} \text{ given by Eq. (3.3).}$$
       For a detailed description, see Algorithm 4.
4:     Compute accelerations at current time step,
           $$\ddot{\vec{r}}^{(n)} \leftarrow \vec{f}^{(n)}/m.$$
5:     Compute positions at next time step,
           $$\vec{r}^{(n+1)} \leftarrow 2\vec{r}^{(n)} - \vec{r}^{(n-1)} + \Delta t^2 \ddot{\vec{r}}^{(n)}$$
6:     Compute velocities at current time step,
           $$\dot{\vec{r}}^{(n)} \leftarrow \left(\vec{r}^{(n+1)} - \vec{r}^{(n-1)}\right)/2\Delta t.$$
7:     Store positions and velocities at current time step.
8: end for
```

Algorithm 4 Force calculation such that each particle pair is considered only once

```
1: Initialize particle forces at current time step,
       $$\vec{f}_i^{(n)} \leftarrow 0, \quad i = 1..N_\mathrm{p}.$$
2: for $i = 1$ to $N_p - 1$ do
3:    for $j = i + 1$ to $N_p$ do
4:       if particles $i$ and $j$ are connected, or $C_{ij} = 1$, then
5:          Compute distance between particles,
                $$d_{ij} \leftarrow |\vec{r}_i - \vec{r}_j|.$$
6:          Compute magnitude of force in spring between particles $i$ and $j$,
                $$f_{ij} \leftarrow k\delta, \quad \text{with} \quad \delta = -\left(d_{ij} - r_{\mathrm{e},ij}\right).$$
7:          Compute unit vector pointing to particle $i$ from particle $j$,
                $$\vec{n}_{ij} \leftarrow \frac{\vec{r}_i - \vec{r}_j}{d_{ij}}.$$
8:          Compute force exerted on particle $i$ by particle $j$,
                $$\vec{f}_{ij} \leftarrow f_{ij}\vec{n}_{ij}.$$
9:          Add this force to the total force on particle $i$,
                $$\vec{f}_i^{(n)} \leftarrow \vec{f}_i^{(n)} + \vec{f}_{ij}.$$
10:         Add this force to the total force on particle $j$,
                $$\vec{f}_j^{(n)} \leftarrow \vec{f}_j^{(n)} - \vec{f}_{ij}.$$
11:      end if
12:   end for
13: end for
```

In the 2D case the position and velocity vectors $\vec{r}_i$ and $\vec{v}_i$ are given by $\vec{r}_i = \begin{bmatrix} x_i & y_i \end{bmatrix}^{\mathrm{T}}$, and $\vec{\dot{r}}_i = \begin{bmatrix} u_i & v_i \end{bmatrix}^{\mathrm{T}}$, respectively, where u_i and v_i are the velocities in x and y direction, respectively. Note that the forms are, however, equally valid for the case of particles moving in one- or threedimensional space (or even higher, if that makes sense).

Ex. 5 — Add the computation of the potential and kinetic energy to the algorithm outlined in this section. Tip: since the force calculation is also a sum over all unique particle pairs, add the computation of the potential energy to the loops for the force calculation.

Satisfying constraints

The algorithm described in the previous section considers a structure without any constraints; i.e. the motion of the particles is determined fully by their interactions with each other. However, usually one or more particles are subject to constraints. For example, the motion in a certain direction could be suppressed, the particles could be subject to an external force, or their motion could be constrained to follow a certain trajectory.

In this course it is probably best to satisfy these constraints by adding lines to the algorithm described, but more general methods for satisfying constraints have been developed, e.g. Lagrange multipliers [1].

3.2.4 Modeling a solid

Thinking of solids as atoms in a regular pattern, subject to each others attraction and repulsion, provides accurate descriptions of the large scale properties we are familiar with.

Figure 3.8 depicts a configuration of particles and springs that leads to the well known behaviour of solids.

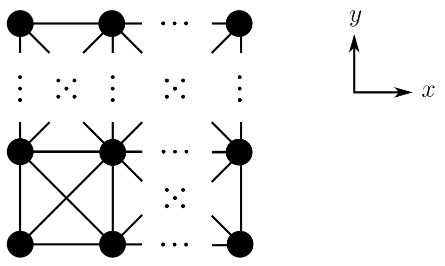

Figure 3.8: Particle configuration giving solid-like behaviour, the solid lines represent springs

The lattice consists of particles arranged in a rectangular pattern, each particle connected horizontally, vertically and diagonally with linear springs

[1]See http://en.wikipedia.org/wiki/Constraint_algorithm

to other particles.

In analyzing the solid-like behaviour of this material, one could observe the forces appearing in the springs as one displaces an edge of the material quasistatically, i.e. sufficiently slow. In the following, definitions of normal stress and strain and shear stress and strain are given that lead to the well known relations between them.

Normal stress and strain

Imagine displacing all particles on one side, i.e. an edge, in the direction perpendicular to it. This is a normal deformation. See the left part of Fig. 3.9, where the right edge of the structure is displaced in x direction, creating tensile force F.

The normal stress in the x direction, denoted σ_{xx}, is defined as follows. Imagine a straight line s perpendicular to the direction of deformation, across the structure, cutting away the part of the structure on the side of the line corresponding to the chosen direction. We now define the normal stress, σ_{xx}, by the total force, F, required on the cut-away side to make equilbrium in this direction [2], divided by the length of the line $L_{1,y}$, or

$$\sigma_{xx} = \frac{F}{L_{1,y}}.$$

The normal strain in a the x direction, denoted ε_{xx}, is defined as the total displacement of a particle in that direction, $L_{1,x} - L_{0,x}$, divided by its initial distance, $L_{0,x}$, or

$$\varepsilon_{xx} = \frac{L_{1,x} - L_{0,x}}{L_{0,x}}.$$

The shear stress and strain

Now imagine displacing an edge in the direction along it, a shear deformation. See the right part of Fig. 3.9, where the right edge of the structure is displaced in the y direction, creating a shear force, F.

The shear stress in the x direction, denoted σ_{xy}, is defined as the total force, F, required to satisfy equilibrium in the direction along the line, divided by the length of the line, $L_{0,y}$, or

$$\sigma_{xy} = \frac{F}{L_{0,y}}.$$

The shear strain in the x direction, denoted ε_{xy}, is defined as the total displacement of a particle in the direction $L_{1,y} - L_{0,y}$, divided by its initial

[2]Actually, the force F equal in magnitude and opposite in direction to the sum of the forces in the cut springs merely approximates the force required to make equilibrium, for the structure is moving during a quasi-static displacement

distance perpendicular to that direction $L_{0,x}$, or

$$\varepsilon_{xy} = \frac{L_{1,y} - L_{0,y}}{L_{0,x}}.$$

Stress-strain relations and Poisson's ratio

One can plot the normal stress in the direction of displacement versus the normal strain in that direction, and the line would be straight. The slope corresponds to Young's modulus.

One can plot the shear stress in the direction of displacement versus twice the shear strain in that direction, and the line would be straight. The slope corresponds to the shear modulus.

One can plot the normal strain in the direction of displacement versus the normal strain in the direction perpendicular to that, and the line would be straight. The slope multiplied with -1 corresponds to Poisson's ratio.

Furthermore, one can verify that the coefficient relating Young's modulus E and the shear modulus G satisfies

$$E = 2(1 + \nu)G$$

where ν is Poisson's ratio.

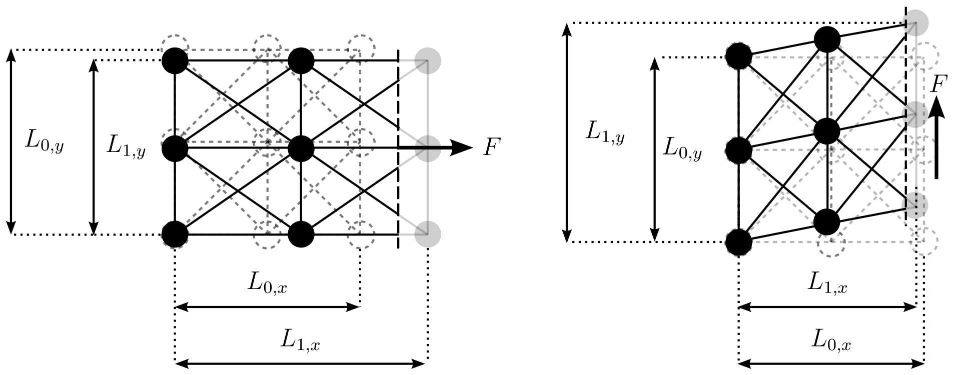

Figure 3.9: A particle lattice in pure normal (left) and pure shear (right) deformation

3.3 Simulating fluids using molecular dynamics

This section first introduces potentials, an alternative to forces in directly specifying the equations of motion [3]. In the second subsection some mathematical and algorithmic differences compared to the previous simulations of solids are discussed, making fluids simulations more accurate and efficient.

[3]Note that although we introduced simulation of solids without potentials, potentials can equally well be used as a starting point for solids simulations.

3.3.1 Potentials

Often, rather than stating equations of motion in terms of forces, they are stated in terms of a potential. The reason is that energy conservation is more easily expressed in terms of potentials. In molecular dynamics, stating equations of motion in terms of potentials is common practice.

For a conservative system, meaning that the total energy of the system does not change over time, the relation between forces, $\vec{f}$, and potential, U, is given by

$$\vec{f} = -\nabla U. \tag{3.8}$$

The force is thus given by the negative gradient of a scalar potential.

In MD, the scalar potential represents the total potential energy of the system under consideration. It is therefore a function of the positions of all particles, $\vec{r}_i$, $i = 1..N_\mathrm{p}$.

The force vector on the ith particle is found by taking the partial derivative of the potential with respect to the position of that particle, or

$$\vec{f}_i = -\frac{\partial}{\partial \vec{r}_i} U. \tag{3.9}$$

The force vector from Eq. 3.8 is thus a vector containing the force components of all particles, or, as in the notation introduced in Section 3.2.3,

$$\vec{f} = \left[\left(\vec{f}_1\right)^\mathrm{T} \quad \left(\vec{f}_2\right)^\mathrm{T} \quad \cdots \quad \left(\vec{f}_{N_\mathrm{p}}\right)^\mathrm{T} \right]^\mathrm{T},$$

where

$$\vec{f}_i = \begin{bmatrix} f_{x,i} & f_{y,i} \end{bmatrix}^\mathrm{T}, \quad i = 1..N_\mathrm{p}.$$

Often, the total potential energy of a system can be expressed as a sum of potentials between pairs of particles, so-called ‘pair potentials’, but this is not true in general [4].

The interaction of particles in conservative systems is thus modeled using potentials. Next two pair potentials are discussed, namely the linear spring and Lennard-Jones potentials.

The linear spring potential

In Section 3.1 and 3.2 we have modeled the interaction between particles as particles being interconnected with linear springs. In the most general case considered, for particles free to move in two dimensions, in Section 3.2.3, the

[4]The accurate modeling of potential energy in conservative systems may need to include interactions between three particles or more.

total potential energy is stated as the sum of the potentials between each unique particle pair, or

$$U = \sum_{i=1}^{N_\mathrm{p}-1} \sum_{j=i+1}^{N_\mathrm{p}} C_{ij} \frac{1}{2} k \left(|\vec{r}_i - \vec{r}_j| - r_{\mathrm{eq},ij}\right)^2,$$

where N_p is the total number of particles, C_{ij} is 1 of particles i and j are connected and 0 if they are not, k is the spring stiffness which is the same for all particle pairs, $\vec{r}_i$ is the position of the ith particles and $r_{\mathrm{eq},ij}$ is the equilibrium distance for the spring between particles i and j.

Ex. 6 — Verify (by using Eq. 3.9) that the force on particle i equals the sum of the forces exerted by the springs connected to particle i.

The Lennard-Jones potential

A potential widely used to model the interaction between pairs of neutral atoms and molecules is the Lennard-Jones potential, given by

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

Here ε and σ are parameters that can be varied to fit the potential between pairs of specific kinds of atoms and molecules [5], i.e. they can be different for each pair. The parameter r is the distance between a pair of particles, or $r = |\vec{r}_i - \vec{r}_j|$. Even though more accurate models of the interactions between atoms and molecules exist, the Lennard-Jones potential is used extensively because computationally it is very simple. A plot of the Lennard-Jones potential is shown in Fig. 3.10.

Except when at the equilibrium position, when $r = \sigma$, the Lennard-Jones potential is nonzero no matter how small or large the distance between a pair of particles, r, is. This implies that all particles interact with each other. In MD this has the implication that at each time step the interaction between all particle pairs must be computed. However, since the force between two particles whose interaction is modeled using the Lennard-Jones potential is very small at large distances (the slope tends to zero as r tends to infinity, see Fig. 3.10), in practice only the interaction of particles whose distance is smaller than some 'cutoff distance', r_c, is computed during simulations.

Since the potentials like the Lennard-Jones potential are not exactly zero at such a cutoff distance, an energy jump occurs each time a particle crosses this distance. To prevent the effects on conservation of energy that would

[5] In the Lennard-Jones potential, ε is the magnitude of the minimum potential energy between a pair of particles, and σ is the is the distance between these particles at which the potential energy is zero. Note that which part of the curve is defined as having a value of zero is arbitrary. I.e. if a specific potential is valid, then so are all the potentials that differ from this potential by an arbitrary constant.

result, usually a shifted version of the Lennard-Jones potential is used such that it is zero at the cutoff distance [6]. I.e. the truncated Lennard-Jones potential, U_{trunc}, satisfies

$$U_{\mathrm{trunc}}(r) = \begin{cases} U(r) - U(r_{\mathrm{c})} & \text{for } r \leq r_{\mathrm{c}}, \\ 0 & \text{for } r > r_{\mathrm{c}}. \end{cases}$$

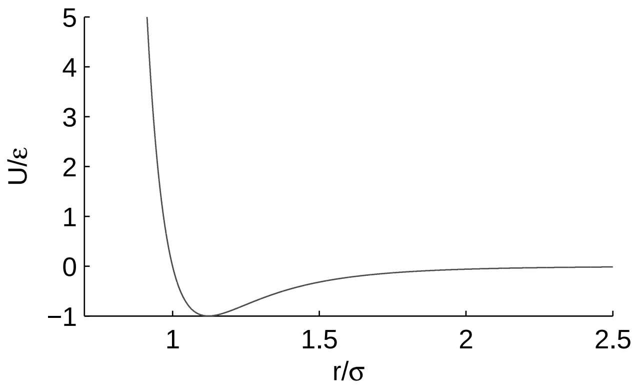

Figure 3.10: Plot of the Lennard-Jones potential

Ex. 7 — Find the expression for the total potential energy of a system of N_{p} particles whose interactions are modeled with the Lennard-Jones potential.

Ex. 8 — Find the expression for the force between a pair of particles whose interaction is modeled with the Lennard-Jones potential.

3.3.2 Efficiently simulating fluids

Compared to the simulations of solids previously performed, the algorithms for simulating fluids can be made more efficient by changing a few things. In order to save memory, particle interaction is not stored in a connectivity matrix. This is discussed in the first subsection. In the second subsection ways of keeping the atoms from flying into infinity are discussed.

Particle interaction

In many solids, e.g. metals, atoms or molecules sit in a regular pattern which doesn't change as the material is deformed. In molecular dynamics this is reflected by interacting pairs that keep interacting, independent of time.

[6]There is still a discontinuity in the force, since the truncated potential energy function is not smooth. This causes extra instability. One way to lessen these effects is by shifting the force function in the same manner as the potential energy function. This does have the effect that the force function is no longer the negative of the derivative of the potential energy function.

In fluids, however, the atoms (for monatomic gases) or molecules move away from their initial configuration. Therefore, during a fluids simulation there is a need to update which particles interact.

For the simulation of solids, 3.2.4, we used a connectivity matrix, 3.2.2, allowing us to look up which particles interact. A possibility for keeping track of particle interaction for fluids is then to update this connectivity matrix, or perhaps better called 'interaction matrix'.

The big disadvantage of using an interaction matrix, however, is that it is neither memory nor computationally efficient. When simulating many particles, you need a lot of memory to store all the interactions simultaneously in addition to having to make the extra computations to read and write the data [7]. A more memory efficient solution is to determine whether particles interact only when their interaction is needed, which is during the force calculation.

In general the condition for interaction between a pair particles in MD simulations is that they are within a certain 'cutoff radius', r_c, of each other. Within this distance the potential of the pair is computed, beyond it, it is not.

Ex. 9 — Modify Algorithm 4 to check for particle interaction by cutoff radius rather than connectivity.

Walls and periodic boundary conditions

The particles in a fluid simulation are, unlike in a solid simulation, not constrained by anything other than their interactions. Particles are therefore free to flow wherever they are pulled or pushed by other particles, and there is nothing stopping them from ending up at ever increasing distances from each other. This is not a realistic situation. In order to create a realistic situation, somehow the particles must stay together. This can be accomplished in three ways. One way is by simulating walls to contain the particles. A second way is by using so-called 'periodic boundary conditions'. A third way is simply by nature of the physics simulated, as in astrophysics. The first two ways are discussed next.

In MD for fluids, walls are usually simulated in two ways.

One is to make a wall out of fluid particles, by fixing particles at a certain distance such that a rigid boundary is created. When the total energy of the system is sufficiently low, no particles will escape the box by passing between two fixed particles.

The other is to implement continuous and smooth walls by adding forces exerted by these walls to the total force on particles. The potential between

[7]The effort to check/update the connectivity matrix is proportional to N_p^2, which is impractical for large N_p. Some techiques for reducing the computational effort are Verlet lists and cell lists.

a particle and a wall is the same as the potential between two particles. In 2D MD this can be implemented by adding the force terms of each of the four walls to the total force on the particle active in the outer loop of the force calculation, and having the outer loop loop over particles $1..N_\mathrm{p}$, rather than loop over particles $1..N_\mathrm{p}-1$ as described in previous algorithms.

The disadvantage of implementing walls is that it creates an unrealistic situation near the walls, where the physics are different than they are away from the walls. The reason the situation is unrealistic is that the ratio of the number of particles close to a wall and the number of all simulated particles is, in any practically possible MD simulation, much larger than it would be in a realistic container [8].

The second option of having particles confined is by applying periodic boundary conditions (PBCs). Instead of simulating a container, this simulates a volume inside an 'infinite' medium. A periodically infinite medium is achieved by creating a box with particles, and adding conditions such that it has the effect of tiling space with the box such that an infinite domain is obtained. Each particle in our original box now has the effect of determining the position of infinitely many other particles, located in the copies of the original box. These copies are called 'images'. Whenever a particle leaves one box, it enters another. In our original box this has the effect of a particle leaving the box at one end appearing at the opposite end with the same velocity. Each particle now not only interacts with the particles in its own box, but also with the particles in the images. Although this way the relative motion of particles inside each of these boxes is identical, the lack of walls has the effect that their motion resembles very well the motion in the inside of a real fluid.

An extra simplification can be made if the lengths in any of the cartesian directions of this box is sufficiently large, compared to the cutoff radius of the potential. It has to be large enough to make sure that each particle can only interact with one copy of each of the other particles at once (it must also not interact with itself). This is achieved if the length, width and height each are larger than the cutoff radius, or

$$r_\mathrm{c} < \min(L_x, L_y, L_z),$$

where L_i is the length in the i-direction, $i = x, y, z$. A typical choice for these lengths is twice the cutoff radius.

The implementation of periodic boundary conditions is quite simple. Just as in simulating solids or fluids in a container, one keeps track of the positions of the 'original' particles. Only during the force calculation, for

[8] In 3D, for $N_\mathrm{p} = 1000$, about $6 \times 10 \times 10 = 600$ particles, or 60%, are close to the wall. For $N_\mathrm{p} = 10^6$, about $6 \times 100 \times 100 = 6 \cdot 10^4$, or 6%, are close to the wall. A volume of hydrogen where the mass of all particles is 1 gram contains a number of particles with an order of magnitude of about 10^{23}, giving only a miniscule fraction of all particles that is close to the wall.

each particle one computes the distance to the closest image of each of the other particles, rather than the distance to the original particles. This is called the 'minimum image convention'.

3.4 Statistical mechanics, connecting microscopic and macroscopic properties

At the macroscopic scale we understand the behaviour of materials in terms of macroscopic quantities such as kinetic energy, potential energy, temperature, density and pressure. There are conditions under which the magnitude of these macroscopic quantities are essentially constant. In these conditions we say the material is in thermodynamic equilibrium.

Yet, when we look at the microscopic level, we see that materials consist of atoms that are in constant motion, bumping into each other. It is not straightforward to understand how macroscopic quantities like temperature and pressure derive from this microscopic behaviour. And while macroscopic quantities like kinetic energy, potential energy and density can be quite easily defined on the microscopic level, it appears they fluctuate at an enormous rate, while on the macroscopic they are measured to be essentially constant [9].

Statistical mechanics is a theory that derives the macroscopic view from the microscopic view. It is based on the assumption that for an isolated system in thermodynamic equilibrium, each microstate, which is a particular configuration of positions and velocities of the particles in the microscopic system, has an equal probability of happening. The constant macroscopic quantities are found as the average of their fluctuating instantaneous values over all microstates. An instantaneous value of a quantity is a number that is found from the positions, velocities and masses of the particles in a microstate.

Since in molecular dynamics each timestep represents a microstate of a system, the macroscopic quantities of such a microscopic system can be approximated from a simulation by generating a lot of microstates and averaging the instantaneous values over all these microstates. Such a collection of microstates is called a statistical ensemble.

Important conditions for these values to be realistic are that the system has to approximate thermodynamic equilibrium, meaning the microscopic quantities fluctuate around a fixed value, and that one averages over enough microstates to approach the value one would get if one would average over all microstates.

In the next subsection, macroscopic quantities are defined in terms of the microscopic quantities. In the subsection following that, other impor-

[9]The microscopic fluctuations do have a macroscopic meaning, see http://en.wikipedia.org/wiki/Thermal_fluctuations.

tant concepts in statistical mechanics are discussed, including how to obtain them from simulations. The last two subsections discuss notes on local instantaneous quantities and on how to simulate in order to keep finding new microstates from the same initial state, respectively.

3.4.1 Obtaining familiar macroscopic quantities from simulations

In this section the definitions of the macroscopic quantities potential energy, kinetic energy, density, temperature and pressure are given in terms of the microscopic quantities; the number of particles, the masses, positions and velocities of the particles, and the forces on those particles.

A macroscopic quantity A is denoted $\langle A \rangle$, which means that it is the average of the computed instantaneous values, $A^{(n)}$, $n = 1..N_\mathrm{t}$, where N_t is the number of instantaneous values considered. In molecular dynamics, the number of instantaneous values considered is the number of timesteps computed. We can state this time average as

$$\langle A \rangle = \frac{1}{N_\mathrm{t}} \sum_{n=1}^{N_\mathrm{t}} A^{(n)}. \tag{3.10}$$

Many instantaneous values are expressed as a sum over particles. In case of a simulation using walls these should be the particles contained within the center region of the container, in order to take particles behaving most like those in the center of a bulk of fluid would behave. A properly sized region considers only the area far away from the wall, e.g. at distances larger than several σ (parameter of Lennard-Jones potential) or λ (the mean free path [10] of the atoms). In case of periodic boundary conditions, the sum of particles can cover all the particles contained initially in the original box.

Potential energy

Given the pairwise potential $U(r)$, where r is the distance between a pair of particles, the average potential energy, $\langle U \rangle$, is found by averaging the instantaneous values

$$U^{(n)} = \sum_{i=1}^{N_\mathrm{p}-1} \sum_{j=i+1}^{N_\mathrm{p}} U(|\vec{r}_i - \vec{r}_j|), \quad n = 1..N_\mathrm{t}. \tag{3.11}$$

[10]The mean free path is the distance an atom travels on average between collisions with other atoms.

Kinetic energy

The average kinetic energy, $\langle E_\mathrm{k} \rangle$, is found by averaging the instantaneous values

$$E_\mathrm{k}^{(n)} = \sum_{i=1}^{N_\mathrm{p}} \frac{1}{2} m_i |\dot{\vec{r}}|^2, \quad n = 1..N_\mathrm{t}. \tag{3.12}$$

Density

The average density, $\langle \rho \rangle$, is found by averaging the sum of the masses of the particles in some sufficiently large region divided by the volume of that region. It is thus found by averaging the instantaneous density

$$\rho^{(n)} = \frac{1}{V} \sum_{i=1}^{N_\mathrm{p}} m_i, \quad n = 1..N_\mathrm{t}, \tag{3.13}$$

where V is the volume (or area in 2D), of the region.

Temperature

The average temperature $\langle T \rangle$ can be found directly from the average kinetic energy $\langle E_\mathrm{k} \rangle$ [11] with

$$\langle E_\mathrm{k} \rangle = \frac{D}{2} N_\mathrm{p} k \langle T \rangle, \tag{3.14}$$

where D is the dimensionality of the system (2 or 3) and k is the Boltzmann constant. The instantaneous value of the temperature can be found by substituting the instantaneous kinetic energy for the average kinetic energy.

Note that since the average kinetic energy is defined in terms of microscopic quantities, indirectly, so is the average temperature. This is not immediately obvious from the previous equation.

Pressure

The average pressure, $\langle P \rangle$, for pairwise interacting particles is found from the virial model of a real gas,

$$\langle P \rangle V = N_\mathrm{p} k \langle T \rangle + \frac{1}{D} \left\langle \sum_{i=1}^{N_\mathrm{p}} \vec{r}_i \cdot \vec{f}_i \right\rangle, \tag{3.15}$$

where V is the volume, D is the dimensionality, $\vec{r}_i$ is the position of the ith particle and $\vec{f}_i$ is the total *internal force* (so excluding the boundary forces

[11] This is based on the equipartition theorem, see http://en.wikipedia.org/wiki/Equipartition_theorem .

[12]) on the ith particle. In terms of the pairwise potential $U(r)$ this results in

$$\langle P\rangle V = N_\mathrm{p} k\langle T\rangle - \frac{1}{D}\left\langle \sum_{i=1}^{N_\mathrm{p}-1}\sum_{j=i+1}^{N_\mathrm{p}} r_{ij}\left.\frac{dU}{dr}\right|_{r=r_{ij}}\right\rangle, \tag{3.16}$$

where r_{ij} is the distance between particles i and j.

Note that just as the average temperature, the average pressure is defined indirectly in terms of microscopic quantities.

Ex. 10 — In case of a simulated container, the pressure can also be found by dividing the total force on a wall by its surface. Check that calculating the pressure in this way gives the same result as Eq. 3.15 / 3.16.

3.4.2 Other important concepts in statistical mechanics

There are many other meaningful things to compute from an MD simulation. Three of such concepts are velocity distributions, the diffusion coefficient and the radial distribution function.

Velocity distributions

The velocities and speeds of the particles in an ideal gas that is in thermodynamic equilibrium are distributed according to the Maxwell-Boltzmann distribution. Sampling the velocities and speeds and plotting their distribution is therefore a way of checking whether the simulation has reached thermodynamic equilibrium.

Each of the velocity components v_i, $i = x, y$ in 2D, is distributed according to a normal distribution with a mean value of zero and a variance of $\sqrt{kT/m}$, where k is the Boltzmann constant, T the temperature and m the particle mass. Thus the velocity destribution in a certain direction of an ideal gas in thermodynamic equilibrium, $f_\mathrm{v}(v_i)$, is given by

$$f_\mathrm{v}(v_i) = \sqrt{\frac{m}{2\pi kT}}\exp\left(\frac{-mv_i^2}{2kT}\right), \quad i = x, y. \tag{3.17}$$

The velocity distribution of the velocity vector, $f_\mathrm{v}(v_x, v_y)$, is given by the product of the velocity distributions in each direction,

$$f_\mathrm{v}(v_x, v_y) = f_\mathrm{v}(v_x) f_\mathrm{v}(v_y). \tag{3.18}$$

[12]In case of periodic boundary conditions there is no explicit boundary. Instead, one takes an arbitrary region, e.g. the original box, and considers all particles outside that region part of the boundary. Thus, forces due to interaction with particles outside the chosen region are not included in the sum of Eq. 3.15

The distribution of the speed, $f(v)$, where the speed, v, is given by the magnitude of the velocity vector, or $v = \sqrt{v_x^2 + v_y^2}$ in 2D , is given by

$$f_v = \frac{m}{kT} v \exp\left(-\frac{mv^2}{2kT}\right), \tag{3.19}$$

The distribution can be found from a simulation in thermodynamic equilibrium as follows. One samples particle velocities at timesteps separated by certain time intervals. One divides the range of velocities and speeds found into partitions, and counts how many samples fall in each partition. Dividing each of these frequencies with the product of the total number of samples and the size of their respective partitions one should find the corresponding distribution.

The size of the time interval between samples should be large enough to allow the velocities to change, typically more than about fifty times the critical timestep, r_c in the case of a dense fluid.

The diffusion coefficient

Another important concept in molecular dynamics is diffusion of mass. It is what makes an odour spread through a room, or a die through a liquid. What happens on the microscopic scale is that atoms spread out through space.

The mean squared distance of a particle gives the average squared distance of a particle with respect to its initial position as a function of time, or

$$\langle r_i^2 \rangle(t) = \left\langle (r_i(t) - r_i(0))^2 \right\rangle, \tag{3.20}$$

where $r_i(t)$ is the position of the ith particle at time t.

A lot of samples can be gathered very quickly by sampling the square mean distance for each particle, in addition to considering each computed microstate as a new set of initial positions. So in the first microstate the particles are starting their first trajectory. At the second microstate the first set of trajectories is at its second timestep, but the trajectories covered by particles from then on are also considered new trajectories. At the third timestep the first set of trajectories is at its third time step, the second set of trajectories at its second, and again a new set of trajectories starts. This way, after n timesteps one will have gathered $(n-1)N_\mathrm{p}$ samples for the distance travelled after one timestep and in general $(n-m)N_\mathrm{p}, m < n$ samples for the distance travelled after m timesteps.

The radial distribution function

The radial distribution function (RDF) is a function that allows one to compute the value of macroscopic quantities from one microstate.

The function describes how, on average, the number density varies as a function of the distance from an arbitrary particle.

The radial distribution function $g(r)$ obtained by computing it for all particles in a single microstate and averaging is stated as

$$g(r) = \frac{2V}{N(N-1)} \frac{1}{V_r} \sum_{i=1}^{N} \sum_{j=1}^{i-1} \theta(r_{ij} - r)\theta(r + \Delta r - r_{ij}), \tag{3.21}$$

where r_{ij} is the distance between particles i and j, $N(N-1)/2$ is the weight such that $g(r)$ tends to 1 as r becomes sufficiently large, $V_r = \pi(2r + \Delta r)\Delta r$ is the area of the ring with inner radius r and width Δr and the function $\theta(x)$ is given by

$$\theta(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases},$$

such that the product $\theta(r_{ij} - r)\theta(r + \Delta r - r_{ij})$ is 1 if particles i and j are within a distance between r and $r + \Delta r$ of each other and 0 otherwise.

This $g(r)$ would be the result from considering a single microstate. A more accurate result can be obtained by averaging the $g(r)$ obtained from several microstates.

3.4.3 Local instantaneous quantities

It can be interesting to plot the evolution of quantities like density in space. This is possible by dividing the box used for simulation into 'bins', and computing local values of the density as the total number of particles in each bin divided by the total area of that bin [13].

3.4.4 Finding new microstates

Since molecular dynamics is a deterministic procedure, given the same initial conditions a simulation will always go through the same microstates. To obtain accurate measurements one would have to wait each time for the simulation to reach thermodynamic equilibrium, and after that the measurements will be identical to ones obtained before. To avoid this, one can save the final microstate of a simulation run to disk, and use this microstate as the initial state of a new run. This way, one can resume from thermodynamic equilibrium and immediately reach new microstates, creating new measurements. Other methods exist, but are not discussed here for the sake of brevity.

[13]This is not discussed further. For more information, see the website and publications of S. Luding at http://www2.msm.ctw.utwente.nl/sluding/.

Chapter 4

The Finite Element Method

The Finite Element Method (FEM) is a numerical method for obtaining approximate solutions to a system of partial differential equations (PDE's) over an arbitrary domain. PDE's arise as a result of mathematical modeling (application of conservation laws) of various physical phenomena, viz. heat conduction, fluid flow, elastic deformation under applied load, electromagnetism etc. The PDE solution can yield insight into the physical phenomena, e.g. stress distribution inside a machine component or the body of a car. The PDE's are often difficult to solve using an analytical approach and therefore one needs to resort to numerical (approximate) solution techniques. Many numerical techniques exist including Finite Difference, Finite volume, Spectral Methods, Discrete Element etc.[1]. The Finite Element Method is simply one amongst many such contenders. Arguably though, many salient mathematical features make FEM an attractive and expedient solution technique for many physical problems, e.g. from structural dynamics. In the following section, we will introduce FEM for very simple systems. To begin with, we will consider a discrete structural system, an example of which are trusses, this system is not only discrete but also algebraic (i.e. involves no PDE) and therefore applying FEM is straightforward (and in this case exact!). However, for more general systems we will see that the approach involves more than a few steps all of which can be easily programmed. In the past few decades this has lead to the emergence of many specialized commercial FEM codes, viz. Abaqus, Ansys, Comsol, Cosmos, LS-Dyna and DiekA [2], to name a few.

PDE's are solved with FEM by assuming the solution varies in a very simple way, e.g. linearly, between a finite number of points on the original domain. The simple way in which the solution varies between these points is described by 'elements'. These elements are joined to form a structure

[1]An overview of numerical techniques for solving partial differential equations can be found on http://en.wikipedia.org/wiki/Numerical_partial_differential_equations

[2]DiekA is a finite element code developed by Han Huétink at the University of Twente to solve non-linear plasticity problems, see http://www.dieka.org/

that approximates the actual structure.

This chapter is divided in four sections.

Section 4.1 discusses the quasistatic limit and introduces what is perhaps the simplest element in the finite element method, the bar element. It is discussed how to apply the finite element method in order to obtain a solution to problems that can be approximated with the static loading of connected bars.

Section 4.2 is on the derivation of the finite element method using the Galerkin method. The Galerkin method is a way of transforming a problem posed as a differential equation with boundary conditions to a problem posed as a system of linear equations. The finite element method can be considered a special case.

Section 4.3 discusses the finite element equations in the dynamic case, where the solution changes with time.

Finally, Section 4.4 discusses applying the finite element method to solve nonlinear equations.

4.1 The finite element method in the quasistatic limit

This section discusses the finite element method when assuming the problem can be approximated by taking the quasisatic limit; the problem solved is assumed to be in static equilibrium.

Section 4.1.1 describes the quasistatic limit, Section 4.1.2 introduces one of the simplest elements used in FEM, the bar element on a one dimensional domain, Section 4.1.3 describes how a simple loading problem is solved using this bar element and Section 4.1.4 describes a generalized bar element that is usable when arbitrarily rotated on a two dimensional domain.

This chapter uses the same notation as Cook[1]; terms representing matrices are surrounded by square brackets, e.g. $[x]$, terms representing column vectors are surrounded by curly brackets, e.g. $\{x\}$, and terms representing row vectors are surrounded by left and right floor brackets, e.g. $\lfloor x \rfloor$.

4.1.1 The quasistatic limit

In the chapter on molecular dynamics, Chapter 3, where the atoms / particles / masses move, obtaining exact solutions required solving the differential equations of motion,

$$m_i \ddot{\vec{r}}_i = \vec{f}_i, \tag{4.1}$$

where m_i is the mass of the ith particle with position $\vec{r}_i$ subject to force $\vec{f}_i$. In many cases, however, the motion of the structure is so subtle that it does not significantly effect the final state. In the equations of motion the inertia terms are negligibly small;

$$m_i\ddot{\vec{r}}_i = \vec{f}_i \ll 1.$$

This limit, in which the loading conditions approach a static loading, is called the quasistatic limit. In these cases solving the equations of motion for the case of static equilibrium,

$$\vec{f}_i = 0,$$

approximates the fully dynamic solution very closely.

4.1.2 The bar element

Consider the elastic bar in Fig. 4.1. It has stiffness k and is attached to the points 1 and 2 at initial positions x_1 and x_2, loaded with external forces F_1 and F_2 and displaced by displacements u_1 and u_2. For a linearly elastic bar the internal forces on node 1 and 2 are given by

$$f_1 = -f_2 = k\Delta L,$$

where $\Delta L = u_2 - u_1$. The displacement varies linearly between the displacements of its endpoints. Also, in contrast to the mass-spring system in Section 3.1.1, the mass is not concentrated in the endpoints of the bar.

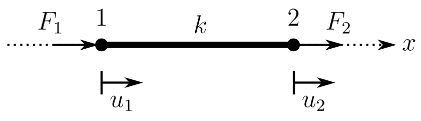

Figure 4.1: The bar element

In case of equilibium we can write down the relations between the external forces and displacements by combining $F_1 = -f_1 = -k\Delta L = k\,(u_1 - u_2)$ and $F_2 = -f_2 = k\Delta L = -k\,(u_1 - u_2)$ into

$$k\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \begin{Bmatrix} F_1 \\ F_2 \end{Bmatrix},$$

or

$$[k]\{u\} = \{f\}, \tag{4.2}$$

where

$$[k] = k\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \tag{4.3}$$

$$\{u\} = \lfloor u_1 \quad u_2 \rfloor^{\mathrm{T}} \text{ and } \{f\} = \lfloor F_1 \quad F_2 \rfloor^{\mathrm{T}}.$$

The matrix $[k]$ is called the stiffness matrix, the vector $\{u\}$ the displacement vector and the vector $\{f\}$ the external force vector.

Suppose we have a cylindrical bar with Young's modulus E, cross-sectional area A and length L, then Eq. (4.2) with $k = EA/L$ represents the equations for static equilibrium of the bar, hence the name bar element [3].

4.1.3 An example of solving a problem with FEM

Figure 4.2 shows a structure with five bars ⓘ with stiffness k_i, $i = 1..5$, cross-sectional areas A_i and lengths L_i attached to four nodes j with displacements u_j, $j = 1..4$. The structure is clamped at its left end $(u_1 = 0)$ andloaded at the right with a force P.

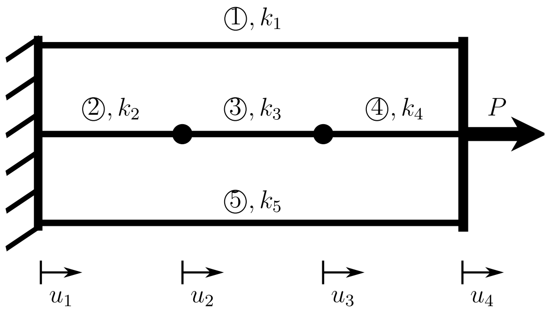

Figure 4.2: Schematic representation of a FEM problem

This structure can be modelled exactly with the finite element method using only bar elements. Combining the bar elements results in a set of four coupled linear equations with four unknowns $u_j, j = 1..4$. These can be solved easily to obtain the displacements, stresses, strains and internal forces of the structure. We do this in five steps.

Step 1: write down equations for each element

The first step is to write down Eq. (4.2) for each element, which gives

$$k_1 \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_4 \end{Bmatrix} = \begin{Bmatrix} F_{1,1} \\ F_{4,1} \end{Bmatrix} \quad \text{for element ①,}$$

$$k_2 \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \begin{Bmatrix} F_{1,2} \\ F_{2,2} \end{Bmatrix} \quad \text{for element ②,}$$

$$k_3 \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} u_2 \\ u_3 \end{Bmatrix} = \begin{Bmatrix} F_{2,3} \\ F_{3,3} \end{Bmatrix} \quad \text{for element ③,}$$

$$k_4 \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} u_3 \\ u_4 \end{Bmatrix} = \begin{Bmatrix} F_{3,4} \\ F_{4,4} \end{Bmatrix} \quad \text{for element ④,}$$

$$k_5 \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_4 \end{Bmatrix} = \begin{Bmatrix} F_{1,5} \\ F_{4,5} \end{Bmatrix} \quad \text{for element ⑤.}$$

where $F_{i,j}$ is the force at node i on element j.

[3]The bar element is also commonly referred to as the 'truss element'.

Step 2: assemble the structure stiffness equations

The second step is to write these equations into a system of linear equations $[K]\{U\} = \{F\}$, called the 'structure stiffness equations', where $\{U\} = \lfloor u_1, u_2, u_3, u_4 \rfloor^{\mathrm{T}}$ is a column vector containing all the displacements and $\{F\} = \lfloor F_1, F_2, F_3, F_4 \rfloor^{\mathrm{T}}$ is a column vector containing the total forces on nodes 1..4, respectively.

An easy way to see how to obtain the structural stiffness equations is to write each of the element equations in a form containing the vector $\{U\}$ instead of only the two displacements corresponding to the nodes attached to the particular element. For example, the equation for element ① is rewritten as

$$\begin{bmatrix} k_1 & 0 & 0 & -k_1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -k_1 & 0 & 0 & k_1 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{Bmatrix} = \begin{Bmatrix} F_{1,1} \\ 0 \\ 0 \\ F_{4,1} \end{Bmatrix}.$$

Doing so for each element and adding the resulting equations gives the assembled equations

$$\begin{bmatrix} k_1 + k_2 + k_5 & -k_2 & 0 & -k_1 - k_5 \\ -k_2 & k_2 + k_3 & -k_3 & 0 \\ 0 & -k_3 & k_3 + k_4 & -k_4 \\ -k_1 - k_5 & 0 & -k_4 & k_1 + k_4 + k_5 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{Bmatrix} = \begin{Bmatrix} F_{1,1} + F_{1,2} + F_{1,5} \\ F_{2,2} + F_{2,3} \\ F_{3,3} + F_{3,4} \\ F_{4,1} + F_{4,4} + F_{4,5} \end{Bmatrix}.$$

Note that the matrix $[K]$ is symmetric. This is generally the case, and provides a way to check for mistakes in the assembly process of a FEM program.

Since the force vector on the right hand side contains the total forces on the nodes, and we know from our assumption of static equilibrium that internal forces are zero we can rewrite it to $\{F\} = \lfloor R, 0, 0, P \rfloor^{\mathrm{T}}$, where R is the reaction force of the structure at the left boundary, node 1.

Step 3: apply boundary conditions

The third step is to apply the boundary conditions. In this case the only boundary condition is that the left boundary does not displace, so $u_1 = 0$. This implies that the first column of $[K]$ can be ignored, leaving four equations in four unknowns (including the reaction force R). Instead of solving the reaction force and the unknown displacements simultaneously, it is easier to first compute the unknown displacements and obtain the reaction force later by substituting the now known displacements in the equation for the reaction force. To solve for the unknown displacements we can discard the first equation, resulting in three equations in three unknowns. This leaves us the following system of equations to be solved.

$$\begin{bmatrix} k_2 + k_3 & -k_3 & 0 \\ -k_3 & k_3 + k_4 & -k_4 \\ 0 & -k_4 & k_1 + k_4 + k_5 \end{bmatrix} \begin{Bmatrix} u_2 \\ u_3 \\ u_4 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ P \end{Bmatrix}.$$

Step 4: solve system of linear equations

The fourth step is to compute the solution to this system of linear equations. There are several ways to do this. An elegant way would be to make use of

$$\{U\} = [K]^{-1}[K]\{U\} = [K]^{-1}\{F\}.$$

So we could compute the inverse of the remainder of the stiffness matrix and postmultiply it with the force vector to obtain $\{U\}$. However, since computing the inverse of a matrix is computationally relatively very expensive, it is better to use other methods. The solution methods in use are divided in direct methods and iterative methods. A familiar example of a direct method is Gaussian elimination. A widely used iterative method is the conjugate gradient method.

In MATLAB the easiest way to solve this system of linear equations would be to use the function `mldivide`, which is called when using the \ operator in MATLAB. This function knows of several solution methods and tries to find the most appropriate for the values it receives as input. Given a matrix `K` and force vector `F` we can compute the solution in MATLAB by executing

```
U = K\F
```

To now compute the reaction force we substitute the solution in the equation we discarded, in the case of this example the first equation, when applying the boundary conditions in step 3. We can now check for errors by verifying the assumption of static equibibium holds. I.e. we verify that the sum of forces external to the structure is zero.

Ex. 11 — Choose some values for $k_i, i = 1..5$ and P and verify static equilibrium by computing `R+P` with MATLAB.

Step 5: compute element strains and stresses

Strain in 1D is defined by

$$\varepsilon = \frac{\partial u}{\partial x}, \tag{4.4}$$

where u is the displacement of the point intially at position x.

Having obtained the displacements $\{U\}$ it is straightforward to compute the element strains ε_i and stresses σ_i, $i = 1..5$.

As mentioned in Section 4.1.2, the displacement of a bar element varies linearly with x, and so we can compute the element strain by only considering its ends, as expressed in

$$\varepsilon_i = \frac{\Delta L_i}{L_i} = \frac{u_{2,i} - u_{1,i}}{L_i}, \; i = 1..5,$$

where ε_i is the strain in bar element i, $u_{1,i}$ and $u_{2,i}$ are the displacements of the first and second node of that bar element, $\Delta L_i = u_{2,i} - u_{1,i}$ is its change in length and L_i is its initial length.

The element stress can be found by Hooke's law,

$$\sigma_i = E_i \varepsilon_i, \ i = 1..5,$$

where σ_i is the stress in element i and E_i is its Young's modulus.

4.1.4 Elements in arbitrary orientation

Consider in Fig. 4.3, a bar element on a two dimensional domain. A way to write down the relation between the element forces and the displacements in the global coordinate system with coordinates x, y is by introducing a local coordinate system with coordinate x'.

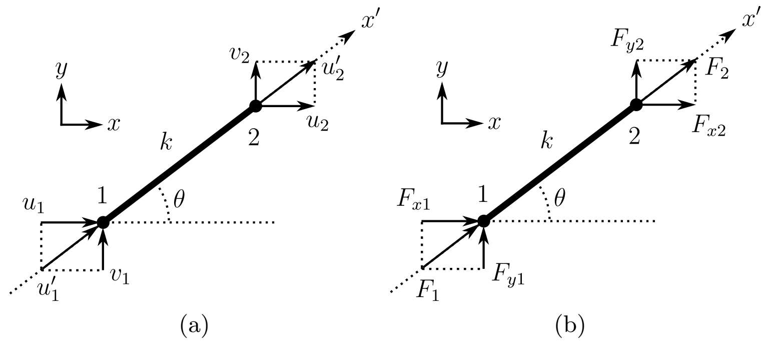

Figure 4.3: Bar element in 2D domain; (a) local and global displacements and (b) local and global forces

From the figure it can be easily verified that

$$u_1 = \cos(\theta) u'_1, \tag{4.5}$$
$$v_1 = \sin(\theta) u'_1, \tag{4.6}$$
$$u_2 = \cos(\theta) u'_2, \tag{4.7}$$
$$v_2 = \sin(\theta) u'_2. \tag{4.8}$$

Multiplying Eq. (4.5) with $\cos\theta$, Eq. (4.6) with $\sin\theta$ and adding we obtain u'_1 expressed in u_1 and v_1. Doing the same with Eqs. (4.7) and (4.8) we obtain u'_2 expressed in u_2 and v_2. We can write these two equations in the form

$$\{u'\} = [T]\{u\}, \tag{4.9}$$

where

$$\{u'\} = \lfloor u'_1 \quad u'_2 \rfloor^{\mathrm{T}}, \quad \{u\} = \lfloor u_1 \quad v_1 \quad u_2 \quad v_2 \rfloor^{\mathrm{T}}$$

and $[T]$ is a so-called transformation matrix, given by

$$[T] = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ 0 & 0 & \cos\theta & \sin\theta \end{bmatrix}. \tag{4.10}$$

Note that $[T]^{\mathrm{T}}[T] = [I]$ and also $\{u\} = [T]^{\mathrm{T}}\{u'\}$.

Similarly we have as a relation between the nodal forces in global coordinates F_{x1}, F_{y1}, F_{x2} and F_{y2} and the nodal forces in local coordinates F_1 and F_2,

$$\{f\} = [T]^{\mathrm{T}}\{f'\}, \tag{4.11}$$

where $\{f'\} = \lfloor F_1, F_2 \rfloor^{\mathrm{T}}$ and $\{f\} = \lfloor F_{x1}, F_{y1}, F_{x2}, F_{y2} \rfloor^{\mathrm{T}}$.

Now a relation between the forces and displacements in the global coordinate system, $\{f\}$ and $\{u\}$, is obtained. From Eq. (4.2) the relation between the forces and displacements in local coordinates is $\{f'\} = [k']\{u'\}$, where $[k']$ is the local stiffness matrix for the bar on a 1D domain, the right hand side of Eq. (4.3). Substituting this expression for $\{f'\}$ in Eq. (4.11) and then substituting Eq. (4.9) we have

$$\{f\} = [T]^{\mathrm{T}}\{f'\} = [T]^{\mathrm{T}}[k']\{u'\} = [T]^{\mathrm{T}}[k'][T]\{u\},$$

or in the notation of Eq. (4.2), $[k]\{u\} = \{f\}$, where $[k]$ now is the stiffness matrix for bar elements in arbitrary orientation on a 2D domain given by

$$[k] = [T]^{\mathrm{T}}[k'][T] = k \begin{bmatrix} c^2 & cs & -c^2 & -cs \\ cs & s^2 & -cs & -s^2 \\ -c^2 & -cs & c^2 & cs \\ -cs & -s^2 & cs & s^2 \end{bmatrix}, \quad c = \cos\theta, \quad s = \sin\theta \tag{4.12}$$

and $\{u\}$ and $\{f\}$ are the displacements and forces in the global coordinate system.

4.2 Formal procedure of deriving the stiffness matrix in FEM

The finite element method started out in a way like described in the previous section. Its application to stress analysis was straightforward enough to be able to use physical insight to understand why it is valid.

To get more out of the finite element method ways were found of putting it on more rigorous foundations. This allowed the derivation of much more of what is and isn't possible using the method. Several approaches to deriving the finite element method have been thought of, making it more and more general so it could be used to solve more and more problems.

This section discusses the Galerkin method, a method capable of converting the problem of solving a continuous description of a problem, a differential equation with boundary conditions, to solving a discretized description of the problem, a system of linear equations. It is used to derive the system of linear equations equivalent to using a single bar element for approximating the solution to a load on a one dimensional domain.

The derivation can be seen as consisting out of three parts. First, in Section 4.2.1, the problem is stated as a differential equation with boundary conditions. This is referred to as the 'strong formulation' of the problem. Next, in Section 4.2.2, the problem is restated as an integral equation called the 'weak formulation' that is equivalent to the strong formulation. In the last step, Section 4.2.3, the discretization takes place. This is the step where the continuous weak formulation is approximated by a system of linear equations.

In the following we denote the first and second derivatives of a function with respect to x as f_x and f_{xx}, respectively, or

$$f_x \equiv \frac{df}{dx} \quad \text{and} \quad f_{xx} \equiv \frac{d^2 f}{dx^2}.$$

4.2.1 Derivation of the strong formulation

In this section the problem of a one dimensional load that varies along the length of a homogeneous bar with constant circular cross-sectional area is stated as a differential equation with boundary conditions. The problem is represented in Fig. 4.4.

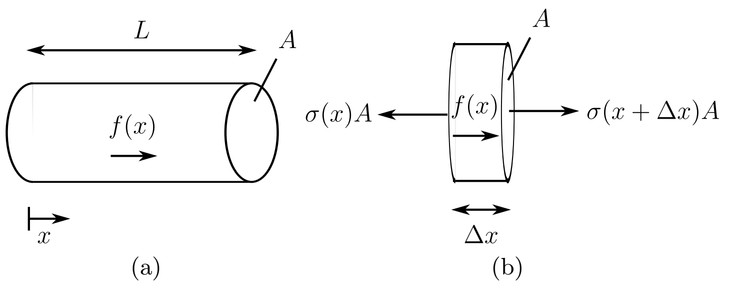

Figure 4.4: (a) the bar element and (b) slice of bar element

The constitutive equation is given by

$$\sigma = E\varepsilon, \tag{4.13}$$

where σ is the stress, E the modulus of elasticity and ε is the strain.

Consider the slice of the bar in Fig. 4.4b. Since we are assuming static equilibrium the sum of the forces acting on it should be zero;

$$\int_{x}^{x+\Delta x} f(s)A\ ds - \sigma(x)A + \sigma(x+\Delta x)A = 0, \tag{4.14}$$

where f is the force density, i.e. the force per unit volume, A is the cross-sectional area and Δx is the length of the element. Let $F(x)$ be the antiderivative of $f(x)$ in this problem, so

$$F_x = f. \tag{4.15}$$

Using Eq. (4.15) we can rewrite Eq. (4.14) to

$$[F(x + \Delta x) - F(x) + \sigma(x + \Delta x) - \sigma(x)]\, A = 0. \tag{4.16}$$

Dividing by $A\Delta x$ and taking the limit of $\Delta x \to 0$, we obtain

$$\lim_{\Delta x \to 0} \left[\frac{F(x + \Delta x) - F(x)}{\Delta x} + \frac{\sigma(x + \Delta x) - \sigma(x)}{\Delta x} \right] = F_x + \sigma_x = 0. \tag{4.17}$$

Substituting Eq. (4.15) we obtain

$$f + \sigma_x = 0. \tag{4.18}$$

Substituting Eq. (4.13) with the strain expressed as $\varepsilon = u_x$, where u is the displacement, gives the strong formulation for the problem depicted in Fig. 4.4;

$$f + Eu_{xx} = 0. \tag{4.19}$$

4.2.2 Derivation of the weak formulation

The next step in the derivation is to rewrite the strong formulation to the so-called 'weak formulation', which is an integral equation equivalent to the strong formulation.

First, we multiply the strong formulation with an arbitrary function v, giving

$$v\,[f + Eu_{xx}] = 0.$$

The function v is also called 'virtual displacement'[4]. As long as $v \neq 0$, this is equivalent to Eq. (4.19).

Next we integrate this equation over the entire volume. To make this form equivalent to the strong formulation, it is required that the integral is zero because Eq. (4.19) holds, not just because the integrand integrated over the particular domain happens to be zero. We therefore require the integral to be zero for any function v, giving

$$\int_0^L v\,[f + Eu_{xx}]\, A dx = 0 \qquad \forall\ v, \tag{4.20}$$

[4]The concept of virtual displacement is in the derivation of the finite element method from the concept of virtual work what the arbitrary function v is in the derivation of the finite element method with the Galerkin method.

where L is the length of the bar.

The product rule states

$$(f \cdot g)_x = f_x \cdot g + f \cdot g_x, \tag{4.21}$$

for any differentiable functions $f(x)$ and $g(x)$. By making use of the product rule (or equivalently, integration by parts) a term representing 'natural' boundary conditions appears, and also the order of the highest occuring derivative reduces to one. Both are desirable properties of a problem to be solved with the finite element method.

With $f = v$ and $g = u_x$ the product rule implies $(vu_x)_x = v_x u_x + vu_{xx}$ or, solving for vu_{xx},

$$vu_{xx} = (vu_x)_x - v_x u_x. \tag{4.22}$$

We can now rewrite Eq. (4.20) to an integral with first derivatives only. First we split the integral on its left hand side;

$$\int_0^L v\,[f + Eu_{xx}]\,Adx = \int_0^L vfAdx + \int_0^L vEu_{xx}Adx. \tag{4.23}$$

Note that even though in this case we could take out E and A out of the integral, given they are constants, in general this is not so. They are kept inside the integrals in order to keep the notation of the derivation closer to that as found in more general derivations.

Next we substitute Eq. (4.22) into the second integral on the right hand side of the last equation. The integral is rewritten to a sum of integrals containing only first order derivatives, as stated in

$$\int_0^L vEu_{xx}Adx = \int_0^L E(vu_x)_x Adx - \int_0^L Ev_x u_x Adx. \tag{4.24}$$

The first integral on the right hand side can be integrated;

$$\int_0^L E(vu_x)_x Adx = [vu_x EA]_0^L\,. \tag{4.25}$$

Substituting Eqs. (4.24) and (4.25) into Eq. (4.23) and reordering terms we obtain a modified weak formulation that only contains first order derivatives,

$$\int_0^L v_x E u_x Adx = [vu_x EA]_0^L + \int_0^L vfAdx \qquad \forall\ v, \tag{4.26}$$

Note that the terms on the right hand side are all known if u_x is known at the boundaries $x = 0$ and $x = L$ and f is known.

In summary. In order to have an integral form equivalent to the strong formulation we multiplied the integrand with an arbitrary function v, and required the integral equation to hold for all v. We applied integration by parts to obtain terms corresponding to natural boundary conditions and to make the highest order derivative occuring in the equations a first order one. Finally we reordered terms to obtain the unknowns on the left and the known terms on the right.

4.2.3 Discretization

We have now rewritten the strong formulation, the differential equation describing the motion of the bar element Eq. (4.19), to its weak formulation, an equivalent form consisting of integrals over the volume of the bar element of Eq. (4.26). Next we discretize the equation leading to a form solvable by a computer.

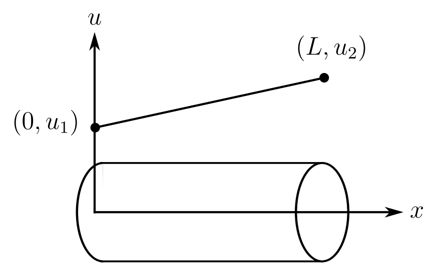

Figure 4.5: Interpolation of displacement for a 1D bar element

In discretizing Eq. (4.26) we assume that the displacement $u(x)$ along the bar varies linearly between its endpoints at $x = 0$ and $x = L$, see Fig. 4.5. The displacement thus satisfies

$$u = ax + b, \quad \text{with} \quad u(0) = u_1,\ u(L) = u_2,$$

where u_1 is the displacement at $x = 0$, u_2 is the displacement at $x = L$, and L is the length of the element.

This implies that $a = (u_2 - u_1)/L$ and $b = u_1$, which can be rewritten in the form

$$u(x) = [N]\{u\}, \tag{4.27}$$

where $[N]$ are the so-called 'basis functions'[5],[6], given by

$$[N] = \begin{bmatrix} 1 - \frac{x}{L} & \frac{x}{L} \end{bmatrix}, \tag{4.28}$$

[5]Just as any vector in a vector space can be represented as a linear combination of the vectors in a basis of that vector space, any function in a function space can be represented as a linear combination of the functions in a basis of that function space. Functions in a basis are called 'basis functions'.

[6]Piecewise linear functions are very well suited as basis functions because they are so simple, but in principle more complicated functions can be (and in practice are) used.

and $\{u\} = \lfloor u_1, u_2 \rfloor^{\mathrm{T}}$ is the displacement vector.

The element strain can be computed by taking the derivative of Eq. (4.27) with respect to x;

$$\varepsilon = u_x = [B]\{u\}, \tag{4.29}$$

where $[B]$ contains the derivatives of the shape functions, thus equal to

$$[B] \equiv [N]_x = \begin{bmatrix} -\frac{1}{L} & \frac{1}{L} \end{bmatrix}. \tag{4.30}$$

In order to end up with a system of linear equations, the form suited for processing with a computer, we no longer require the virtual displacements to be any function, but instead choose them to be of the same form as the displacements. So we choose

$$v(x) = [N]\{v\} \tag{4.31}$$

and also

$$v_x(x) = [B]\{v\}, \tag{4.32}$$

where $\{v\} = \lfloor v_1, v_2 \rfloor^{\mathrm{T}}$.

Substituting Eqs. (4.29) and (4.32) in Eq. (4.26) results in

$$\int_0^L ([B]\{v\})^{\mathrm{T}} E[B]\{u\} A dx = \left[([N]\{v\})^{\mathrm{T}} \sigma A\right]_0^L + \int_0^L ([N]\{v\})^{\mathrm{T}} f A dx.$$

On the right hand side we replaced Eu_x with σ in order to emphasize it is only evaluated at the boundaries and therefore doesn't depend on the displacements $\{u\}$.

Since $([A]\{b\})^{\mathrm{T}} = \{b\}^{\mathrm{T}}[A]^{\mathrm{T}}$ for any matrix $[A]$ and row vector $\{b\}$ and $\{v\}$ and $\{u\}$ are independent of the domain of integration we can rewrite this to

$$\{v\}^{\mathrm{T}} \int_0^L [B]^{\mathrm{T}} E[B] A dx \ \{u\} = \{v\}^{\mathrm{T}} \left[[N]^{\mathrm{T}} \sigma A\right]_0^L + \{v\}^{\mathrm{T}} \int_0^L [N]^{\mathrm{T}} f A dx. \tag{4.33}$$

Since we require this last equation to hold for any $\{v\}$ we may divide both sides by it to obtain

$$[K]\{u\} = \{F\}, \tag{4.34}$$

where $[K]$ is given by

$$[K] = \int_0^L [B]^{\mathrm{T}} E[B] A dx, \tag{4.35}$$

and the force vector $\{F\}$ is given by

$$\{F\} = \{F\}^{\mathrm{b}} + \{F\}^{\mathrm{f}}, \quad \{F\}^{\mathrm{b}} = \left[[N]^{\mathrm{T}} \sigma A\right]_0^L, \quad \{F\}^{\mathrm{f}} = \int_0^L [N]^{\mathrm{T}} f A dx, \tag{4.36}$$

where $\{F\}^{\mathrm{b}}$ is the boundary force vector and $\{F\}^{\mathrm{f}}$ is the volume force vector.

Evaluating the expression for the stiffness matrix Eq. (4.35) we find the same expression we did when introducing the bar element;

$$\begin{aligned}
[K] &= \int_0^L [B]^{\mathrm{T}} E[B] A dx \\
&= EA \int_0^L \begin{bmatrix} -\frac{1}{L} \\ \frac{1}{L} \end{bmatrix} \begin{bmatrix} -\frac{1}{L} & \frac{1}{L} \end{bmatrix} dx \\
&= EA \int_0^L \begin{bmatrix} \frac{1}{L^2} & -\frac{1}{L^2} \\ -\frac{1}{L^2} & \frac{1}{L^2} \end{bmatrix} dx \\
&= EA \begin{bmatrix} \frac{x}{L^2} & -\frac{x}{L^2} \\ -\frac{x}{L^2} & \frac{x}{L^2} \end{bmatrix}_0^L \\
&= \frac{EA}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}.
\end{aligned}$$

Summary

This section discussed a method allowing the derivation of the finite element method, called the 'Galerkin method'.

The method was applied to derive the finite element equations for the case where a loaded bar is modelled with a single bar element.

The Galerkin method works by transforming a problem stated in the form of a differential equation with boundary conditions, called the 'strong formulation', to an integral equation form, called the 'weak formulation'. This weak formulation is then discretized to obtain the finite element equations.

4.3 Dynamic formulation for FEM

This section discusses the application of the finite element method to problems where the solution changes with time.

It starts by deriving the finite element equations of a single bar in the dynamic case (Section 4.3.1). Next modal analysis, where the frequencies and modes of vibration of a structure are determined, using the finite element method is discussed (Section 4.3.2). Finally, the integration of the discretized problem in time by using the Newmark-β method is discussed (Section 4.3.3)

4.3.1 Discretization of the dynamic formulation; the mass matrix

In this section the system of linear equations approximating the solution to the problem depicted in Fig. 4.4 is derived using the Galerkin method. The method can be thought of as consisting out of three steps; stating the strong formulation, restating the strong formulation as the weak formulation and discretizing the weak formulation. This was discussed in more detail in Section 4.2.

In the derivation of the strong formulation in the quasistatic case, Section 4.2.1, the sum forces was assumed to be zero. In this case it is assumed to equal the mass of the bar times its acceleration $\rho A \Delta x\ \ddot{u}$, where ρ is the density, A the cross-sectional area, Δx the width of a slice of the bar and $\ddot{u}$ its acceleration. Following the rest of the derivation it is easy to obtain the strong formulation for the dynamic case:

$$f + E u_{xx} = \rho \ddot{u}. \tag{4.37}$$

The weak formulation is obtained by taking the right hand side of this equation to the left, and following the derivation of obtaining the weak formulation from the strong formulation in the quasistatic case, Section (4.20). The weak formulation is

$$\int_0^L v \rho \ddot{u} A\ dx + \int_0^L (v)_x E u_x A\ dx = EA(v u_x)_0^L + \int_0^L v f A\ dx. \tag{4.38}$$

To be able to solve this equation using a computer we discretize the weak formulation, Eq. (4.38). Since we are considering the dynamic case we now have an extra dimension, time, which will be discretized as well. The discretization of time is discussed in Section 4.3.3. First we discretize the spatial dimension.

In discretizing the quasistatic case, Section 4.2.3, we assumed that both the displacement and the virtual displacements vary linearly between the displacements and virtual displacements of the end points of the bar, as expressed in Eqs. (4.27) and (4.31). For the dynamic case we assume that the same holds for the acceleration; it varies linearly between the accelerations of the end points, or

$$\ddot{u}(x) = [N]\{\ddot{u}\}, \tag{4.39}$$

where $[N]$ is a matrix containing the shape functions, given by Eq. (4.28), and $\{\ddot{u}\} = \lfloor \ddot{u}_1, \ddot{u}_2 \rfloor^{\mathrm{T}}$ is the vector with accelerations of the end points $\ddot{u}_1$ and $\ddot{u}_2$.

Substiuting the expressions for the strain, virtual displacement, virtual strain and the acceleration, Eqs. (4.29), (4.31), (4.32) and (4.39), respectively, in the right hand side of Eq. (4.38) we obtain its discretized form,

$$\int_0^L ([N]\{v\})^{\mathrm{T}} \rho[N]\{\ddot{u}\}A\ dx + \int_0^L ([B]\{v\})^{\mathrm{T}} E[B]\{u\}A\ dx$$
$$= \left[([N]\{v\})^{\mathrm{T}}\sigma A\right]_0^L + \int_0^L ([N]\{v\})^{\mathrm{T}} fA\ dx.$$

On the right hand side we replaced Eu_x with σ in order to emphasize it is only evaluated at the boundaries and therefore doesn't depend on the displacements $\{u\}$.

Since $([A]\{b\})^{\mathrm{T}} = \{b\}^{\mathrm{T}}[A]^{\mathrm{T}}$ for any matrix $[A]$ and row vector $\{b\}$ and $\{v\}$, $\{u\}$ and $\{\ddot{u}\}$ are independent of the domain of integration we can rewrite this to

$$\{v\}^{\mathrm{T}} \int_0^L [N]^{\mathrm{T}}\rho[N]A\ dx\{\ddot{u}\} + \{v\}^{\mathrm{T}} \int_0^L [B]^{\mathrm{T}}E[B]A\ dx\{u\}$$
$$= \{v\}^{\mathrm{T}} \left[[N]^{\mathrm{T}}\sigma A\right]_0^L + \{v\}^{\mathrm{T}} \int_0^L [N]^{\mathrm{T}} fA\ dx. \quad (4.40)$$

Since we require this last equation to hold for any $\{v\}$ we may divide both sides by it to obtain

$$[M]\{\ddot{u}\} + [K]\{u\} = \{F\}, \quad (4.41)$$

where $[K]$ and $\{F\}$ are given by Eqs. (4.35) and (4.36) from the quasistatic case and $[M]$ is the so-called mass matrix, given by

$$[M] = \int_0^L \rho[N]^{\mathrm{T}}[N]A\ dx. \quad (4.42)$$

Here the shape functions $[N]$ are given by Eq. (4.28), or equivalently, $[N] = [N_1\ N_2]$, where $N_1 = 1 - x/L$ and $N_2 = x/L$. With this we can rewrite the mass matrix to

$$[M] = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}, \quad (4.43)$$

where

$$
\begin{aligned}
M_{11} &= \int_0^L \rho N_1^2 A \, dx, \\
M_{12} &= \int_0^L \rho N_1 N_2 A \, dx, \\
M_{21} &= \int_0^L \rho N_2 N_1 A \, dx = M_{12}, \\
M_{22} &= \int_0^L \rho N_2^2 A \, dx,
\end{aligned}
$$

where

$$
\begin{aligned}
N_1^2 &= \left(1 - \frac{x}{L}\right)^2 = 1 - 2\frac{x}{L} + \frac{x^2}{L^2}, \\
N_1 N_2 &= \left(1 - \frac{x}{L}\right)\frac{x}{L} = \frac{x}{L} - \frac{x^2}{L^2}, \\
N_2^2 &= \frac{x^2}{L^2}.
\end{aligned}
$$

The expression for the mass matrix contains the density ρ inside the integral. In order to compute the integral we have to decide how the density varies over the volume of the bar. There are two common ways of assuming the distribution of the density leading to the so-called ‘consistent mass matrix’ and ‘lumped mass matrix’.

The consistent mass matrix

The consistent mass matrix $[M^{\mathrm{C}}]$ follows from assuming the mass is constant along the bar. The coefficients of the matrix become

$$
\begin{aligned}
M_{11}^{\mathrm{C}} &= \rho A \int_0^L N_1^2 \, dx = \frac{\rho A L}{3} = \frac{m}{3}, \\
M_{12}^{\mathrm{C}} = M_{21}^{\mathrm{C}} &= \rho A \int_0^L N_1 N_2 \, dx = \frac{\rho A L}{6} = \frac{m}{6}, \\
M_{22}^{\mathrm{C}} &= \rho A \int_0^L N_2^2 \, dx = \frac{\rho A L}{3} = \frac{m}{3},
\end{aligned}
$$

where $m = \rho AL$ is the mass of the bar element. Substituting the coefficients we obtain the consistent matrix for the bar element in 1D,

$$[M^{\mathrm{C}}] = \frac{m}{6}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}. \tag{4.44}$$

The lumped mass matrix

Another way of assuming the mass is distributed is dividing it in point masses, or lumps, located at the end points of the beam. Mathematically this can be expressed by making the density of the form

$$\rho(x) = \frac{1}{A}\left(\frac{m}{2}\left(\delta(x) + \delta(x-L)\right)\right), \tag{4.45}$$

where $\delta(x)$ is the Dirac delta function, defined by

$$\delta(x) = \begin{cases} \infty, & x = 0 \\ 0, & x \neq 0 \end{cases}, \qquad \int_{-\infty}^{\infty} \delta(x)dx = 1.$$

It has the property

$$\int_a^b \delta(x-c)f(x)dx = f(c), \quad a \le c \le b.$$

Since $\rho(x)$ is zero everywhere except at the end points of the bar the lumped mass matrix $[M^{\mathrm{L}}]$ is given by Eq. (4.42) evaluated only at $x = 0$ and $x = L$. Substituting the expression for ρ, Eq. (4.45), the coefficient M^{L}_{11} becomes

$$\begin{aligned} M^{\mathrm{L}}_{11} &= \int_0^L \rho N_1^2 A dx \\ &= \int_0^L \frac{m}{2}\left(\delta(x) + \delta(x-L)\right) N_1^2 dx \\ &= \frac{m}{2}\left(\int_0^L \delta(x) N_1^2 dx + \int_0^L \delta(x-L) N_1^2 dx\right) \\ &= \frac{m}{2}\left(N_1^2\big|_{x=0} + N_1^2\big|_{x=L}\right) \\ &= \frac{m}{2} \end{aligned}$$

Similarly, the other coefficients become

$$M^{\mathrm{L}}_{12} = M^{\mathrm{L}}_{21} = 0 \quad \text{and} \quad M^{\mathrm{L}}_{22} = \frac{m}{2}.$$

Substituting the coefficients we obtain the lumped mass matrix for a bar element in 1D,

$$[M^{\mathrm{L}}] = \frac{m}{2}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}. \tag{4.46}$$

Mass matrices of a bar arbitrarily rotated on a 2D plane

In 2D the consistent mass and lumped matrices are given by

$$[M^{\mathrm{C}}] = \begin{bmatrix}2 & 0 & 1 & 0\\ 0 & 2 & 0 & 1\\ 1 & 0 & 2 & 0\\ 0 & 1 & 0 & 2\end{bmatrix} \quad \text{and} \quad [M^{\mathrm{L}}] = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}, \tag{4.47}$$

respectively. Note that in contrast to the stiffness matrix the mass matrices do not depend on the orientation of the bar.

4.3.2 Modal analysis

The modes of vibration and the frequencies at which they vibrate, called the natural frequencies, of a structure are those motions a structure can make without being excited by an external force. In modal analysis the modes of vibration and the natural frequencies are derived.

By assuming all nodes move sinusoidally and in phase without there being an external force the eigenfrequencies and modes of vibration can be calculated. Consider the FEM equations, Eq. (4.41). We assume

$$\{u\} = \{u_0\}\cos\omega t \quad \text{and} \quad \{F\} = 0.$$

Substituting these in the FEM equations we obtain

$$\left(-\omega^2[M] + [K]\right)\{u_0\}\cos\omega t = 0.$$

Since we are looking for the solution where this is true independent of time we divide left and right by $\cos\omega t$ and obtain

$$([K] - \lambda[M])\{u_0\} = 0, \tag{4.48}$$

where $\lambda = \omega^2$. This is a generalized eigenvalue problem and there are many methods for solving it.

In MATLAB the easiest way is to use the function `eig`. Given matrices `A` and `B` the command

```
[V, D] = eig(A, B);
```

returns a matrix `V` whose columns are eigenvectors and a matrix `D` whose diagonal elements contain the eigenfrequencies.

In practice the generalized eigenvalue problem is solved with iterative methods such as inverse iteration, Jacobi iteration and the Lanczos algorithm. An overview can be found on Wikipedia by searching for ‘List of numerical analysis topics’.

4.3.3 Transient analysis; the Newmark-β method

In a transient analysis the response of a system to an arbitrary excitation is computed. For the finite element method this is often achieved by integrating a system of linear differential equations in time. A numerical integration scheme widely used for numerically integrating finite element equations is the Newmark-β method.

Only the case of a linear FE problem is considered, but the method can be applied to material or geometrical nonlinearity.

In order to solve the problem we need to know the initial conditions $\{u\}(t_0) = \{u\}_0$ and $\{\dot{u}\}(t_0) = \{\dot{u}\}_0$ and the boundary conditions describing the value of the displacements and velocities of the structure that are on the boundary for all time steps.

Let n be the current time step. The finite element equations must hold;

$$[M]\{\ddot{u}\}_n + [K]\{u\}_n = \{F\}_n, \tag{4.49}$$

where $\{\ddot{u}\}_n$, $\{u_n\}$ and $\{F\}_n$ are the nodal accelerations, nodal positions and nodal forces at time step n, respectively.

To compute the solution at the next time step $n+1$ the Newmark-β method is used.

With this method the nodal velocities and nodal positions at the next time step, $\{\dot{u}\}_{n+1}$ and $\{u\}$, respectively, are computed with

$$\{\dot{u}\}_{n+1} = \{\dot{u}\}_n + \Delta t\{\ddot{u}\}_\gamma, \tag{4.50}$$

$$\{u\}_{n+1} = \{u\}_n + \Delta t\{\dot{u}\}_n + \frac{1}{2}\Delta t^2\{\ddot{u}\}_\beta, \tag{4.51}$$

where

$$\{\ddot{u}\}_\gamma = (1-\gamma)\{\ddot{u}\}_n + \gamma\{\ddot{u}\}_{n+1}, \quad 0 \leq \gamma \leq 1$$

and

$$\{\ddot{u}\}_\beta = (1-2\beta)\{\ddot{u}\}_n + 2\beta\{\ddot{u}\}_{n+1}, \quad 0 \leq \beta \leq 1$$

where γ and β are two parameters for which different choices lead to integration schemes having different properties. This is discussed in more detail shortly.

These equations can be evaluated after determining the nodal acceleration at the next time step, $\{\ddot{u}\}_{n+1}$. They follow from substituting the expression for $\{u\}_{n+1}$ in the finite element equations at the $(n+1)$th time step. Doing so yields

$$[M]\{\ddot{u}\}_{n+1} + [K]\left(\{u_n\} + \Delta t\{\dot{u}\}_n + \frac{1}{2}\Delta t^2\left((1-2\beta)\{\ddot{u}\}_n + 2\beta\{\ddot{u}\}_{n+1}\right)\right) = \{F\}_{n+1}.$$

Collecting the terms with $\{\ddot{u}\}_{n+1}$ on the left hand side and moving the rest to the right yields

$$[A]\{\ddot{u}\}_{n+1} = \{F\}_{n+1} + [K]\{\hat{u}\}_n,$$

where

$$[A] = [M] + \Delta t^2 \beta [K] \quad \text{and} \quad \{\hat{u}\}_n = \{u\}_n + \Delta t \{\dot{u}\}_n + \frac{1}{2}\Delta t^2 (1 - 2\beta)\{\ddot{u}\}_n.$$

The nodal accelerations result from premultiplying with the inverse of $[A]$,

$$\{\ddot{u}\}_{n+1} = [A]^{-1}\left(\{F\}_{n+1} + [K]\{\hat{u}\}_n\right). \tag{4.52}$$

To obtain a solution one must choose the value of the parameters γ and β. There are several choices with interesting properties. Choosing $\gamma = 1/2$ and $\beta = 1/4$ results in the so-called 'constant average acceleration method'. The advantage of this method is that the algorithm is unconditionally stable, i.e. the time step can be chosen based on accuracy considerations only. Choosing $\gamma = 1/2$ and $\beta = 1/6$ results in the so-called 'linear acceleration method'. In contrast to the constant average acceleration method this method is conditionally stable. Choosing $\gamma = 1/2$ and $\beta = 0$ results in the so-called 'central difference scheme'. The advantage is that $[A] = [M]$, which makes the inversion required for computing the nodal accelerations $\{\ddot{u}\}_{n+1}$ very cheap in case of a lumped mass matrix. A disadvantage is that as the linear acceleration method this method is conditionally stable.

4.4 Finite Element Method for Nonlinear Systems

4.4.1 A primer on nonlinearities

A nonlinear system is one where the solution of the system does not linearly depend on the applied load. As such a nonlinear system does not obey the principle of superposition[7]. Output of a nonlinear system is not directly proportional to its input. Nonlinear systems are of paramount interest in science and engineering as most naturally occuring physical phenomena are inherently nonlinear. As an example, an equation of the form

$$y = ax$$

is said to be linear while

$$y = ax^2$$

represents a nonlinear equation.

[7]A linear system obeys the principle of superposition which states that if U_1 and U_2 are solutions to a linear system (PDE) then for any a and b, $aU_1 + bU_2$ is also a solution of the system

In structural analysis, using FEM, the nonlinearity can manifest itself in two ways, viz. *geometric* and *material* nonlinearity. Geometrically nonlinear problems are those involving large deformation (large strains), e.g. a fishing rod, which deforms under the applied load to the extent that it changes the original configuration (small strain theories ignore change in configuration!). On the other hand, material nonlinearity refers to the case where the contitutive relations are not linear. E.g. a nonlinear stress-strain relationship in nonlinear elastic, elasto-plastic and visco-plastic materials. Thus, an important departure in nonlinear FEM from linear FEM is that the underlying global stiffness matrix $[K]$ constantly changes with the solution ($\{u\}$) itself. A nonlinear system therefore is solved in an incremental (iterative) fashion by taking small (linear) steps. We accomplish this by using iterative schemes like the Newton-Raphson method to solve the resulting system of nonlinear equations. One must remember that while a well defined linear system has only one solution, a nonlinear system can potentially have multiple solutions (Shocks / bifurcation / chaos are all a result of nonlinearity in the system) making it harder (and more interesting!) to solve.

4.4.2 Conjugate Gradient Method

The system of linear equations

$$K\{u\} = f \tag{4.53}$$

can be solved directly by matrix inversion or Gaussian elimination. For very large systems, however, most direct approaches are computationally complex and have a very large memory footprint. To solve the system of equations efficiently (and approximately) one therefore resorts to iterative solvers. There are a large number of iterative solvers for solving linear systems of equations. They are collectively categorized as *Krylov space solvers* due to certain characteristics of the solutionspace generated by these solvers. An example of such a solver is the *Conjugate Gradient* (CG) method which is based on decomposing the solution of the system into a conjugate orthonormal basis $\mathbf{p_i}$. This basis is generated using the *Gram-Schmidt orthogonalization process.* The CG method is only applicable to symmetric matrices, but other iterative solvers are available for unsymmetric sytems. The CG algorithm can be written as in Algorithm 5.

In general the CG method will converge in at most n iterations, however, the convergence of the method will depend on the condition number of the matrix and somewhat on your initial guess.

4.4.3 Newton-Raphson Method

Newton-Raphson (NR) or simply Newton's method is a numerical method for solving nonlinear equations written in the form of $F(u) = 0$, where F is

Algorithm 5 The CG algorithm

```
$\mathbf{r}_0 := \mathbf{f} - \mathbf{K}\mathbf{u}_0$
$\mathbf{p}_0 := \mathbf{r}_0$
$i := 0$
$DONE := 0$
while $DONE \neq 1$ do
  $\alpha_i := \frac{\mathbf{r}_i^{\mathrm{T}}\mathbf{r}_i}{\mathbf{p}_i^{\mathrm{T}}\mathbf{K}\mathbf{p}_i}$
  $\mathbf{u}_{i+1} := \mathbf{u}_i + \alpha_i \mathbf{p}_i$
  $\mathbf{r}_{i+1} := \mathbf{r}_i - \alpha_i \mathbf{K}\mathbf{p}_i$
  if $r_{i+1} \leq Tol$ then
    $DONE := 1$
  end if
  $\beta_i := \frac{\mathbf{r}_{i+1}^{\mathrm{T}}\mathbf{r}_{i+1}}{\mathbf{r}_i^{\mathrm{T}}\mathbf{r}_i}$
  $\mathbf{p}_{i+1} := \mathbf{r}_{i+1} + \beta_i \mathbf{p}_i$
  $i := i + 1$
end while
```

an arbitrary function [8] (linear or nonlinear) of the variable u. Thus, NR is a method to find the zeros (or root) of a nonlinear function F. NR starts the solution process from an initial guess u_0 (i.e. $F(u_0) \approx 0$) and successively refines the guess based on the Jacobian (derivative w.r.t. solution) of the function $\frac{\partial F}{\partial u}$.

Suppose we have a current guess u_n, and let u_{n+1} be the next guess such that $F(u_{n+1}) = 0$ then by the definition of derivative we can write,

$$F'(u_n) = \frac{F(u_n) - 0}{u_n - u_{n+1}} \tag{4.54}$$

which implies

$$u_{n+1} = u_n - \frac{F(u_n)}{F'(u)} \tag{4.55}$$

As n goes larger and larger the $|u_{n+1} - u_n| \to 0$ and the scheme is said to converge to the correct solution.

Remark: As mentioned earlier the convergence of NR scheme will strongly depend on your initial guess, however for physical systems which have a unique solution a NR scheme uniformly converge with any starting guess.

4.4.4 Nonlinear bar

In the previous section you saw the finite element formulation of a linear bar element. Let us now consider a bar with material nonlinearity. Recall

[8] In case of FEM one can think of $F(u) = K(u)\{u\} - \{f\}$, which now defines a vector function in terms of FEM unknowns. In this case solving $F(u) = 0 \implies K(u)\{u\} = f$

that the bar under the stretching load can be modeled by the differential equation

$$E\frac{d^2u}{dx^2} = f(x) \tag{4.56}$$

Where E is the Young's modulus of the bar and is generally taken to be a constant. For nonlinear materials E is not a constant and the differential equation in a more general form is written as

$$\frac{d}{dx}\left(E(u)\frac{du}{dx}\right) = f(x), \tag{4.57}$$

where the material stiffness $E(u)$ now depends on the 'stretch' u and renders this equation nonlinear. A possible form of $E(u)$ can be written as

$$E(u) = E_0\left(1 - \frac{du}{dx}\right) \tag{4.58}$$

As before, to obtain the finite element formulation (weak form) we multiply 4.57 by a test function v and perform integration by parts over the length of the bar Ω. We obtain

$$\int\limits_{\Omega} E_0\left(1 - \frac{du}{dx}\right)\frac{du}{dx}\frac{dv}{dx}dx = \int\limits_{\Omega} f(x)vdx,$$

which can be written as

$$\underbrace{\int\limits_{\Omega} E_0\frac{du}{dx}\frac{dv}{dx}dx}_{\text{Linear term}} - \underbrace{\int\limits_{\Omega} E_0\frac{du}{dx}\frac{du}{dx}\frac{dv}{dx}dx}_{\text{Nonlinear term}} = \underbrace{\int\limits_{\Omega} f(x)\ v\ dx}_{\text{RHS vector}} \tag{4.59}$$

4.4.5 Newton's method for nonlinear finite elements

We now know that the stiffness matrix of the nonlinear finite element method depends on the instantaneous solution u and can be written as

$$K(u)u = f. \tag{4.60}$$

In order to write it in the form $f(x) = 0$, we define the residual

$$r(u) := K(u)u - f \qquad \Longrightarrow \qquad r(u) = 0. \tag{4.61}$$

Then, the Newton iteration formula becomes

$$\begin{aligned} u_{r+1} &= u_r - \frac{r(u_r)}{\frac{dr(u_r)}{du}} \\ &= u_r - T^{-1}(u_r)\ r(u_r) \end{aligned} \tag{4.62}$$

The tangent stiffness matrix (a.k.a. Jacobian matrix), matrix T is

$$\begin{aligned} T(u_r) := \frac{dr(u_r)}{du} &= \frac{d}{du}[K(u)u - f] \\ &= K(u) + \frac{dK(u)}{du}u \\ &= K(u_r) + \frac{dK(u_r)}{du}u_r \end{aligned}$$

Therefore the "tangent stiffness" is,

$$T(u_r) := K(u_r) + \frac{dK(u_r)}{du}u_r. \tag{4.63}$$

Since $\{u_r\}$ is a vector (of FEM coefficients) and $[K]$ is a matrix, we can write it more appropriately as

$$\mathbf{T} := \mathbf{K}(\mathbf{u_r}) + \frac{d\mathbf{K}(\mathbf{u_r})}{d\mathbf{u_r}}\mathbf{u_r},$$

or

$$T_{ij} := K_{ij} + \frac{dK_{ik}}{du_j}u_k \tag{4.64}$$

Computing (4.64) is not always trivial and depends on the weak form, often if the variation of K with u is smooth we approximate $T_{ij} \approx K_{ij}$, which simplifies the computation. Also note that taking the inverse of T in Eq. (4.62) requires solving a linear system of equations. Thus a nonlinear solver consists of a bunch of iterative linear solvers (using CG or other linear solvers).

Chapter 5

Random numbers

Some natural phenomena can be accurately approximated as stochastic processes, meaning that they have a random element. Outcomes can only be determined as having a certain probability.

This chapter discusses two topics that are very important in simulating stochastic processes. Section 5.1 discusses random number generators, which take care of providing a sequence of random numbers to simulate with. Section 5.2 discusses random walks, which give accurate descriptions of phenomena such as Brownian motion, molecular chaos and diffusion.

5.1 Random number generators

To simulate a natural phenomenon as a stochastic process, a source of randomness is required. Such sources are called 'random number generators' (RNGs). Random number generators can be divided in generators of 'true' randomness and generators of pseudo-randomness.

True randomness is obtained from measurements of natural phenomena, such as radioactive decay [1], thermal noise or radio noise [2]. While generators of true randomness can provide the highest quality of randomness, for many applications it is more desirable to have a source that is capable of generating a sequence of random numbers repeatedly.

Pseudo-random number generators (PRNGs) are algorithms that generate a sequence of numbers that approximates the properties of random numbers. Next, two such algorithms are highlighted. The first subsection, Section 5.1.1, discusses the linear congruential generator. The subsection following, Section 5.1.2, discusses the lagged Fibonacci generator. The third subsection, Section 5.1.3, mentions a few other generators that in general produce higher quality random numbers. The last subsection, Section 5.1.4, discusses ways of testing the randomness of a sequence of numbers.

[1]See http://www.fourmilab.ch/hotbits/.
[2]See http://www.random.org/

5.1.1 The linear congruential generator

A linear congruential generator (LCG) is one of the oldest and well-known PRNGs. The advantages are that it's simple and computationally cheap.

The generated numbers satisfy

$$X_{n+1} = (aX_n + c) \mod m, \tag{5.1}$$

where X_n is the nth generated number. To obtain a sequence of numbers, four integer constants need to be chosen. They are m, the 'modulus' satisfying $m > 0$, a, the 'multiplier' satisfying $0 < a < m$, c, the 'increment' satisfying $0 \leq c < m$, and X_0, the 'seed' or 'start value' satifying $0 \leq X_0 < m$.

Choosing the same seed value will produce the same sequence of numbers, allowing one to repeat a simulation exactly, which is helpful in e.g. debugging.

The random numbers are periodic; they repeat in the sequence after at most m numbers. To obtain a sequence of random numbers with the maximum period the following conditions for c, m and a must be met.

- c and m must be relatively prime, meaning their only common divisor is 1.
- $a - 1$ is divisable by all prime factors of m. [3]
- $a - 1$ is a multiple of 4 if m is a multiple of 4.

The quality of the generated randomness is extremely sensitive to the choice of these parameters. A common choice for parameters is $m = 2^{32}$, $a = 1103515245$ and $c = 12345$, where only bits 30 to 16 are used as representing random numbers [4].

5.1.2 The lagged Fibonacci generator

The lagged Fibonacci generator (LFG) satisfies

$$S_n = S_{n-j} \star S_{n-k} \mod m, \quad 0 < j < k, \tag{5.2}$$

where S_n is the nth generated number and $\star$ is either the addition, subtraction, multiplication or bitwise arithmetic exclusive-or operator (XOR). The modulus m is usually a power of 2.

The maximum period is $(2^k - 1)2^{m-1}$ for addition and subtraction and $(2^k - 1)k$ for the XOR operator. The maximum period for multiplication is $(2^k - 1)2^{m-3}$, i.e. a quarter of that for addition and subtraction.

[3] The prime factors of a number n are the prime numbers that, when multiplied, result in n. Each integer has a unique set of prime factors.

[4] For more choices of parameters, see
http://en.wikipedia.org/wiki/Linear_congruential_generator#Parameters_in_common_use

Achieving this maximum period requires that j and k are chosen such that $1 + x^j + x^k$ is primitive for all integers mod 2.

To start generating random numbers using an LFG requires a random sequence. Such a seed can be obtained from an LCG.

The quality of the output is again very sensitive to the choice of constants. An example is $(24, 55)$ [5].

5.1.3 Other random number generators

Other random number generators producing better quality random numbers are the Mersenne twister [6] and the multiply-with-carry [7] generators.

5.1.4 Testing randomness

There are many ways of testing the randomness of a sequence of numbers. To mention a few:

- *Squaretest*; the points (X_n, X_{n+1}) are plotted for all n. The more homogeneous the resulting plot is, the higher quality the randomness.
- *Cubetest*; the points (X_n, X_{n+1}, X_{n+2}) are plotted for all n. The more homogeneous the resulting plot is, the higher the quality of randomness.
- Computing the *mean value*. For a large enough sequence of random numbers the mean value, given by

$$\bar{X} = \frac{1}{N} \sum_{n=1}^{N} X_n,$$

  for a sequence of N numbers, should approximate the arithmetic mean of the full sequence of random numbers the generator is capable of generating.
- *Spectral test*; the result of the Fast Fourier Transform of the sequence of numbers should resemble that of white noise.
- *Chi-square test*. More specifically, Pearson's chi-square test, tests the hypothesis that sample data is distributed according to some theoretical distribution. In the case of the RNGs of Section 5.1.1 to 5.1.3 the hypothesized theoretical distribution is the uniform distribution. In this case, and assuming a sequence containing N generated random numbers, the test is performed according to the following steps.

[5]More pairs can be found on http://en.wikipedia.org/wiki/Lagged_fibonacci_generator#Properties_of_lagged_Fibonacci_generators

[6]See http://en.wikipedia.org/wiki/Mersenne_twister.

[7]See http://en.wikipedia.org/wiki/Multiply-with-carry.

1. Divide the range of the generator in n bins (typically, $n = 10$), such that at least 5 numbers are expected to fall into each bin.
2. Compute the expected frequencies for the bins, E_i, $i = 1..n$ which for the uniform distribution is given by N times the ratio of the range of the bin and the total range.
3. Compute the test-statistic for the chi-square test,

$$X^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i},$$

   where O_i is the observed frequency. The test-statistic has the property that it grows as the observed frequencies O_i differ more from the expected frequencies E_i, as well as following a chi-square distribution if the assumed theoretical distribution is correct.
4. Compute the probability that the test statistic takes on the computed value or higher assuming the theoretical distribution is the true distribution, by evaluating 1 minus the cumulative distribution function of the chi-square distribution [8] for the value of the test statistic.
5. The test fails if the found probability is lower than the chosen criterium of statistical significance (typically 0.05).

Tip: use the MATLAB function `chi2gof`.

- *Correlation functions*; correlation functions are functions that result in a measure of dependency between sequences of numbers. They are suited for testing random numbers, because different sequences of random numbers shouldn't correlate. One way of using correlation to check the randomness of a sequence of numbers is by computing the autocorrelation for a sequence of random numbers, given by

$$R_{xx}(j) = \sum_{n} x_n x_{n-j},$$

  where x is the sequence of numbers and x_n is the nth number from this sequence. For a sequence of truly random numbers the autocorrelation function becomes infinite at $j = 0$ and zero elsewhere, as the length of the sequence tends to infinity.

5.2 Random walks

A random walk (RW) is the mathematical description of a trajectory consisting of steps that are each in a random direction. The random walk turns out

[8] See
http://en.wikipedia.org/wiki/Chi-square_distribution#Cumulative_distribution_function

to model accurately the natural phenomena of Brownian motion, molecular chaos and diffusion.

In the following sections the 1D random walk is discussed (Section 5.2.1), there are some remarks on random walks in 3D (Section 5.2.2), as well as some remarks on continuous time random walks (Section 5.2.3), the probability distribution for the position of a particle in a random walk is discussed (Section 5.2.4) and the diffusion equation is derived from the random walk (Section 5.2.5).

5.2.1 The one dimensional random walk

Consider a particle that can move in one dimension, x. For each timestep Δt at time t, it has a probability of 0.5 of moving a step of Δx to the left, and a probability of 0.5 of moving a step of Δx to the right, or

$$x_{t+1} = x_t + \begin{cases} \Delta x & \text{with a probability of 0.5,} \\ -\Delta x & \text{with a probability of 0.5.} \end{cases} \tag{5.3}$$

For the rest of this section we will take the size of the timestep Δx to be equal to 1.

Now assume that one has simulated the N trajectories of N such particles.

The left plot in Fig. 5.1 shows two such trajectories. The average position of the particles at time t, $\langle x_t \rangle$, is given by

$$\langle x_t \rangle = \frac{1}{N} \sum_{i=1}^{N} x_t^{(i)},$$

where N is the number of trajectories considered and $x_t^{(i)}$ is the position on the ith trajctory at time t. As the number of trjactories N tends to infinity, the average position of the particles at time t, $\langle x_t \rangle$, tends to zero, or

$$\lim_{N\to\infty} \langle x_t \rangle = 0.$$

The right plot in Fig. 5.1 shows the average squared distance over all N trajectories as a function of time, or

$$\langle x_t^2 \rangle = \frac{1}{N} \sum_{i=1}^{N} \left(x_t^{(i)} \right)^2 .$$

From the figure it is clear that in the limit of the number of trajectories N being infinite, the average squared distance grows linearly with time, or

$$\lim_{N\to\infty} \langle x_t^2 \rangle = Dt,$$

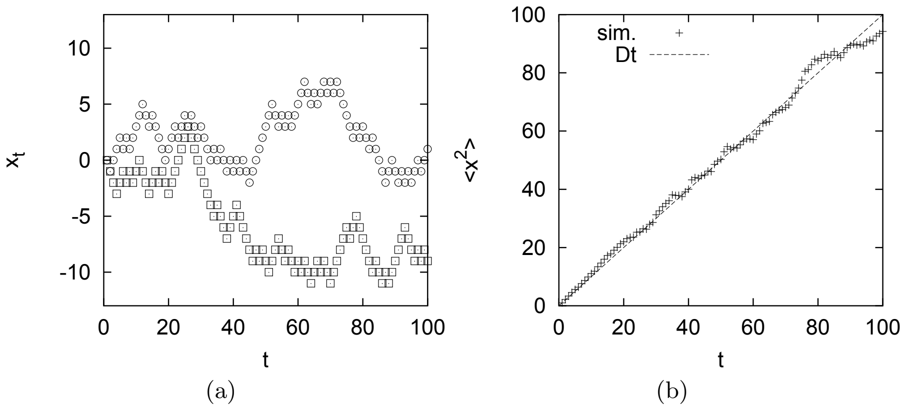

Figure 5.1: Position x_t for two different random walks (left) and the squared distance averaged over all N trajectories, $\langle x^2 \rangle$

where D is the constant of diffusion.

The average squared distance of the particles being linear with time can be derived as follows. Let γ_i be the ith step of a particle in a random walk, so $\gamma_i \in \{-1, 1\}$. The position of a particle whose steps are γ_i at time t, x_t, is given by

$$x_t = \sum_{i=1}^{t} \gamma_i.$$

The squared distance of this particle is therefore

$$x_t^2 = \left(\sum_{i=1}^{t} \gamma_i \right)^2 = \sum_{i=1}^{t} \sum_{j=1}^{t} \gamma_i \gamma_j.$$

We split this sum in the addition of a sum for which $i = j$ and one for which $i \neq j$, which we denote by $\sum_{i,j=1..t,i=j}$ and $\sum_{i,j=1..t,i\neq j}$, respectively. So the squared distance of the particle, x_t^2, becomes

$$x_t^2 = \sum_{i,j=1..t,i=j} \gamma_i \gamma_j + \sum_{i,j=1..t,i\neq j} \gamma_i \gamma_j.$$

The first part of this sum is simply the sum of squares of all timesteps, $\sum_{i=1}^{t} \gamma_i^2 = t$, whereas the second part is the sum of all products of two timesteps at different times. Since the probability of a timestep being positive or negative is 0.5, so is the probability of the product of two timesteps, $\gamma_i \gamma_j$ at different times. The second part is therefore a random walk.

Now consider the average squared distance of N particles at time t, $\langle x_t^2 \rangle$. It is given by the average over all particles of the first part, which is simply t, in addition to the average over all particles of the second part, which is the average of N random walks and therefore zero.

It can be easily verified that in the more general case where $\gamma_i \in \{-a, a\}$ the average squared distance is given by $\langle x_t^2 \rangle = a^2 t$, which is proportional with time.

5.2.2 Random walks in 3D

The random walk of a particle moving in three dimensions can be considered as three independent random walks in one dimension. The squared distance is given by

$$r^2 = x^2 + y^2 + z^2.$$

As in the one dimensional case, the distance is proportional with the square root of time.

5.2.3 Continuous time random walks

Now consider a random walk where the step size varies continuously between $-a$ and a. The left plot of Fig. 5.2 shows two trajectories for $a = 2$, the right plot shows the averaged squared distance as a function of time for $a = 1$ and $a = 2$. It can be seen that the difference is merely the constant of diffusion. The distance is still proportional with the square root of time.

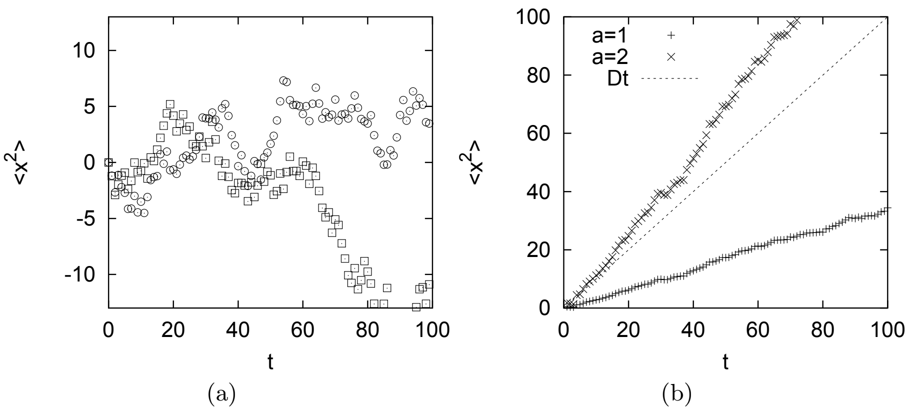

Figure 5.2: Results of simulating random walks with continuously varying step size

5.2.4 The probability distribution for the position of a particle in a random walk

Consider now the simulation of a lot of random walk trajectories and let $N_t(x)$ be the number of particles that end up at the position x at time t. It turns out that the shape of $N_t(x)$ is that of a normal distribution.

The equation describing $N_t(x)$ divided by the total number of trajectories considerd, N, written $n_t(x)$, is given by the normal distribution

$$n_t(x) = \frac{N_t(x)}{N} = \frac{1}{\sqrt{\pi}\sigma} e^{-\frac{(x-\mu)^2}{\sigma^2}}, \tag{5.4}$$

with an expected value of $\mu = 0$ and a variance of $\sigma^2 = 2Dt = 2\langle x_t^2 \rangle$.

The standard deviation σ is a linear function of time. $n_t(x)$ is therefore a bell curve that starts out as a peak of height 1 and, as time progresses, 'flattens' as the variance grows linearly, see Fig. 5.3.

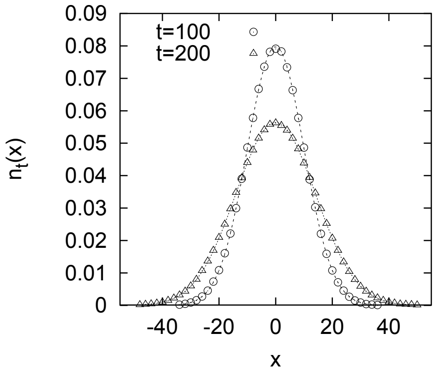

Figure 5.3: The probability of finding a particle at position x for different times t

5.2.5 The random walk and diffusion

Fick's second law of diffusion, analogous to the heat equation, can be derived from a random walk by considering the master equation. The master equation describes the change in the probability of finding a particle at position x from time t to time $t + \Delta t$, given the probabilities at time t of a particle jumping to a different x. For the one dimensional random walk considered in Section 5.2.1, it is given by

$$\Delta p(x, t+\Delta t) = p(x+\Delta x, t)W_- + p(x-\Delta x, t)W_+ - p(x, t)(W_- + W_+), \tag{5.5}$$

where $p(x, t)$ is the probability of finding a particle at position x, W_- is the probability of a particle jumping to the left and W_+ is the probability of a particle jumping to the right. The first term represents the probability at time t of a particle jumping from position $x + \Delta x$ to position x, the second term represents the probability at time t of a particle jumping from position $x - \Delta x$ to position x, and the third term represents the probability at time t of a particle at position x jumping to position $x + \Delta x$ or $x - \Delta x$. In the the case of the random walk considered in the previous paragraphs $W_- = W_+ = 0.5$.

By taking the limits $\Delta t \to 0$ and $\Delta x \to 0$ we obtain

$$\frac{\partial p}{\partial t} = \Delta p(x, t + \Delta t)/\Delta t,$$

and

$$\frac{\partial^2 p}{\partial x^2} = [p(x + \Delta x, t) - 2p(x + \Delta x, t) + p(x - \Delta x, t)]/\Delta x^2.$$

which can be combined to give

$$\frac{\partial p}{\partial t} = \mathcal{D}\frac{\partial^2 p}{\partial x^2},$$

where the constant of diffusion is given by

$$\mathcal{D} = \frac{\Delta x^2}{2\Delta t}.$$

The solution to this equation is

$$p(x, t) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

where

$$\sigma = \sqrt{2\mathcal{D}t}.$$

The difference with Eq. 5.4 is in the definition of the constants of diffusion. They satisfy $D = 2\mathcal{D}$.

Chapter 6

Smoothed Particle Hydrodynamics

6.1 Introduction

Smoothed-particle hydrodynamics (SPH) is a numerical method that approximates the continuous description of fluid motion, fluid dynamics, using particles.

Like with the finite element method, see Section 4, the state of the fluid modeled at points between the particles are interpolated from the values at interpolation points. In contrast to the finite element method, the interpolation points are not in a static configuration, but instead move as part of the fluid modeled.

In grid based methods such as the finite difference method, derivatives occuring in the partial differential equations describing the problem are approximated by an expression containing the value of the quantity at neighbouring grid points. In contrast to such methods SPH computes derivatives by differentiating a continuous function resulting from interpolating the values at the particles.

Like molecular dynamics, see Section 3, the simulation is performed by determining the motions as resulting from the interactions with neighbouring particles. In contrast to molecular dynamics the particles do not represent physical objects.

The parts of the method this chapter describes follows largely the way they were described by Monaghan in [2] and [3] [1]. First, the way the quantities are interpolated from their values at the particles is discussed in Section 6.2. Next, a set of equations determining the motion of for the case of inviscid flow is given in Section 6.3. Finally, a way of modeling viscosity using SPH is outlined in Section 6.4.

[1]The papers can be found at http://adsabs.harvard.edu/full/1992ARA%26A..30..543M and http://iopscience.iop.org/0034-4885/68/8/R01, respectively.

6.2 Interpolation

6.2.1 Integral and summation interpolants

In SPH, the interpolation of a quantity A, $A(\mathbf{r})$, is based on the integral interpolant

$$A_{\mathrm{I}}(\mathbf{r}) = \int A(\mathbf{r}')W(\mathbf{r}-\mathbf{r}')d\mathbf{r}', \tag{6.1}$$

where the function W is called the 'kernel' and $d\mathbf{r}$ is a differential volume element. The parameter h controls the range which the kernel affects; as h gets larger the kernel covers a larger part of the domain. When W is the Dirac delta function A is reproduced exactly. The kernels used in practice tend to the delta function as h tends to zero. The kernels are normalized to 1, meaning that integrating them over their domain results in 1. The Gaussian kernel, given by

$$W(x,h) = \frac{1}{h\sqrt{\pi}}e^{-x^2/h^2},$$

was used originally.

Equation 6.1 can be rewritten to

$$\int \frac{A(\mathbf{r}')}{\rho(\mathbf{r}')}W(\mathbf{r}-\mathbf{r}',h)\rho(\mathbf{r}')d\mathbf{r}',$$

where $\rho(\mathbf{r})d\mathbf{r}'$ is a differential element of mass. This integral can be approximated by the summation interpolant

$$A_{\mathrm{S}}(\mathbf{r}) = \sum_b m_b \frac{A_b}{\rho_b} W(\mathbf{r}-\mathbf{r}',h), \tag{6.2}$$

where the summation is over all particles, but in practice, due to the kernel being zero outside the neighbourhood of a particle, only covers the nearest particles.

The disadvantage of using a Gaussian kernel is that even particles that are an infinite distance away contribute to a quantity at a point. A kernel that does not have this disadvantage is a spline function. For the 1D case it is given by

$$W(\mathbf{r},h) = \frac{2}{3h}\begin{cases} 1-\frac{3}{2}q^2+\frac{3}{4}q^3 & \text{if } 0 \le \frac{r}{h} \le 1, \\ \frac{1}{4}(2-q)^3 & \text{if } 1 \le \frac{r}{h} \le 2, \\ 0 & \text{otherwise,} \end{cases} \tag{6.3}$$

The kernel has compact support, meaning that it quickly tends to zero, the second derivative is continuous and the dominant error term in the integral interpolant is $O(h^2)$.

6.2.2 First derivatives

Derivatives can be easily calculated in SPH. Given a kernel W that is differentiable, the derivative of a quantity A is approximated by

$$\frac{\partial A_{\mathrm{s}}}{\partial x} = \sum_b m_b \frac{A_b}{\rho_b} \frac{\partial W}{\partial x}.$$

In SPH, the derivative is thus found as an exact derivative of an approximate function. The disadvantage of this form is that it doesn't vanish if A is constant. This is resolved by using the fact that

$$\frac{\partial A}{\partial x} = \frac{1}{\Phi}\left(\frac{\partial(\Phi A)}{\partial x} - A\frac{\partial \Phi}{\partial x}\right).$$

where Φ is any differentiable function. The SPH form is

$$\frac{\partial A}{\partial x} = \frac{1}{\Phi}\sum_b m_b \frac{\Phi_b}{\rho_b}(A_b - A_a)\frac{\partial W_{ab}}{\partial x_a}.$$

Choosing $\Phi = 1$ the derivative can be expressed as

$$\frac{\partial A_a}{\partial x_a} = \sum_b \frac{m_b}{\rho_b} A_{ba} \frac{\partial W_{ab}}{\partial x_a},, \tag{6.4}$$

where $A_{ba} = A_b - A_a$ and $W_{ab} = W(\mathbf{r} - \mathbf{r}', h)$.

The 2005 paper [3] also discusses second derivatives in one, two and three dimensions.

6.3 Simple Equations of Motion

An inviscid fluid is described by equations expressing the conservation of momentum, mass and energy. Furthermore, an equation of state is needed.

Next, in Section 6.3.1, SPH equations of motion derived from the conservation laws are given. In Section 6.3.2 the integration of the equations of motion is discussed.

6.3.1 SPH equations of motion satisfying the conservation laws

The equation expressing the change in particle velocity, $\mathbf{v}_a$ satisfying the conservation of linear and angular momentum is given by

$$\frac{d\mathbf{v}_a}{dt} = -\sum_b m_b \left(\frac{P_b}{\rho_b^2} + \frac{P_a}{\rho_a^2}\right)\nabla_a W_{ab}, \tag{6.5}$$

where P_a is the pressure of particle a and ∇_a is the gradient at the position of particle a.

To approximate the conservation of mass, one of two equations is typically used. They are

$$\rho_a = \sum_b m_b W_{ab} \tag{6.6}$$

and

$$\frac{d\rho_a}{dt} = \sum_b m_b \mathbf{v}_{ab} \nabla_a W_{ab}, \tag{6.7}$$

where $\mathbf{v}_{ab} = \mathbf{v}_a - \mathbf{v}_b$.

The second form has some advantages over the first form. See section 3.2 in [2].

An equation approximating the conservation of thermal energy is given by

$$\frac{du_a}{dt} = \frac{P_a}{\rho_a^2} \sum_b m_b \mathbf{v}_{ab} \cdot \nabla_a W_{ab} \tag{6.8}$$

The equation of state can be anything that applies to the problem under consideration.

6.3.2 Integration of the equations of motion

Using numerical integration, the equations from Section 6.3.1 can be integrated to simulate an inviscid fluid.

One of the advantages of SPH is that it it is easy to implement variable spatial and temporal resolution, greatly improving the efficiency of simulations. This means that time integration schemes requiring the time step to remain constant, such as Verlet, cannot be used. Instead, schemes such as Euler–Cromer (Section 2.3.2) or the midpoint method (Section 2.3.4) can be used.

6.4 Viscosity

Artifical viscosity is a way to smooth out regions approximating discontinuities to make them smooth enough to compute with relative ease.

Chapter 7

Finite Volume Method

The Finite Volume Method is a method for integrating partial differential equations over a domain.

Like the finite element method (Chapter 4) and in contrast to smoothed particle hydrodynamics (Chapter 6), the domain is discretized by representing it as a meshed grid.

The finite volume method is called that way because each point on the grid can be thought of as being surrounded by a small volume. The solution at each point is found by approximating the flux through the surfaces of the volume surrounding it. This way of discretization gives the finite volume method the important property of being conservative.

In Section 7.1 it is shown how to convert a problem stated in the form of a partial differential equation to an equivalent integral form appropriate for discretization. Section 7.2 discusses the *total variation diminishing* property, a desirable property of integration schemes preventing unrealistic oscillations in the solution. Section 7.3 discusses the CFL condition, a necessary but not sufficient condition for an integration scheme to be convergent. Section 7.4 discusses the Lax-Friedrichs method, a numerical method allowing the solution of the finite volume equations. Section 7.5 discusses flux limiters, mathematical terms that allow higher order schemes to be total variation diminishing. Section 7.6 discusses the total variation diminishing Lax-Friedrichs method, a higher order numerical method that is total variation diminishing. Finally, Section 7.7 discusses initial and boundary conditions for the finite volume method.

7.1 The FVM integral equation

In introducing the Finite Volume Method, we solve $\omega(x,t)$ for $0 \leq x \leq L$ and $0 \leq t \leq t_{\text{end}}$, where ω satisfies the partial differential equation

$$\frac{\partial \omega}{\partial t} + a(\omega)\frac{\partial \omega}{\partial x} = 0. \tag{7.1}$$

Here, $a(\omega)$ is the 'characteristic speed'. This equation can be rewritten to the form

$$\frac{\partial\omega}{\partial t} + \frac{\partial f(\omega)}{\partial x} = 0, \tag{7.2}$$

where f is related to a by

$$a(\omega) = \frac{\partial f(\omega)}{\partial\omega}. \tag{7.3}$$

The variable f may be interpreted physically as the flux of the variable ω in the x direction. The boundary conditions are given by $u(0,t) = u_0$ and $u(L,t) = u_L$, the initial condition is $u(x,0) = u^0$.

In this section Eq. 7.2 is transformed to an equivalent integral form, suitable for discretization.

We first divide the domain into 'cells'. Each cell covers a region $[x_{j-1/2}, x_{j+1/2}] \times [t^n, t^{n+1}]$ with area $\Delta x \Delta t$. The indexes j and n satisfy $j = 1..(L/\Delta x)$ and $n = 0..(t_{\text{end}}/\Delta t)$, respectively. The jth position and nth time are given by $x_j = (j-1/2)\Delta x$ and $t_n = n\Delta t$, respectively. See Fig. 7.1.

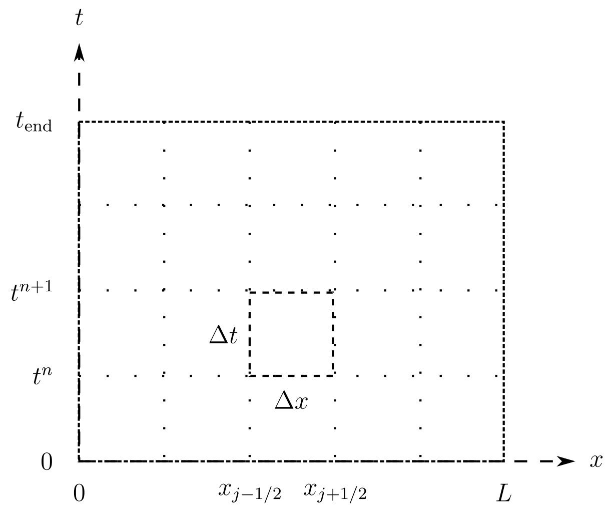

Figure 7.1: The domain of the problem from Section 7.1 partitioned into cells

From Eq. 7.2 we know that for each cell

$$\int_{x_{j-1/2}}^{x_{j+1/2}} \int_{t^n}^{t^{n+1}} \left\{ \frac{\partial\omega}{\partial t} + \frac{\partial f}{\partial x} \right\} dt\ dx = 0, \tag{7.4}$$

by integrating Eq. 7.2 over the cell region.

Green's theorem states that the relation between an integral over a region and an integral over the boundary of the region satisfies

$$\int\int_R \left(\frac{\partial Q}{\partial y} - \frac{\partial P}{\partial z}\right) \, dt \, dx = \int_C (Q \, dz + P \, dy), \tag{7.5}$$

where C is the boundary of the region R. By Green's theorem with the identifications $Q \to \omega$, $P \to -f$, $y \to t$ and $z \to x$ we can therefore write Eq. 7.4 as

$$\int_C (\omega \, dx - f \, dt) = 0, \tag{7.6}$$

where C is given by the line surrounding the small square region of width Δx and height Δt in Fig. 7.1.

From Fig. 7.1 it is clear that the integral on the left hand side satisfies

$$\begin{aligned}\int_C (\omega \, dx - f \, dt) = &\int_{x_{j-1/2}}^{x_{j+1/2}} \omega(x, t^n) \, dx - \int_{t^n}^{t^{n+1}} f(x_{j+1/2}, t) \, dt \\ &- \int_{x_{j-1/2}}^{x_{j+1/2}} \omega(x, t^{n+1}) \, dx + \int_{t^n}^{t^{n+1}} f(x_{j-1/2}, t) \, dt,\end{aligned} \tag{7.7}$$

where the integrals on the right hand side follow from integrating $\omega \, dx - f \, dt$ along the four lines making up the boundary C in a counter-clockwise direction.

Substituting Eq. 7.7 in Eq. 7.6, multiplying left and right with $1/\Delta x$ and rearranging terms we obtain

$$U_j^{n+1} = U_j^n - \nu \left[F(U; j + 1/2) - F(U; j - 1/2)\right], \tag{7.8}$$

where

$$U_j^n = \frac{1}{\Delta x} \int_{x_{j-1/2}}^{x_{j+1/2}} \omega(x, t^n) \, dx,$$

is the mean value of ω over the cell width $x_{j-1/2} \leq x < x_{j+1/2}$ at time t^n,

$$F(U; j + 1/2) = \frac{1}{\Delta t} \int_{t^n}^{t^{n+1}} f(x_{j+1/2}, t) \, dt,$$

is the mean value of f at the location $x_{j+1/2}$ over cell height $t^n \leq t < t^{n+1}$ and

$$\nu = \frac{\Delta t}{\Delta x}.$$

Since the necessary boundary and initial conditions are known, we can compute the solution to the problem by computing $F(U; j+1/2)$ for all j at each time step. Rather than computing the exact value, giving the exact solution, the Finite Volume Method works by computing an approximate flux $\mathcal{F}$ from the values of U at locations x_j.

Depending on whether we choose to compute the approximate flux from values at the current time step or those at the next time step the integration scheme is called explicit or implicit, respectively. The steps in explicit methods are cheap to compute but require a very small time step in order for the solution to converge, whereas the steps in implicit methods are more expensive to compute, but a larger time step is allowed. Depending on the problem, an explicit or implicit scheme may require the least amount of computation. In this chapter only explicit schemes are discussed.

7.2 Total variation diminishing schemes

Methods that approximate the flux terms with first order accuracy are known to result in smeared solutions, causing shocks to be lost. Second order methods, though more accurate, often produce unrealistic oscillations in the solution.

Second order accurate methods that are total variation diminishing (TVD) have the property that they produce no unrealistic oscillations while being second order accurate. A method is TVD if the total variation (TV) of the solution satisfies a certain condition.

The total variation (TV) of a numerical solution at a given time step t^n is given by

$$TV(U^n) = \sum_{j=0}^{N-1} \left| U_{j+1}^n - U_j^n \right|, \tag{7.9}$$

where N is the number of grid points.

When a solution shows oscillations the total variation will oscillate as well. A way of ensuring a solution has no oscillations is therefore to require a solution to satisfy that its TV monotonically decreases with time, or

$$TV(U^{n+1}) \leq TV(U^n). \tag{7.10}$$

Any method that results in solutions that all satisfy the condition of Eq. 7.10 is called a TVD method.

It has been shown that for the constant coefficients case, so when a given by Eq. 7.3 does not depend on ω, the general form

$$U_j^{n+1} = U_j^n - C_{j-1/2}(U_j^n - U_{j-1}^n) + D_{j+1/2}(U_{j+1}^n - U_j^n) \tag{7.11}$$

is a TVD method if it satisfies the conditions

$$0 \leq C_{j-1/2} \quad \forall j,$$

$$0 \le D_{j+1/2} \quad \forall j,$$

and

$$0 \le C_{j+1/2} + D_{j+1/2} \le 1 \quad \forall j.$$

7.3 The CFL condition

The Courant-Friedrichs-Lewy condition (CFL condition) is a necessary but not sufficient condition for any explicit finite volume or difference method to be stable. It states that for an explicit scheme to be stable, the rate at which information can travel across the gridpoints, which is $\Delta x/\Delta t$, must be larger or equal to the rate at which the physical information can travel, given by the speed at which a shock travels, a.

The CFL condition is often stated as a condition on the Courant number C, which is the dimensionless number resulting from dividing the speed at which physical information travels, a, with $\Delta x/\Delta t$, or

$$C = a\nu, \tag{7.12}$$

where $\nu = \Delta t/\Delta x$.

7.4 The Lax-Friedrichs method

As discussed at the end of Section 7.1, the finite volume method works by approximating the average flux at the boundaries of a cell, $\mathcal{F}(U_{j+1/2})$. One way to approximate this value would be to take the average of the flux at x_{j+1} and x_j given by $f(U_{j+1})$ and $f(U_j)$, which can be computed directly. I.e. define the approximate flux as

$$\mathcal{F}(U^n_{j+1/2}) = \frac{1}{2}\left(f(U^n_{j+1}) + f(U^n_j)\right). \tag{7.13}$$

The CFL condition for this case is $\Delta x \ge a\Delta t$, or expressed as a condition on the Courant number given by Eq. 7.12,

$$C \le 1. \tag{7.14}$$

The resulting method is, however, very unstable and cannot be used even with a timestep satisfying the CFL condition, Eq. 7.14. To recover stability, an addtional term is added to make the flux at time step n as approximated by the Lax-Friedrichs method, $\mathcal{F}^{\mathrm{LF}}$, satisfy

$$\mathcal{F}^{\mathrm{LF}} = \frac{1}{2}\left(f(U_{j+1}) + f(U_j)\right) - \frac{1}{2\nu}\left(U_{j+1} - U_j\right). \tag{7.15}$$

The additional term adds numerical diffusion, damping out instabilities but also smearing out the solution. Satisfying the CFL condition it is stable.

Substituting Eq. 7.15 for F in Eq. 7.8 results in

$$U_j^{n+1} = \frac{1}{2}\left[U_{j+1}^n + U_{j-1}^n\right] - \frac{\nu}{2}\left[f(U_{j+1}^n) - f(U_{j-1}^n)\right]. \tag{7.16}$$

Considering the constant coefficient case, i.e. $f(U_{j+1}) = a_0 U_{j+1}$, the method is reduced to

$$U_j^{n+1} = \frac{1}{2}\left[U_{j+1}^n + U_{j-1}^n\right] - \frac{\nu}{2}a_0\left[U_{j+1}^n - U_{j-1}^n\right]. \tag{7.17}$$

To check whether the Lax-Friedrichs method is a TVD method we rewrite this equation to the form of Eq. 7.11 by adding and subtracting U_j^n and $(\nu/2)a_0 U_j^n$, resulting in

$$U_j^{n+1} = U_j^n - \left[\frac{1}{2} + \frac{\nu}{2}a_0\right]\left(U_j^n - U_{j-1}^n\right) + \left[\frac{1}{2} - \frac{\nu}{2}a_0\right]\left(U_{j+1}^n - U_j^n\right), \tag{7.18}$$

which is of the form of Eq. 7.11 with the identifications

$$C_{j-1/2} = \frac{1}{2} + \frac{\nu}{2}a_0 \quad \forall j,$$

and

$$D_{j+1/2} = \frac{1}{2} - \frac{\nu}{2}a_0 \quad \forall j.$$

It can be easily verified that when the CFL condition is met all the conditions for a method to be TVD from Section 7.2 are met.

7.5 Flux limiters

Flux limiters, also called slope limiters, are used to make the solutions obtained with high-resolution schemes total variation diminishing. They work by smoothing out the solution when it tends towards discontinuity.

A flux limiter comes in the form of a function $\delta(\theta)$ of the so-called 'smoothness' θ. The smoothness at x_j, θ_j, is given by

$$\theta_j = \frac{U_j - U_{j-1}}{U_{j+1} - U_j}, \tag{7.19}$$

which is the ratio of successive differences around x_j.

For a second-order scheme to be TVD the limiter must satisfy a few conditions. These conditions can be represented in the Sweby diagram, displayed in Fig. 7.2. For a limiter to be TVD it must lie entirely within the area marked in the Sweby diagram.

There exist many limiters that satisfy these conditions. A list can be found on Wikipedia[1]. The minmod limiter, satisfying

$$\delta(\theta) = \max\left[0, \min(1, \theta)\right],$$

[1]See http://en.wikipedia.org/wiki/Flux_limiter#Limiter_functions

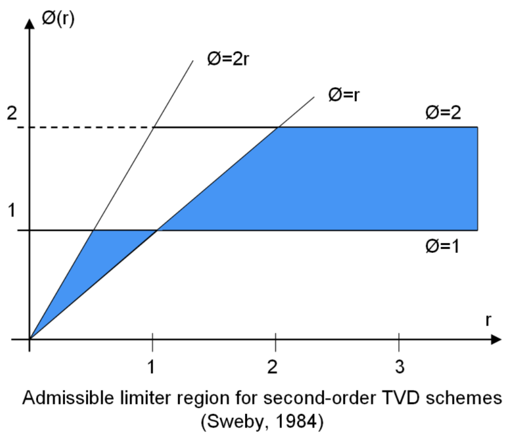

Figure 7.2: The Sweby diagram. Any limiter lying entirely within the marked region is a TVD limiter

sits along the bottom of the marked region in the Sweby diagram, giving the most diffusive TVD limiter. The superbee limiter, satisfying

$$\delta(\theta) = \max\left[0, \min(1, 2\theta), \min(2, \theta)\right],$$

sits along the top of the marked region in the Sweby diagram, giving the least diffusive TVD limiter. The Woordward limiter, satsifying

$$\delta(\theta) = \max\left[0, \min\left(\frac{1+\theta}{2}, 2, 2\theta\right)\right],$$

sits between the two and is continuous at $(1, 1)$.

7.6 TVD Lax-Friedrichs

Applying flux limiters to the Lax-Friedrichs scheme the Lax-Friedrichs TVD scheme (TVDLF) can be constructed. For this scheme the approximate flux at $x_{j+1/2}$, $\mathcal{F}_{j+1/2}$, satisfies

$$\mathcal{F}_{j+1/2} = \frac{1}{2}\left(f(U^{\mathrm{L}}_{j+1/2}) + f(U^{\mathrm{R}}_{j+1/2}) - \Phi\right), \tag{7.20}$$

where $U^{\mathrm{L}}_{j+1/2}$ is given by

$$U^{\mathrm{L}}_{j+1/2} = U_j + \frac{1}{2}\Delta x \delta_j,$$

$U^{\mathrm{R}}_{j+1/2}$ is given by

$$U^{\mathrm{R}}_{j+1/2} = U_{j+1} - \frac{1}{2}\Delta x \delta_{j+1},$$

and Φ is given by

$$\Phi = \frac{\Delta x}{\Delta t} \Delta U^{\mathrm{RL}}_{j+1/2}.$$

Here δ_j is a flux limiter evaluated around x_j and $\Delta U^{\mathrm{RL}}_{j+1/2}$ is given by

$$\Delta U^{\mathrm{RL}}_{j+1/2} = U^{\mathrm{R}}_{j+1/2} - U^{\mathrm{L}}_{j+1/2}.$$

7.7 Initial and boundary conditions

Simulation requires conditions at the boundary of the grid. To simulate one needs to specify both initial conditions (Section 7.7.1) and boundary conditions (Section 7.7.2).

7.7.1 Initial conditions

The initial conditions consist out of the set of cell averages of ω for each cell at time t^0, U^0_j for all j.

7.7.2 Boundary conditions

Three types of boundary conditions are considered. Inflow conditions, outflow conditions and solid boundary conditions.

Inflow conditions

Inflow conditions hold when the cell average of ω in a boundary cell is known as a function of time. It may be written as

$$U^n_i = \phi(t^n), \quad \forall n, \tag{7.21}$$

where the ith cell is a cell at the boundary and ϕ is known.

Outflow conditions

Outflow conditions hold when a boundary of the cellgrid doesn't represent a boundary of the simulated domain. The cellgrid merely represents the boundary of the part of the simulated domain that is of interest. The flow at an outflow boundary should therefore behave as if there is no boundary.

A way to approximate this is to imagine extending the grid outside of the boundary and approximating the cell average of ω of the first cell outside the boundary from known values inside of the boundary, such that the flux at the boundary can still be computed.

Consider one dimensional flow. Adding the Taylor series for U_{i-1} and U_{i+1} around U_i, we find that

$$U_{i-1} + U_{i+1} = 2U_i + O(\Delta x^2). \tag{7.22}$$

If the outflow boundary is at the first cell, we can use the approximation

$$U_{i-1} \approx 2U_i - U_{i+1}, \tag{7.23}$$

where the ith cell is at the left boundary. If the outflow boundary is at the last cell, we can use the approximation

$$U_{i+1} \approx 2U_i - U_{i-1}, \tag{7.24}$$

where the ith cell is at the right boundary. Now the fluxes at the left and right boundary, $\mathcal{F}_{i-1/2}$ and $\mathcal{F}_{i+1/2}$, respectively, can be computed.

Solid boundaries

Solid boundaries can be implemented by making the flux at the boundary 0, i.e.

$$\mathcal{F}_{i-1/2} = 0 \tag{7.25}$$

at the left boundary and

$$\mathcal{F}_{i+1/2} = 0 \tag{7.26}$$

at the right boundary.

Bibliography

[1] Robert D. Cook, David S. Malkus, Michael E. Plesha, and Robert J. Witt. *Concepts and Applications of Finite Element Analysis*. John Wiley and Sons. Inc, fourth edition, 2002.

[2] J. J. Monaghan. Smoothed particle hydrodynamics. *araa*, 30:543–574, 1992.

[3] J J Monaghan. Smoothed particle hydrodynamics. *Reports on Progress in Physics*, 68(8):1703, 2005.