Computational Physics

Prof. Matthias Troyer

ETH Zürich, 2005/2006

Contents

Chapter 1

Introduction

1.1 General

For **physics students** the computational physics courses are recommended prerequisites for any computationally oriented semester thesis, proseminar, diploma thesis or doctoral thesis.

For **computational science and engineering (RW) students** the computational physics courses are part of the "Vertiefung" in theoretical physics.

1.1.1 Lecture Notes

All the lecture notes, source codes, applets and supplementary material can be found on our web page `http://www.itp.phys.ethz.ch/lectures/RGP/`.

1.1.2 Exercises

Programming Languages

Except when a specific programming language or tool is explicitly requested you are free to choose any programming language you like. Solutions will often be given either as C++ programs or Mathematica Notebooks.

If you do not have much programming experience we recommend to additionally attend the"Programmiertechniken" lecture on Wednesday.

Computer Access

The lecture rooms offer both Linux workstations, for which accounts can be requested with the computer support group of the physics department in the HPR building, as well as connections for your notebook computers. In addition you will need to sign up for accounts on the supercomputers.

1.1.3 Prerequisites

As a prerequisite for this course we expect knowledge of the following topics. Please contact us if you have any doubts or questions.

Computing

- Basic knowledge of UNIX
- At least one procedural programming language such as C, C++, Pascal, Modula or FORTRAN. C++ knowledge is preferred.
- Knowledge of a symbolic mathematics program such as Mathematica or Maple.
- Ability to produce graphical plots.

Numerical Analysis

- Numerical integration and differentiation
- Linear solvers and eigensolvers
- Root solvers and optimization
- Statistical analysis

Physics

- Classical mechanics
- Classical electrodynamics
- Classical statistical physics

1.1.4 References

1. J.M. Thijssen, *Computational Physics*, Cambridge University Press (1999) ISBN 0521575885

2. Nicholas J. Giordano, *Computational Physics*, Pearson Education (1996) ISBN 0133677230.

3. Harvey Gould and Jan Tobochnik, *An Introduction to Computer Simulation Methods*, 2nd edition, Addison Wesley (1996), ISBN 00201506041

4. Tao Pang, *An Introduction to Computational Physics*, Cambridge University Press (1997) ISBN 0521485924

5. D. Landau and K. Binder, *A Guide to Monte Carlo Simulations in Statistical Physics*, Cambridge University Press (2000), ISBN 0521653665

6. Wolfgang Kinzel und Georg Reents *Physik per Computer*, Spektrum Akademischer Verlag, ISBN 3827400201; english edition: *Physics by Computer*, Springer Verlag, ISBN 354062743X

7. Dietrich Stauffer and Ammon Aharony, *Introduction to percolation theory*, Taylor & Francis(1991) ISBN 0748402535

8. Various articles in the journal *Computers in Physics* and its successor journal *Computers in Science and Engineering*

1.2 Overview

1.2.1 What is computational physics?

Computational physics is a new way of doing physics research, next to experiment and theory. Traditionally, the experimentalist has performed measurements on real physical systems and the theoretical physicist has explained these measurements with his theories. While this approach to physics has been extremely successful, and we now know the basis equations governing most of nature, we face the problem that an exact solution of the basic theories is often possible only for very simplified models. Approximate analytical methods, employing e.g. mean-field approximations or perturbation theory extend the set of solvable problems, but the question of validity of these approximation remains – and many cases are known where these approximations even fail completely.

The development of fast digital computers over the past sixty years has provided us with the possibility to quantitatively solve many of these equations not only for simplified toy models, but for realistic applications. And, indeed, in fields such as fluid dynamics, electrical engineering or quantum chemistry, computer simulations have replaced not only traditional analytical calculations but also experiments. Modern aircraft (e.g. the new Boeing and Airbus planes) and ships (such as the Alinghi yachts) are designed on the computer and only very few scale models are built to test the calculations.

Besides these fields, which have moved from "physics" to "engineering", simulations are also of fundamental importance in basic physics research to:

- solve problems that cannot be solved analytically
- check the validity of approximations and effective theories
- quantitatively compare theories to experimental measurements
- visualize complex data sets
- control and perform experimental measurements

In this lecture we will focus on the first three applications, starting from simple classical one-body problems and finishing with quantum many body problems in the summer semester.

Already the first examples in the next chapter will show one big advantage of numerical simulations over analytical solutions. Adding friction or a third particle to the Kepler problem makes it unsolvable analytically, while a program written to solve the Kepler problem numerically can easily be extended to cover these cases and allows realistic modelling.

1.2.2 Topics

In this lecture we will focus on classical problems. Computational quantum mechanics will be taught in the summer semester.

- Physics:
  - Classical few-body problems
  - Classical many-body problems
  - Linear and non-linear wave equations
  - Other important partial differential equations
  - Monte Carlo integration
  - Percolation
  - Spin models
  - Phase transitions
  - Finite Size Scaling
  - Algorithms for N-body problems
  - Molecular Dynamics
- Computing:
  - Mathematica
  - Vector supercomputing
  - Shared memory parallel computing
  - Distributed memory parallel computing

1.3 Programming Languages

There have been many discussions and fights about the "perfect language" for numerical simulations. The simple answer is: it depends on the problem. Here we will give a short overview to help you choose the right tool.

1.3.1 Symbolic Algebra Programs

Mathematica and Maple have become very powerful tools and allow symbolic manipulation at a high level of abstraction. They are useful not only for exactly solvable problems but also provide powerful numerical tools for many simple programs. Choose Mathematica or Maple when you either want an exact solution or the problem is not too complex.

1.3.2 Interpreted Languages

Interpreted languages range from simple shell scripts and perl programs, most useful for data handling and simple data analysis to fully object-oriented programming languages such as Python. We will regularly use such tools in the exercises.

1.3.3 Compiled Procedural Languages

are substantially faster than the interpreted languages discussed above, but usually need to be programmed at a lower level of abstraction (e.g. manipulating numbers instead of matrices).

FORTRAN (FORmula TRANslator)

was the first scientific programming languages. The simplicity of FORTRAN 77 and earlier versions allows aggressive optimization and unsurpassed performance. The disadvantage is that complex data structures such as trees, lists or text strings, are hard to represent and manipulate in FORTRAN.

Newer versions of FORTRAN (FORTRAN 90/95, FORTRAN 2000) converge towards object oriented programming (discussed below) but at the cost of decreased performance. Unless you have to modify an existing FORTRAN program use one of the languages discussed below.

Other procedural languages: C, Pascal, Modula,...

simplify the programming of complex data structures but cannot be optimized as aggressively as FORTRAN 77. This can lead to performance drops by up to a factor of two! Of all the languages in this category C is the best choice today.

1.3.4 Object Oriented Languages

The class concept in object oriented languages allows programming at a higher level of abstraction. Not only do the programs get simpler and easier to read, they also become

easier to debug. This is usually paid for by an "abstraction penalty", sometimes slowing programs down by more than a factor of ten if you are not careful.

Java

is very popular in web applications since a compiled Java program will run on any machine, though not at the optimal speed. Java is most useful in small graphics applets for simple physics problems.

C++

Two language features make C++ one of the best languages for scientific simulations: operator overloading and generic programming. Operator overloading allows to define mathematical operations such multiplication and addition not only for numbers but also for objects such as matrices, vectors or group elements. Generic programming, using template constructs in C++, allow to program at a high level of abstraction, without incurring the abstraction penalty of object oriented programming. We will often provide C++ programs as solutions for the exercises. If you are not familiar with the advanced features of C++ we recommend to attend the "Programmiertechniken" lecture on Wednesday.

1.3.5 Which programming language should I learn?

We recommend C++ for three reasons:

- object oriented programming allows to express codes at a high level of abstraction
- generic programming enables aggressive optimization, similar to FORTRAN
- C++-knowledge will help you find a job.

Chapter 2

The Classical Few-Body Problem

2.1 Solving Ordinary Differential Equations

2.1.1 The Euler method

The first set of problems we address are simple initial value problems of first order ordinary differential equations of the form

$$\frac{dy}{dt} = f(y,t) \tag{2.1}$$
$$y(t_0) = y_0 \tag{2.2}$$

where the initial value y_0 at the starting time t_0 as well as the time derivative $f(y,t)$ is given. This equations models, for example, simple physical problems such as radioactive decay

$$\frac{dN}{dt} = -\lambda N \tag{2.3}$$

where N is the number of particles and λ the decay constant, or the "coffee cooling problem"

$$\frac{dT}{dt} = -\gamma(T - T_{\text{room}}) \tag{2.4}$$

where T is the temperature of your cup of coffee, T_{room} the room temperature and γ the cooling rate.

For these simple problems an analytical solution can easily be found by rearranging the differential equation to

$$\frac{dT}{T - T_{\text{room}}} = -\gamma dt, \tag{2.5}$$

integrating both sides of this equation

$$\int_{T(0)}^{T(t)} \frac{dT}{T - T_{\text{room}}} = -\gamma \int_0^t dt, \tag{2.6}$$

evaluating the integral

$$\ln(T(t) - T_{\text{room}}) - \ln(T(0) - T_{\text{room}}) = -\gamma t \tag{2.7}$$

and solving this equation for $T(t)$

$$T(t) = T_{\text{room}} + (T(0) - T_{\text{room}}) \exp(-\gamma t). \tag{2.8}$$

While the two main steps, evaluating the integral (2.6) and solving the equation (2.7) could easily be done analytically in this simple case, this will not be the case in general.

Numerically, the value of y at a later time $t + \Delta t$ can easily be approximated by a Taylor expansion up to first order

$$y(t_0 + \Delta t) = y(t_0) + \Delta t \frac{dy}{dt} = y_0 + \Delta t f(y_0, t_0) + \mathrm{O}(\Delta \tau^2) \tag{2.9}$$

Iterating this equation and introducing the notation $t_n = t_0 + n\Delta t$ and $y_n = y(t_n)$ we obtain the Euler algorithm

$$y_{n+1} = y_n + \Delta t f(y_n, t_n) + \mathrm{O}(\Delta \tau^2) \tag{2.10}$$

In contrast to the analytical solution which was easy for the simple examples given above but can be impossible to perform on more complex differential equations, the Euler method retains its simplicity no matter which differential equation it is applied to.

2.1.2 Higher order methods

Order of integration methods

The truncation of the Taylor expansion after the first term in the Euler method introduces an error of order $\mathrm{O}(\Delta t^2)$ at each time step. In any simulation we need to control this error to ensure that the final result is not influenced by the finite time step. We have two options if we want to reduce the numerical error due to ths finite time step Δt. We can either choose a smaller value of Δt in the Euler method or choose a *higher order method*.

A method which introduces an error of order $\mathrm{O}(\Delta t^n)$ in a single time step is said to be *locally* of n-th order. Iterating a locally n-th order method over a fixed time interval T these truncation errors add up: we need to perform $T/\Delta t$ time steps and at each time step we pick up an error of order $\mathrm{O}(\Delta t^n)$. The total error over the time T is then:

$$\frac{T}{\Delta t} \mathrm{O}(\Delta t^n) = \mathrm{O}(\Delta t^{n-1}) \tag{2.11}$$

and the method is *globally* of $(n-1)$-th order.

The Euler method, which is of second order locally, is usually called a first order method since it is globally of first order.

Predictor-corrector methods

One straightforward idea to improve the Euler method is to evaluate the derivative dy/dt not only at the initial time t_n but also at the next time t_{n+1}:

$$y_{n+1} \approx y_n + \frac{\Delta t}{2} \left[f(y_n, t_n) + f(y_{n+1}, t_{n+1}) \right]. \tag{2.12}$$

This is however an *implicit* equation since y_{n+1} appears on both sides of the equation. Instead of solving this equation numerically, we first *predict* a rough estimate $\tilde{y}_{n+1}$ using the Euler method:

$$\tilde{y}_{n+1} = y_n + \Delta t f(y_n, t_n) \tag{2.13}$$

and then use this estimate to *correct* this Euler estimate in a second step by using $\tilde{y}_{n+1}$ instead of y_{n+1} in equation (2.12):

$$y_{n+1} \approx y_n + \frac{\Delta t}{2} \left[f(y_n, t_n) + f(\tilde{y}_{n+1}, t_{n+1}) \right]. \tag{2.14}$$

This correction step can be repeated by using the new estimate for y_{n+1} instead of $\tilde{y}_{n+1}$ and this is iterated until the estimate converges.

Exercise: determine the order of this predictor-corrector method.

The Runge-Kutta methods

The Runge-Kutta methods are families of systematic higher order improvements over the Euler method. The key idea is to evaluate the derivative dy/dt not only at the end points t_n or t_{n+1} but also at intermediate points such as:

$$y_{n+1} = y_n + \Delta t f\left(t_n + \frac{\Delta t}{2}, y\left(t_n + \frac{\Delta t}{2}\right)\right) + \mathrm{O}(\Delta t^3). \tag{2.15}$$

The unknown solution $y(t_n + \Delta t/2)$ is again approximated by an Euler step, giving the second order Runge-Kutta algorithm:

$$\begin{aligned} k_1 &= \Delta t f(t_n, y_n) \\ k_2 &= \Delta t f(t_n + \Delta t/2, y_n + k_1/2) \\ y_{n+1} &= y_n + k_2 + \mathrm{O}(\Delta t^3) \end{aligned} \tag{2.16}$$

The general ansatz is

$$y_{n+1} = y_n + \sum_{i=1}^{N} \alpha_i k_i \tag{2.17}$$

where the approximations k_i are given by

$$k_i = \Delta t f(y_n + \sum_{j=1}^{N-1} \nu_{ij} k_i, t_n + \sum_{j=1}^{N-1} \nu_{ij} \Delta t) \tag{2.18}$$

and the parameters α_i and ν_{ij} are chosen to obtain an N-th order method. Note that this choice is usually not unique.

The most widely used Runge-Kutta algorithm is the fourth order method:

$$\begin{aligned} k_1 &= \Delta t f(t_n, y_n) \\ k_2 &= \Delta t f(t_n + \Delta t/2, y_n + k_1/2) \\ k_3 &= \Delta t f(t_n + \Delta t/2, y_n + k_2/2) \\ k_4 &= \Delta t f(t_n + \Delta t, y_n + k_3) \\ y_{n+1} &= y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + \mathrm{O}(\Delta t^5) \end{aligned} \tag{2.19}$$

in which two estimates at the intermediate point $t_n + \Delta t/2$ are combined with one estimate at the starting point t_n and one estimate at the end point $t_n = t_0 + n\Delta t$.

Exercise: check the order of these two Runge-Kutta algorithms.

2.2 Integrating the classical equations of motion

The most common ordinary differential equation you will encounter are Newton's equation for the classical motion for N point particles:

$$m_i \frac{d\vec{v}_i}{dt} = \vec{F}_i(t, \vec{x}_1, \ldots, \vec{x}_N, \vec{v}_1, \ldots, \vec{v}_N) \tag{2.20}$$

$$\frac{d\vec{x}_i}{dt} = \vec{v}_i, \tag{2.21}$$

where m_i, $\vec{v}_i$ and $\vec{x}_i$ are the mass, velocity and position of the i-th particle and $\vec{F}_i$ the force acting on this particle.

For simplicity in notation we will restrict ourselves to a single particle in one dimension, before discussing applications to the classical few-body and many-body problem. We again label the time steps $t_{n+1} = t_n + \Delta t$, and denote by x_n and v_n the approximate solutions for $x(t_n)$ and $v(t_n)$ respectively. The accelerations are given by $a_n = a(t_n, x_n, v_n) = F(t_n, x_n, v_n)/m$.

The simplest method is again the forward-Euler method

$$\begin{aligned} v_{n+1} &= v_n + a_n \Delta t \\ x_{n+1} &= x_n + v_n \Delta t. \end{aligned} \tag{2.22}$$

which is however unstable for oscillating systems as can be seen in the Mathematica notebook on the web page. For a simple harmonic oscillator the errors will increase exponentially over time no matter how small the time step Δt is chosen and the forward-Euler method should thus be avoided!

For velocity-independent forces a surprisingly simple trick is sufficient to stabilize the Euler method. Using the backward difference $v_{n+1} \approx (x_{n+1} - x_n)/\Delta t$ instead of a forward difference $v_n \approx (x_{n+1} - x_n)/\Delta t$ we obtain the stable backward-Euler method:

$$\begin{aligned} v_{n+1} &= v_n + a_n \Delta t \\ x_{n+1} &= x_n + v_{n+1} \Delta t, \end{aligned} \tag{2.23}$$

where the new velocity v_{n+1} is used in calculating the positions x_{n+1}.

A related stable algorithm is the mid-point method, using a central difference:

$$\begin{aligned} v_{n+1} &= v_n + a_n \Delta t \\ x_{n+1} &= x_n + \frac{1}{2}(v_n + v_{n+1}) \Delta t. \end{aligned} \tag{2.24}$$

Equally simple, but surprisingly of second order is the leap-frog method, which is one of the commonly used methods. It evaluates positions and velocities at different

times:

$$\begin{aligned} v_{n+1/2} &= v_{n-1/2} + a_n \Delta t \\ x_{n+1} &= x_n + v_{n+1/2} \Delta t. \end{aligned} \tag{2.25}$$

As this method is not self-starting the Euler method is used for the first half step:

$$v_{1/2} = v_0 + \frac{1}{2} a_0 \Delta t. \tag{2.26}$$

For velocity-dependent forces the second-order Euler-Richardson algorithm can be used:

$$\begin{aligned} a_{n+1/2} &= a\left(x_n + \frac{1}{2} v_n \Delta t,\ v_n + \frac{1}{2} a_n \Delta t,\ t_n + \frac{1}{2} \Delta t\right) \\ v_{n+1} &= v_n + a_{n+1/2} \Delta t \\ x_{n+1} &= x_n + v_n \Delta t + \frac{1}{2} a_{n+1/2} \Delta t^2. \end{aligned} \tag{2.27}$$

The most commonly used algorithm is the following form of the Verlet algorithm ("velocity Verlet"):

$$\begin{aligned} x_{n+1} &= x_n + v_n \Delta t + \frac{1}{2} a_n (\Delta t)^2 \\ v_{n+1} &= v_n + \frac{1}{2}(a_n + a_{n+1}) \Delta t. \end{aligned} \tag{2.28}$$

It is third order in the positions and second order in the velocities.

2.3 Boundary value problems and "shooting"

So far we have considered only the initial value problem, where we specified both the initial position and velocity. Another type of problems is the boundary value problem where instead of two initial conditions we specify one initial and one final condition. Examples can be:

- We lanuch a rocket from the surface of the earth and want it to enter space (defined as an altitude of 100km) after one hour. Here the initial and final positions are specified and the question is to estimate the required power of the rocket engine.
- We fire a cannon ball from ETH Hnggerberg and want it to hit the tower of the university of Zürich. The initial and final positions as well as the initial speed of the cannon ball is specified. The question is to determine the angle of the cannon barrel.

Such boundary value problems are solved by the "shooting" method which should be familiar to Swiss students from their army days. In the second example we guess an angle for the cannon, fire a shot, and then iteratively adjust the angle until we hit our target.

More formally, let us again consider a simple one-dimensional example but instead of specifying the initial position x_0 and velocity v_0 we specify the initial position $x(0) = x_0$ and the final position after some time t as $x(t) = x_f$. To solve this problem we

1. guess an initial velocity $v_0 = \alpha$
2. define $x(t;\alpha)$ as the numerically integrated value of for the final position as a function of α
3. numerically solve the equation $x(t;\alpha) = x_f$

We thus have to combine one of the above integrators for the equations of motion with a numerical root solver.

2.4 Numerical root solvers

The purpose of a root solver is to find a solution (a root) to the equation

$$f(x) = 0, \tag{2.29}$$

or in general to a multi-dimensional equation

$$\vec{f}(\vec{x}) = 0. \tag{2.30}$$

Numerical root solvers should be well known from the numerics courses and we will just review three simple root solvers here. Keep in mind that in any serious calculation it is usually best to use a well optimized and tested library function over a hand-coded root solver.

2.4.1 The Newton and secant methods

The Newton method is one of best known root solvers, however it is not guaranteed to converge. The key idea is to start from a guess x_0, linearize the equation around that guess

$$f(x_0) + (x - x_0)f'(x_0) = 0 \tag{2.31}$$

and solve this linearized equation to obtain a better estimate x_1. Iterating this procedure we obtain the **Newton method**:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{2.32}$$

If the derivative f' is not known analytically, as is the case in our shooting problems, we can estimate it from the difference of the last two points:

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} \tag{2.33}$$

Substituting this into the Newton method (2.32) we obtain the **secant method**:

$$x_{n+1} = x_n - (x_n - x_{n-1})\frac{f(x_n)}{f(x_n) - f(x_{n-1})}. \tag{2.34}$$

The Newton method can easily be generalized to higher dimensional equations, by defining the matrix of derivatives

$$A_{ij}(\vec{x}) = \frac{\partial f_i(\vec{x})}{\partial x_j} \tag{2.35}$$

to obtain the **higher dimensional Newton method**

$$\vec{x}_{n+1} = \vec{x}_n - A^{-1}\vec{f}(\vec{x}) \tag{2.36}$$

If the derivatives $A_{ij}(\vec{x})$ are not known analytically they can be estimated through finite differences:

$$A_{ij}(\vec{x}) = \frac{f_i(\vec{x} + h_j\vec{e_j}) - f_i(\vec{x})}{h_j} \qquad \text{with} \qquad h_j \approx x_j\sqrt{\varepsilon} \tag{2.37}$$

where ε is the machine precision (about 10^{-16} for double precision floating point numbers on most machines).

2.4.2 The bisection method and regula falsi

Both the bisection method and the regula falsi require two starting values x_0 and x_1 surrounding the root, with $f(x_0) < 0$ and $f(x_1) > 0$ so that under the assumption of a continuous function f there exists at least one root between x_0 and x_1.

The **bisection method** performs the following iteration

1. define a mid-point $x_m = (x_0 + x_1)/2$.
2. if $\text{sign}f(x_m) = \text{sign}f(x_0)$ replace $x_0 \leftarrow x_m$ otherwise replace $x_1 \leftarrow x_m$

until a root is found.

The **regula falsi** works in a similar fashion:

1. estimate the function f by a straight line from x_0 to x_1 and calculate the root of this linearized function: $x_2 = (f(x_0)x_1 - f(x_1)x_0)/(f(x_1) - f(x_0)$
2. if $\text{sign}f(x_2) = \text{sign}f(x_0)$ replace $x_0 \leftarrow x_2$ otherwise replace $x_1 \leftarrow x_2$

In contrast to the Newton method, both of these two methods will always find a root.

2.4.3 Optimizing a function

These root solvers can also be used for finding an extremum (minimum or maximum) of a function $f(\vec{x})$, by looking a root of

$$\nabla f(\vec{x}) = 0. \tag{2.38}$$

While this is efficient for one-dimensional problems, but better algorithms exist.

In the following discussion we assume, without loss of generality, that we want to minimize a function. The simplest algorithm for a multi-dimensional optimization is

steepest descent, which always looks for a minimum along the direction of steepest gradient: starting from an initial guess $\vec{x}_n$ a one-dimensional minimization is applied to determine the value of λ which minimizes

$$f(\vec{x}_n + \lambda \nabla f(\vec{x}_n)) \tag{2.39}$$

and then the next guess $\vec{x}_{n+1}$ is determined as

$$\vec{x}_{n+1} = \vec{x}_n + \lambda \nabla f(\vec{x}_n) \tag{2.40}$$

While this method is simple it can be very inefficient if the "landscape" of the function f resembles a long and narrow valley: the one-dimensional minimization will mainly improve the estimate transverse to the valley but takes a long time to traverse down the valley to the minimum. A better method is the **conjugate gradient** algorithm which approximates the function locally by a paraboloid and uses the minimum of this paraboloid as the next guess. This algorithm can find the minimuim of a long and narrow parabolic valley in one iteration! For this and other, even better, algorithms we recommend the use of **library functions**.

One final word of warning is that all of these minimizers will only find a **local minimum**. Whether this local minimum is also the global minimum can never be decided by purely numerically. A necessary but never sufficient check is thus to start the minimization not only from one initial guess but to try many initial points and check for consistency in the minimum found.

2.5 Applications

In the last section of this chapter we will mention a few interesting problems that can be solved by the methods discussed above. This list is by no means complete and should just be a starting point to get you thinking about which other interesting problems you will be able to solve.

2.5.1 The one-body problem

The one-body problem was already discussed in some examples above and is well known from the introductory classical mechanics courses. Here are a few suggestions that go beyond the analytical calculations performed in the introductory mechanics classes:

Friction

Friction is very easy to add to the equations of motion by including a velocity-dependent term such as:

$$\frac{d\vec{v}}{dt} = \vec{F} - \gamma |\vec{v}|^2 \tag{2.41}$$

while this term usually makes the problem impossible to solve analytically you will see in the exercise that this poses no problem for the numerical simulation.

Another interesting extension of the problem is adding the effects of spin to a thrown ball. Spinning the ball causes the velocity of airflow differ on opposing sides. This in

turn exerts leads to differing friction forces and the trajectory of the ball curves. Again the numerical simulation remains simple.

Relativistic equations of motion

It is equally simple to go from classical Newtonian equations of motion to Einsteins equation of motion in the special theory of relativity:

$$\frac{d\vec{p}}{dt} = \vec{F} \tag{2.42}$$

where the main change is that the momentum $\vec{p}$ is no longer simply $m\vec{v}$ but now

$$\vec{p} = \gamma m_0 \vec{v} \tag{2.43}$$

where m_0 is the mass at rest of the body,

$$\gamma = \sqrt{1 + \frac{|\vec{p}|^2}{m_0^2 c^2}} = \frac{1}{\sqrt{1 - \frac{|\vec{v}|^2}{c^2}}}, \tag{2.44}$$

and c the speed of light.

These equations of motion can again be discretized, for example in a forward-Euler fashion, either by using the momenta and positions:

$$\vec{x}_{n+1} = \vec{x}_n + \frac{\vec{p}_n}{\gamma m_0}\Delta t \tag{2.45}$$

$$\vec{p}_{n+1} = \vec{p}_n + \vec{F}_n \Delta t \tag{2.46}$$

or using velocities and positions

$$\vec{x}_{n+1} = \vec{x}_n + \vec{v}_n \Delta t \tag{2.47}$$

$$\vec{v}_{n+1} = \vec{v}_n + \frac{\vec{F}_n}{\gamma m_0}\Delta t \tag{2.48}$$

The only change in the program is a division by γ, but this small change has large consequences, one of which is that the velocity can never exceed the speed of light c.

2.5.2 The two-body (Kepler) problem

While the generalization of the integrators for equations of motion to more than one body is trivial, the two-body problem does not even require such a generalization in the case of forces that depend only on the relative distance of the two bodies, such as gravity. The equations of motion

$$m_1 \frac{d^2\vec{x}_1}{dt^2} = \vec{F}(\vec{x}_2 - \vec{x}_1) \tag{2.49}$$

$$m_2 \frac{d^2\vec{x}_2}{dt^2} = \vec{F}(\vec{x}_1 - \vec{x}_2) \tag{2.50}$$

where $\vec{F}(\vec{x}_2 - \vec{x}_1) = -\vec{F}(\vec{x}_2 - \vec{x}_1)$ we can perform a transformation of coordinates to center of mass and relative motion. The important *relative* motion gives a single body problem:

$$m\frac{d^2\vec{x}}{dt^2} = \vec{F}(\vec{x}) = -\nabla V(|\vec{x}|), \tag{2.51}$$

where $\vec{x} = \vec{x}_2 - \vec{x}_1$ is the distance, $m = m_1 m_2/(m_1 + m_2)$ the reduced mass, and V the potential

$$V(r) = -\frac{Gm}{r} \tag{2.52}$$

In the case of gravity the above problem is called the Kepler problem with a force

$$\vec{F}(\vec{x}) = -Gm\frac{\vec{x}}{|\vec{x}|^3} \tag{2.53}$$

and can be solved exactly, giving the famous solutions as either circles, ellipses, parabolas or hyperbolas.

Numerically we can easily reproduce these orbits but can again go further by adding terms that make an analytical solution impossible. One possibility is to consider a satellite in orbit around the earth and add **friction** due to the atmosphere. We can calculate how the satellite spirals down to earth and crashes.

Another extension is to consider effects of Einsteins **theory of general relativity**. In a lowest order expansion its effect on the Kepler problem is a modified potential:

$$V(r) = -\frac{Gm}{r}\left(1 + \frac{\vec{L}^2}{r^2}\right), \tag{2.54}$$

where $\vec{L} = m\vec{x} \times \vec{v}$ is the angular momentum and a constant of motion. When plotting the orbits including the extra $1/r^3$ term we can observe a rotation of the main axis of the elliptical orbit. The experimental observation of this effect on the orbit of Mercury was the first confirmation of Einsteins theory of general relativity.

2.5.3 The three-body problem

Next we go to three bodies and discuss a few interesting facts that can be checked by simulations.

Stability of the three-body problem

Stability, i.e. that a small perturbation of the initial condition leads only to a small change in orbits, is easy to prove for the Kepler problem. There are 12 degrees of freedom (6 positions and 6 velocities), but 11 integrals of motion:

- total momentum: 3 integrals of motion
- angular momentum: 3 integrals of motion
- center of mass: 3 integrals of motion

- Energy: 1 integral of motion
- Lenz vector: 1 integral of motion

There is thus only one degree of freedom, the initial position on the orbit, and stability can easily be shown.

In the three-body problem there are 18 degrees of freedom but only 10 integrals of motion (no Lenz vector), resulting in 8 degrees of freedom for the orbits. Even restricting the problem to planar motions in two dimensions does not help much: 12 degrees of freedom and 6 integrals of motion result in 6 degrees of freedom for the orbits.

Progress can be made only for the *restricted three-body problem*, where the mass of the third body $m_3 \to 0$ is assumed to be too small to influence the first two bodies which are assumed to be on circular orbits. This restricted three-body problem has four degrees of freedom for the third body and one integral of motion, the energy. For the resulting problem with three degrees of freedom for the third body the famous KAM (Kolmogorov-Arnold-Moser) theorem can be used to prove stability of moon-like orbits.

Lagrange points and Trojan asteroids

In addition to moon-like orbits, other (linearly) stable orbits are around two of the Lagrange points. We start with two bodies on circular orbits and go into a rotating reference frame at which these two bodies are at rest. There are then five positions, the five Lagrange points, at which a third body is also at rest. Three of these are colinear solutions and are unstable. The other two stable solutions form equilateral triangles.

Astronomical observations have indeed found a group of asteroids, the Trojan asteroids on the orbit of Jupiter, 60 degrees before and behind Jupiter. They form an equilateral triangle with the sun and Jupiter.

Numerical simulations can be performed to check how long bodies close to the perfect location remain in stable orbits.

Kirkwood gaps in the rings of Saturn

Going farther away from the sun we next consider the Kirkwood gaps in the rings of Saturn. Simulating a system consisting of Saturn, a moon of Saturn, and a very light ring particle we find that orbits where the ratio of the period of the ring particle to that of the moon are unstable, while irrational ratios are stable.

The moons of Uranus

Uranus is home to an even stranger phenomenon. The moons Janus and Epimetheus share the same orbit of 151472 km, separated by only 50km. Since this separation is less than the diameter of the moons (ca. 100-150km) one would expect that the moons would collide.

Since these moons still exist something else must happen and indeed a simulation clearly shows that the moons do not collide but instead switch orbits when they approach each other!

2.5.4 More than three bodies

Having seen these unusual phenomena for three bodies we can expect even stranger behavior for four or five bodies, and we encourage you to start exploring them with your programs.

Especially noteworthy is that for five bodies there are extremely unstable orbits that diverge in finite time: five bodies starting with the right initial positions and finite velocities can be infinitely far apart, and flying with infinite velocities after finite time! For more information see `http://www.ams.org/notices/199505/saari-2.pdf`

Chapter 3

Partial Differential Equations

In this chapter we will present algorithms for the solution of some simple but widely used partial differential equations (PDEs), and will discuss approaches for general partial differential equations. Since we cannot go into deep detail, interested students are referred to the lectures on numerical solutions of differential equations offered by the mathematics department.

3.1 Finite differences

As in the solution of ordinary differential equations the first step in the solution of a PDE is to discretize space and time and to replace differentials by differences, using the notation $x_n = n\Delta x$. We already saw that a first order differential $\partial f/\partial x$ can be approximated in first order by

$$\frac{\partial f}{\partial x} = \frac{f(x_{n+1}) - f(x_n)}{\Delta x} + \mathrm{O}(\Delta x) = \frac{f(x_n) - f(x_{n-1})}{\Delta x} + \mathrm{O}(\Delta x) \tag{3.1}$$

or to second order by the symmetric version

$$\frac{\partial f}{\partial x} = \frac{f(x_{n+1}) - f(x_{n-1})}{2\Delta x} + \mathrm{O}(\Delta x^2), \tag{3.2}$$

From these first order derivatives can get a second order derivative as

$$\frac{\partial^2 f}{\partial x^2} = \frac{f(x_{n+1}) + f(x_{n-1}) - 2f(x_n)}{\Delta x^2} + \mathrm{O}(\Delta x^2). \tag{3.3}$$

To derive a general approximation for an arbitrary derivative to any given order use the ansatz

$$\sum_{k=-l}^{l} a_k f(x_{n+k}), \tag{3.4}$$

insert the Taylor expansion

$$f(x_{n+k}) = f(x_n) + \Delta x f'(x_n) + \frac{\Delta x^2}{2} f''(x_n) + \frac{\Delta x^3}{6} f'''(x_n) + \frac{\Delta x^4}{4} f^{(4)}(x_n) + \ldots \tag{3.5}$$

and choose the values of a_k so that all terms but the desired derivative vanish.

As an example we give the fourth-order estimator for the second derivative

$$\frac{\partial^2 f}{\partial x^2} = \frac{-f(x_{n-2}) + 16f(x_{n-1}) - 30f(x_n) + 16f(x_{n+1}) - f(x_{n+2})}{12\Delta x^2} + \mathrm{O}(\Delta x^4). \quad (3.6)$$

and the second order estimator for the third derivative:

$$\frac{\partial^3 f}{\partial x^3} = \frac{-f(x_{n-2}) + 2f(x_{n-1}) - 2f(x_{n+1}) + f(x_{n+2})}{\Delta x^3} + \mathrm{O}(\Delta x^2). \quad (3.7)$$

Extensions to higher dimensions are straightforward, and these will be all the differential quotients we will need in this course.

3.2 Solution as a matrix problem

By replacing differentials by differences we convert the (non)-linear PDE to a system of (non)-linear equations. The first example to demonstrate this is determining an electrostatic or gravitational potential Φ given by the Poisson equation

$$\nabla^2 \Phi(\vec{x}) = -4\pi\rho(\vec{x}), \quad (3.8)$$

where ρ is the charge or mass density respectively and units have been chosen such that the coupling constants are all unity.

Discretizing space we obtain the system of linear equations

$$\begin{aligned} &\Phi(x_{n+1}, y_n, z_n) + \Phi(x_{n-1}, y_n, z_n) \\ &+\Phi(x_n, y_{n+1}, z_n) + \Phi(x_n, y_{n-1}, z_n) \\ &+\Phi(x_n, y_n, z_{n+1}) + \Phi(x_n, y_n, z_{n-1}) \\ &\qquad -6\Phi(x_n, y_n, z_n) \;=\; -4\pi\rho(x_n, y_n, z_n)\Delta x^2, \end{aligned} \quad (3.9)$$

where the density $\rho(x_n, y_n, z_n)$ is defined to be the average density in the cube with linear extension Δx around the point $\rho(x_n, y_n, z_n)$.

The general method to solve a PDE is to formulate this linear system of equations as a matrix problems and then to apply a linear equation solver to solve the system of equations. For small linear problems Mathematica can be used, or the `dsysv` function of the LAPACK library.

For larger problems it is essential to realize that the matrices produced by the discretization of PDEs are usually very sparse, meaning that only $\mathrm{O}(N)$ of the N^2 matrix elements are nonzero. For these sparse systems of equations, optimized iterative numerical algorithms exist[1] and are implemented in numerical libraries such as in the ITL library.[2]

[1] R. Barret, M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods* (SIAM, 1993)

[2] J.G. Siek, A. Lumsdaine and Lie-Quan Lee, *Generic Programming for High Performance Numerical Linear Algebra* in *Proceedings of the SIAM Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing (OO'98)* (SIAM, 1998); the library is availavle on the web at: `http://www.osl.iu.edu/research/itl/`

This is the most general procedure and can be used in all cases, including boundary value problems and eigenvalue problems. The PDE eigenvalue problem maps to a matrix eigenvalue problem, and an eigensolver needs to be used instead of a linear solver. Again there exist efficient implementations[3] of iterative algorithms for sparse matrices.[4]

For non-linear problems iterative procedures can be used to linearize them, as we will discuss below.

Instead of this general and flexible but brute-force method, many common PDEs allow for optimized solvers that we will discuss below.

3.3 The relaxation method

For the Poisson equation a simple iterative method exists that can be obtained by rewriting above equation as

$$\begin{aligned}\Phi(x_n,y_n,z_n) &= \frac{1}{6}[\Phi(x_{n+1},y_n,z_n)+\Phi(x_{n-1},y_n,z_n)+\Phi(x_n,y_{n+1},z_n)\\ &\quad +\Phi(x_n,y_{n-1},z_n)+\Phi(x_n,y_n,z_{n+1})+\Phi(x_n,y_n,z_{n-1})]\\ &\quad -\frac{2}{3}\pi\rho(x_n,y_n,z_n)\Delta x^2, \end{aligned} \tag{3.10}$$

The potential is just the average over the potential on the six neighboring sites plus a term proportinal to the density ρ.

A solution can be obtained by iterating equation 3.10:

$$\begin{aligned}\Phi(x_n,y_n,z_n) &\leftarrow \frac{1}{6}[\Phi(x_{n+1},y_n,z_n)+\Phi(x_{n-1},y_n,z_n)+\Phi(x_n,y_{n+1},z_n)\\ &\quad +\Phi(x_n,y_{n-1},z_n)+\Phi(x_n,y_n,z_{n+1})+\Phi(x_n,y_n,z_{n-1})]\\ &\quad -\frac{2}{3}\pi\rho(x_n,y_n,z_n)\Delta x^2, \end{aligned} \tag{3.11}$$

This iterative solver will be implemented in the exercises for two examples:

1. Calculate the potential between two concentric metal squares of size a and $2a$. The potential difference between the two squares is V. Starting with a potential 0 on the inner square, V on the outer square, and arbitrary values in-between, a two-dimensional variant of equation 3.11 is iterated until the differences drop below a given threshold. Since there are no charges the iteration is simply:

$$\Phi(x_n,y_n) \leftarrow \frac{1}{4}[\Phi(x_{n+1},y_n)+\Phi(x_{n-1},y_n)+\Phi(x_n,y_{n+1})+\Phi(x,y_{n-1})]. \tag{3.12}$$

2. Calculate the potential of a distribution of point charges: starting from an arbitrary initial condition, e.g. $\Phi(x_n,y_n,z_n)=0$, equation 3.11 is iterated until convergence.

[3] http://www.comp-phys.org/software/ietl/

[4] Z. Bai, J. Demmel and J. Dongarra (Eds.), *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide* (SIAM, 2000).

Since these iterations are quite slow it is important to improve them by one of two methods discussed below.

3.3.1 Gauss-Seidel Overrelaxtion

Gauss-Seidel overrelaxtion determines the change in potential according to equation 3.11 but then changes the value by a multiple of this proposed change:

$$\begin{aligned}
\Delta\Phi(x_n, y_n, z_n) &= \frac{1}{6}[\Phi(x_{n+1}, y_n, z_n) + \Phi(x_{n-1}, y_n, z_n) + \Phi(x_n, y_{n+1}, z_n) \\
&\quad +\Phi(x_n, y_{n-1}, z_n) + \Phi(x_n, y_n, z_{n+1}) + \Phi(x_n, y_n, z_{n-1})] \\
&\quad -\frac{2}{3}\pi\rho(x_n, y_n, z_n)\Delta x^2 - \Phi(x_n, y_n, z_n) \\
\Phi(x_n, y_n, z_n) &\leftarrow \Phi(x_n, y_n, z_n) + w\Delta\Phi(x_n, y_n, z_n)
\end{aligned} \tag{3.13}$$

with an overrelaxation factor of $1 < w < 2$. You can easily convince yourself, by considering a single charge and initial values of $\Phi(x_n, y_n, z_n) = 0$ that choosing value $w \geq 2$ is unstable.

3.3.2 Multi-grid methods

Multi-grid methods dramatically accelerate the convergence of many iterative solvers. We start with a very coarse grid spacing $\Delta x \Delta x_0$ and iterate

- solve the Poisson equation on the grid with spacing Δx
- refine the grid $\Delta x \leftarrow \Delta x/2$
- interpolate the potential at the new grid points
- and repeat until the desired final fine grid spacing Δx is reached.

Initially convergence is fast since we have a very small lattice. In the later steps convergence remains fast since we always start with a very good guess.

3.4 Solving time-dependent PDEs by the method of lines

3.4.1 The diffusion equation

Our next problem will include a first order time-derivative, with a partial differential equation of the form

$$\frac{\partial f(\vec{x}, t)}{\partial t} = F(f, t) \tag{3.14}$$

where f contains only spatial derivatives and the initial condition at time t_0 is given by

$$f(\vec{x}, t_0) = u(\vec{x}). \tag{3.15}$$

One common equation is the diffusion equation, e.g. for heat transport

$$\frac{\partial T(\vec{x},t)}{\partial t} = -\frac{K}{C\rho}\nabla^2 T(\vec{x},t) + \frac{1}{C\rho}W(\vec{x},t) \tag{3.16}$$

where T is the temperature, C the specific heat, ρ the density and K the thermal conductivity. External heat sources or sinks are specified by $W(\vec{x},t)$.

This and similar initial value problems can be solved by the method of lines: after discretizing the spatial derivatives we obtain a set of coupled ordinary differential equations which can be evolved fort each point along the time line (hence the name) by standard ODE solvers. In our example we obtain, specializing for simplicity to the one-dimensional case:

$$\frac{\partial T(x_n,t)}{\partial t} = -\frac{K}{C\rho\Delta x^2}\left[T(x_{n+1},t) + T(x_{n-1},t) - 2T(x_n,t)\right] + \frac{1}{C\rho}W(x_n,t) \tag{3.17}$$

Using a forward Euler algorithm we finally obtain

$$T(x_n,t+\Delta t) = T(x_n,t) - \frac{K\Delta t}{C\rho\Delta x^2}\left[T(x_{n+1},t) + T(x_{n-1},t) - 2T(x_n,t)\right] + \frac{\Delta t}{C\rho}W(x_n,t) \tag{3.18}$$

This will be implemented in the exercises and used in the supercomputing examples.

3.4.2 Stability

Great care has to be taken in choosing appropriate values of Δx and Δt, as too long time steps Δt immediately lead to instabilities. By considering the case where the temperature is 0 everywhere except at one point it is seen immediately, like in the case of overrelaxation that a choice of $K\Delta t/C\rho\Delta x^2 > 1/2$ is unstable. A detailed analysis, which is done e.g. in the lectures on numerical solutions of differential equations, shows that this heat equation solver is only stable for

$$\frac{K\Delta t}{C\rho\Delta x^2} < \frac{1}{4}. \tag{3.19}$$

We see that, for this PDE with second order spatial and first order temporal derivatives, it is not enough to just adjust Δt proportional to Δx, but $\Delta t \ll \mathrm{O}(\Delta x^2)$ is needed. Here it is even more important to **check for instabilities** than in the case of PDEs!

3.4.3 The Crank-Nicolson method

The simple solver above can be improved by replacing the forward Euler method for the time integration by a midpoint method:

$$T(x,t+\Delta t) = T(x,t) + \frac{K\Delta t}{2C\rho}\left[\nabla^2 T(x,t) + \nabla^2 T(x,t+\Delta t)\right] + \frac{\Delta t}{2C\rho}\left[W(x,t) + W(x,t+\Delta t)\right] \tag{3.20}$$

Discretizing space and introducing the linear operator A defined by

$$AT(x_n,t) = \frac{K\Delta t}{C\rho\Delta x^2}\left[T(x_{n+1},t) + T(x_{n-1},t) - 2T(x_n,t)\right] \tag{3.21}$$

to simplify the notation we obtain an *implicit* algorithm:

$$(2 \cdot \mathbf{1} - A)\vec{T}(t + \Delta t) = (2 - A)\vec{T}(t) + \frac{\Delta t}{C\rho}\left[\vec{W}(t) + \vec{W}(t + \Delta t)\right], \tag{3.22}$$

where $\mathbf{1}$ is the unit matrix and

$$\vec{T}(t) = (T(x_1, t), \ldots T(x_N, t)) \tag{3.23}$$

$$\vec{W}(t) = (W(x_1, t), \ldots W(x_N, t)) \tag{3.24}$$

are vector notations for the values of the temperature and heat source at each point. In contrast to the *explicit* solver (3.18) the values at time $t + \Delta t$ are not given explicitly on the right hand side but only as a solution to a linear system of equations. After evaluating the right hand side, still a linear equation needs to be solved. This extra effort, however, gives us greatly improved stability and accuracy.

Note that while we have discussed the Crank-Nicolson method here in the context of the diffusion equation, it can be applied to any time-dependent PDE.

3.5 The wave equation

3.5.1 A vibrating string

Another simple PDE is the wave equation, which we will study for the case of a string running along the x-direction and vibrating transversely in the y-direction:

$$\frac{\partial^2 y}{\partial t^2} = c^2 \frac{\partial^2 y}{\partial x^2}. \tag{3.25}$$

The wave velocity $c = \sqrt{T/\mu}$ is a function of the string tension T and the mass density μ of the string.

As you can easily verify, analytic solutions of this wave equation are of the form

$$y = f_+(x + ct) + f_-(x - ct). \tag{3.26}$$

To solve the wave equation numerically we again discretize time and space in the usual manner and obtain, using the the second order difference expressions for the second derivative:

$$\frac{y(x_i, t_{n+1}) + y(x_i, t_{n-1}) - 2y(x_i, t_n)}{(\Delta t)^2} \approx c^2 \frac{y(x_{i+1}, t_n) + y(x_{i-1}, t_n) - 2y(x_i, t_n)}{(\Delta x)^2}. \tag{3.27}$$

This can be transformed to

$$y(x_i, t_{n+1}) = 2(1 - \kappa^2)y(x_i, t_n) - y(x_i, t_{n-1}) + \kappa^2 \left[y(x_{i+1}, t_n) + y(x_{i-1}, t_n)\right], \tag{3.28}$$

with $\kappa = c\Delta t/\Delta x$.

Again, we have to choose the values of Δt and Δx carefully. Surprisingly, for the wave equation when choosing $\kappa = 1$ we obtain the exact solution without any error! To check this, insert the exact solution (3.26) into the difference equation (3.27).

Decreasing both Δx and Δt does not increase the accuracy but only the spatial and temporal resolution. This is a very special feature of the linear wave equation.

Choosing smaller time steps and thus $\kappa < 1$ there will be solutions propagating faster than the speed of light, but since they decrease with the square of the distance r^{-2} this does not cause any major problems.

On the other hand, choosing a slightly larger time step and thus $\kappa > 1$ has catastrophic consequences: these unphysical numerical solution *increase* and diverge rapidly, as can be seen in the Mathematica Notebook posted on the web page.

3.5.2 More realistic models

Real strings and musical instruments cause a number of changes and complications to the simple wave equation discussed above:

- Real strings vibrate in both the y and z direction, which is easy to implement.
- Transverse vibrations of the string cause the length of the string and consequently the string tension T to increase. This leads to an increase of the velocity c and the value of κ. The nice fact that $\kappa = 1$ gives the exact solution can thus no longer be used and special care has to be taken to make sure that $\kappa < 1$ even for the largest elongations of the string.
- Additionally there will be longitudinal vibrations of the string, with a much higher velocity $c_{||} \gg c$. Consequently the time step for longitudinal vibrations $\Delta t_{||}$ has to be chosen much smaller than for transverse vibrations. Instead of of simulating both transverse and longitudinal vibrations with the same small time step $\Delta t_{||}$ one still uses the larger time step Δt for the transverse vibrations but updates the transverse positions only every $\Delta t / \Delta t_{||}$ iterations of the longitudinal positions.
- Finally the string is not in vacuum and infinitely long, but in air and attached to a musical instrument. Both the friction of the air and forces exerted by the body of the instrument cause damping of the waves and a modified sound.

For more information about applications to musical instruments I refer to the article by N. Giordano in Computers in Phsyics **12**, 138 (1998). This article also discusses numerical approaches to the following problems

- How is sound created in an acoustic guitar, an electric guitar and a piano?
- What is the sound of these instruments?
- How are the strings set into motion in these instruments?

The simulation of complex instruments such as pianos still poses substantial unsolved challenges.

3.6 The finite element method

3.6.1 The basic finite element method

While the finite difference method used so far is simple and straightforward for regular mesh discretizations it becomes very hard to apply to more complex problems such as:

- spatially varying constants, such as spatially varying dielectric constants in the Poisson equation.
- irregular geometries such as airplanes or turbines.
- dynamically adapting geometries such as moving pistons.

In such cases the finite element method has big advantages over finite differences since it does not rely on a regular mesh discretization. We will discuss the finite element method using the one-dimensional Poisson equation

$$\phi''(x) = -4\pi\rho(x) \tag{3.29}$$

with boundary conditions

$$\phi(0) = \phi(1) = 0. \tag{3.30}$$

as our example.

The first step is to expand the solution $\phi(x)$ in terms of basis functions $\{v_i\}$, $i = 1, \ldots, \infty$ of the function space:

$$\phi(x) = \sum_{i=1}^{\infty} a_i v_i(x). \tag{3.31}$$

For our numerical calculation the infinite basis set needs to be truncated, choosing a finite subset $\{u_i\}$, $i = 1, \ldots, N$ of N linearly independent, but not necessarily orthogonal, functions:

$$\phi_N(x) = \sum_{i=1}^{N} a_i u_i(x). \tag{3.32}$$

The usual choice are functions localized around some mesh points x_i, which in contrast to the finite difference method do not need to form a regular mesh.

The coefficients $\vec{a} = (a_1, \ldots, a_N)$ are chosen to minimize the residual

$$\phi_N''(x) + 4\pi\rho(x) \tag{3.33}$$

over the whole interval. Since we can choose N coefficients we can impose N conditions

$$0 = g_i \int_0^1 \left[\phi_N''(x) + 4\pi\rho(x)\right] w_i(x) dx, \tag{3.34}$$

where the weight functions $w_i(x)$ are often chosen to be the same as the basis functions $w_i(x) = u_i(x)$. This is called the Galerkin method.

In the current case of a linear PDE this results in a linear system of equations

$$A\vec{a} = \vec{b} \tag{3.35}$$

with

$$\begin{aligned} A_{ij} &= -\int_0^1 u_i''(x) w_j(x) dx = \int_0^1 u_i'(x) w_j'(x) dx \\ b_i &= 4\pi \int_0^1 \rho(x) w_i(x) dx, \end{aligned} \tag{3.36}$$

where in the first line we have used integration by parts to circumvent problems with functions that are not twice differentiable.

A good and simple choice of local basis functions fulfilling the boundary conditions (3.30) are local triangles centered over the points $x_i = i\Delta x$ with $\Delta x = 1/(n+1)$:

$$u_i(x) = \begin{cases} (x - x_{i-1})/\Delta x & \text{for } x \in [x_i - 1, x_i] \\ (x_{i+1} - x)/\Delta x & \text{for } x \in [x_i, x_i + 1] \\ 0 & \text{otherwise} \end{cases}, \tag{3.37}$$

but other choices such as local parabolas are also possible.

With the above choice we obtain

$$A_{ij} = -\int_0^1 u_i''(x) u_j(x) dx = \int_0^1 u_i'(x) u_j'(x) dx = \begin{cases} 2/\Delta x & \text{for } i = j \\ -1/\Delta x & \text{for } i = j \pm 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.38}$$

and, choosing a charge density $\rho(x) = (\pi/4)\sin(\pi x)$

$$b_i = 4\pi \int_0^1 \rho(x) u_i(x) dx = \frac{1}{\Delta x} \left(2 \sin \pi x_i - \sin \pi x_{i-1} - \sin \pi x_{i+1}\right) \tag{3.39}$$

In the one-dimensional case the matrix A is tridiagonal and efficient linear solvers for this tridiagonal matrix can be found in the LAPACK library. In higher dimensions the matrices will usually be sparse band matrices and iterative solvers will be the methods of choice.

3.6.2 Generalizations to arbitrary boundary conditions

Our example assumed boundary conditions $\phi(0) = \phi(1) = 0$. These boundary conditions were implemented by ensuring that all basis functions $u_i(x)$ were zero on the boundary. Generalizations to arbitrary boundary conditions $\phi(0) = \phi_0$ and $\phi(1) = \phi_1$ are possible either by adding additional basis functions that are non-zero at the boundary or be starting from a generalized ansatz that automatically ensures the correct boundary conditions, such as

$$\phi_N(x) = \phi_0(1 - x) + \phi_1 x \sum_{i=1}^{N} a_i u_i(x). \tag{3.40}$$

3.6.3 Generalizations to higher dimensions

Generalizations to higher dimensions are done by

- creating higher-dimensional meshes

- and providing higher-dimensional basis functions, such as pyramids centered on a mesh point.

While the basic principles remain the same, stability problems can appear at sharp corners and edges and for time-dependent geometries. The creation of appropriate meshes and basis functions is an art in itself, and an important area of industrial research. Interested students are referred to advanced courses on the subject of finite element methods

3.6.4 Nonlinear partial differential equations

The finite element method can also be applied to non-linear partial differential equations without any big changes. Let us consider a simple example

$$\phi(x)\frac{d^2\phi}{dx^2}(x) = -4\pi\rho(x) \tag{3.41}$$

Using the same ansatz, Eq. (3.32) as before and minimizing the residuals

$$g_i = \int_0^1 \left[\phi\phi''(x) + 4\pi\rho(x)\right] w_i(x)dx \tag{3.42}$$

as before we now end up with a nonlinear equation instead of a linear equation:

$$\sum_{i,j} A_{ijk}a_i a_j = b_k \tag{3.43}$$

with

$$A_{ijk} = -\int_0^1 u_i(x)u_j''(x)w_k(x)dx \tag{3.44}$$

and b_k defined as before.

The only difference between the case of linear and nonlinear partial differential equations is that the former gives a set of coupled linear equations, while the latter requires the solution of a set of coupled nonlinear equations.

Often, a Picard iteration can be used to transform the nonlinear problem into a linear one. In our case we can start with a crude guess $\phi_0(x)$ for the solution and use that guess to linearize the problem as

$$\phi_0(x)\frac{d^2\phi_1}{dx^2}(x) = -4\pi\rho(x) \tag{3.45}$$

to obtain a better solution ϕ_1. Replacing ϕ_0 by ϕ_1 an iterating the procedure by solving

$$\phi_n(x)\frac{d^2\phi_{n+1}}{dx^2}(x) = -4\pi\rho(x) \tag{3.46}$$

for ever better solutions ϕ_{n+1} we converge to the solution of the nonlinear partial differential equation by solving a series of linear partial differential equations.

3.7 Maxwell's equations

The last linear partial differential equation we will consider in this section are Maxwell's equations for the electromagnetic field. We will first calculate the field created by a single charged particle and then solve Maxwell's equations for the general case.

3.7.1 Fields due to a moving charge

The electric potential at the location $\vec{R}$ due to a single static charge q at the position $\vec{r}$ can directly be written as

$$V(\vec{R}) = \frac{q}{|\vec{r} - \vec{R}|}, \tag{3.47}$$

and the electric field calculated from it by taking the gradient $\vec{E} = -\nabla V$.

When calculating the fields due to moving charges one needs to take into account that the electromagnetic waves only propagate with the speed of light. It is thus necessary to find the retarded position

$$r_{\text{ret}} = \left|\vec{R} - \vec{r}(t_{\text{ret}})\right| \tag{3.48}$$

and time

$$t_{\text{ret}} = t - \frac{r_{\text{ret}}(t_{\text{ret}})}{c} \tag{3.49}$$

so that the distance r_{ret} of the particle at time t_{ret} was just ct_{ret}. Given the path of the particle this just requires a root solver. Next, the potential can be calculated as

$$V(\vec{R}, t) = \frac{q}{r_{\text{ret}}\left(1 - \hat{r}_{\text{ret}} \cdot \vec{v}_{\text{ret}}/c\right)} \tag{3.50}$$

with the retarded velocity given by

$$\vec{v}_{\text{ret}} = \left.\frac{d\vec{r}(t)}{dt}\right|_{t=t_{\text{ret}}} \tag{3.51}$$

The electric and magnetic field then work out as

$$\vec{E}(\vec{R}, t) = \frac{q r_{\text{ret}}}{\vec{r}_{\text{ret}}\vec{u}_{\text{ret}}}\left[\vec{u}_{\text{ret}}\left(c^2 - v_{\text{ret}}^2\right) + \vec{r}_{\text{ret}} \times \left(\vec{u}_{\text{ret}} \times \vec{a}_{\text{ret}}\right)\right] \tag{3.52}$$

$$\vec{B}(\vec{R}, t) = \hat{r}_{\text{ret}} \times \vec{E}(\vec{R}, t) \tag{3.53}$$

with

$$\vec{a}_{\text{ret}} = \left.\frac{d^2\vec{r}(t)}{dt^2}\right|_{t=t_{\text{ret}}} \tag{3.54}$$

and

$$\vec{u}_{\text{ret}} = c\hat{r}_{\text{ret}} - \vec{v}_{\text{ret}}. \tag{3.55}$$

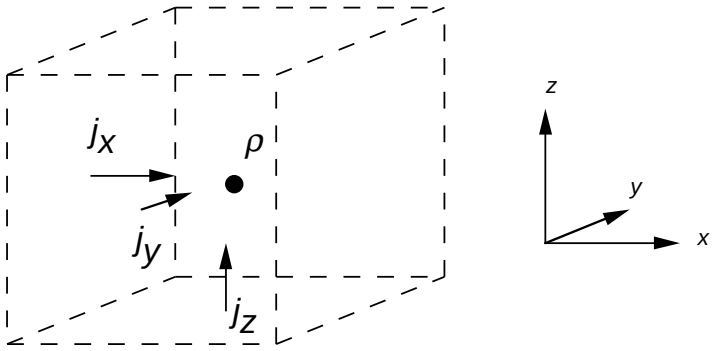

Figure 3.1: Definition of charges and currents for the Yee-Vischen algorithm

3.7.2 The Yee-Vischen algorithm

For the case of a single moving charge solving Maxwell's equation just required a root solver to determine the retarded position and time. In the general case of many particles it will be easier to directly solve Maxwell's equations, which (setting $\epsilon_0 = \mu_0 = c = 1$) read

$$\frac{\partial \vec{B}}{\partial t} = -\nabla \times \vec{E} \tag{3.56}$$

$$\frac{\partial \vec{E}}{\partial t} = \nabla \times \vec{B} - 4\pi \vec{j} \tag{3.57}$$

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot \vec{j} \tag{3.58}$$

The numerical solution starts by dividing the volume into cubes of side length Δx, as shown in figure 3.7.2 and defining by $\rho(\vec{x})$ the total charge inside the cube.

Next we need to define the currents flowing between cubes. They are most naturally defined as flowing perpendicular to the faces of the cube. Defining as $j_x(\vec{x})$ the current flowing into the box from the left, $j_y(\vec{x})$ the current from the front and $j_z(\vec{x})$ the current from the bottom we can discretize the continuity equation (3.58) using a half-step method

$$\rho(\vec{x}, t + \Delta t/2) = \rho(\vec{x}, t + \Delta t/2) - \frac{\Delta t}{\Delta x} \sum_{f=1}^{6} j_f(\vec{x}, t). \tag{3.59}$$

The currents through the faces $j_f(\vec{x}, t)$ are defined as

$$\begin{aligned} j_1(\vec{x}, t) &= -j_x(\vec{x}, t) \\ j_2(\vec{x}, t) &= -j_y(\vec{x}, t) \\ j_3(\vec{x}, t) &= -j_z(\vec{x}, t) \\ j_4(\vec{x}, t) &= j_x(\vec{x} + \Delta x \hat{e}_x, t) \\ j_5(\vec{x}, t) &= j_y(\vec{x} + \Delta x \hat{e}_y, t) \\ j_6(\vec{x}, t) &= j_z(\vec{x} + \Delta x \hat{e}_z, t). \end{aligned} \tag{3.60}$$

Be careful with the signs when implementing this.

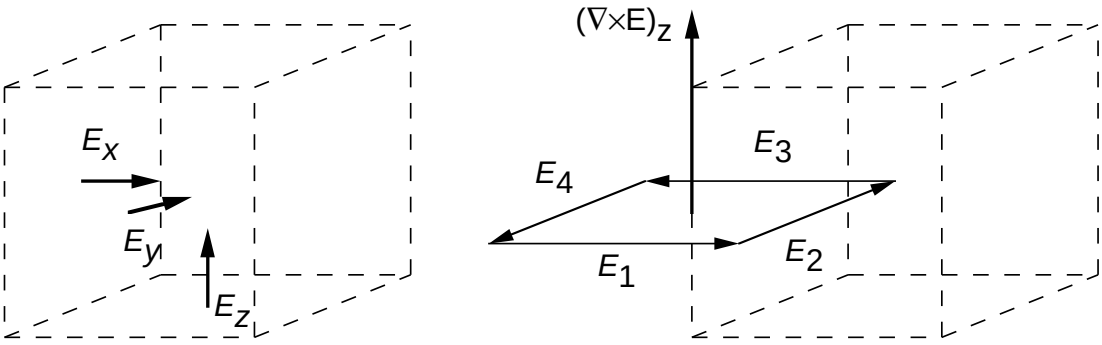

Figure 3.2: Definition of electric field and its curl for the Yee-Vischen algorithm.

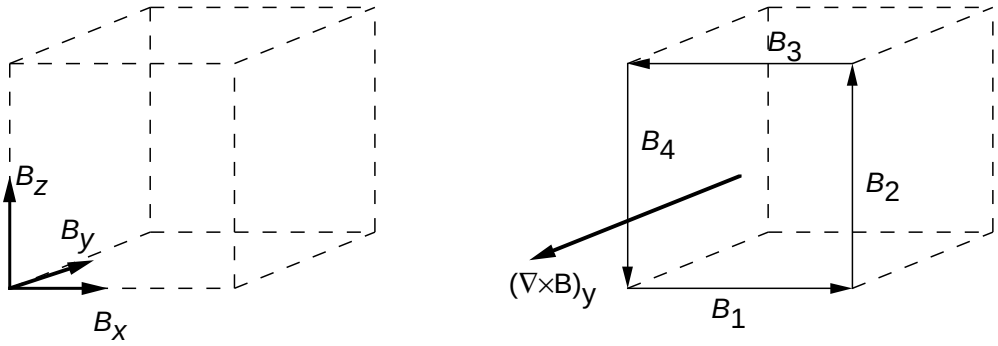

Figure 3.3: Definition of magnetic field and its curl for the Yee-Vischen algorithm.

Next we observe that equation (3.57) for the electric field $\vec{E}$ contains a term proportional to the currents j and we define the electric field also perpendicular to the faces, but offset by a half time step. The curl of the electric field, needed in equation (3.56) is then most easily defined on the edges of the cube, as shown in figure 3.7.2, again taking care of the signs when summing the electric fields through the faces around an edge.

Finally, by noting that the magnetic field term (3.56) contains terms proportional to the curl of the electric field we also define the magnetic field on the edges of the cubes, as shown in figure 3.7.2. We then obtain for the last two equations:

$$\vec{E}(\vec{x}, t + \Delta t/2) = \vec{E}(\vec{x}, t + \Delta t/2) + \frac{\Delta t}{\Delta x}\left[\sum_{e=1}^{4} \vec{B}_e(\vec{x}, t) - 4\pi\vec{j}(\vec{x}, t)\right] \quad (3.61)$$

$$\vec{B}(\vec{x}, t + \Delta t) = \vec{B}(\vec{x}, t) - \frac{\Delta t}{\Delta x}\sum_{f=1}^{4} \vec{E}_f(\vec{x}, t + \Delta t/2) \quad (3.62)$$

which are stable if $\Delta t/\Delta x \leq 1/\sqrt{3}$.

3.8 Hydrodynamics and the Navier Stokes equation

3.8.1 The Navier Stokes equation

The Navier Stokes equation is one of the most famous, if not the most famous set of partial differential equations. They describe the flow of a classical Newtonian fluid.

The first equation describing the flow of the fluid is the continuity equation, describing conservation of mass:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0 \tag{3.63}$$

where ρ is the local mass density of the fluid and $\vec{v}$ its velocity. The second equation is the famous Navier-Stokes equation describing the conservation of momentum:

$$\frac{\partial}{\partial t} (\rho \vec{v}) + \nabla \cdot \Pi = \rho \vec{g} \tag{3.64}$$

where $\vec{g}$ is the force vector of the gravitational force coupling to the mass density, and Π_{ij} is the momentum tensor

$$\Pi_{ij} = \rho v_i v_j - \Gamma_{ij} \tag{3.65}$$

with

$$\Gamma_{ij} = \eta \left[\partial_i v_j + \partial_j v_i\right] + \left[\left(\zeta - \frac{2\eta}{3}\right) \nabla \cdot \vec{v} - P\right] \delta_{ij}. \tag{3.66}$$

The constants η and ζ describe the shear and bulk viscosity of the fluid, and P is the local pressure.

The third and final equation is the energy transport equation, describing conservation of energy:

$$\frac{\partial}{\partial t} \left(\rho \epsilon + \frac{1}{2} \rho v^2\right) + \nabla \cdot \vec{j_e} = 0 \tag{3.67}$$

where ϵ is the local energy density, the energy current is defined as

$$\vec{\mathrm{j}}_\mathrm{e} = \vec{v} \left(\rho \epsilon + \frac{1}{2} \rho v^2\right) - \vec{v} \cdot \Gamma - \kappa \nabla k_B T, \tag{3.68}$$

where T is the temperature and κ the heat conductivity.

The only nonlinearity arises from the momentum tensor Π_{ij} in equation (3.65). In contrast to the linear equations studied so far, where we had nice and smoothly propagating waves with no big surprises, this nonlinearity causes the fascinating and complex phenomenon of turbulent flow.

Despite decades of research and the big importance of turbulence in engineering it is still not completely understood. Turbulence causes problems not only in engineering applications but also for the numerical solution, with all known numerical solvers becoming unstable in highly turbulent regimes. Its is then hard to distinguish the chaotic effects caused by turbulence from chaotic effects caused by an instability of the numerical solver. In fact the question of finding solutions to the Navier Stokes equations, and whether it is even possible at all, has been nominated as one of the seven millennium challenges in mathematics, and the Clay Mathematics Institute (`http:/www.claymath.org/`) has offered a prize money of one million US$ for solving the Navier-Stokes equation or for proving that they cannot be solved.

Just keep these convergence problems and the resulting unreliability of numerical solutions in mind the next time you hit a zone of turbulence when flying in an airplane, or read up on what happened to American Airlines flight AA 587.

3.8.2 Isothermal incompressible stationary flows

For the exercises we will look at a simplified problem, the special case of an isothermal (constant T) static ($\partial/\partial T = 0$) flow of an incompressible fluid (constant ρ). In this case the Navier-Stokes equations simplify to

$$\rho\vec{v}\cdot\nabla\vec{v} + \nabla P - \eta\nabla^2\vec{v} = \rho\vec{g} \tag{3.69}$$

$$\nabla\cdot\vec{v} = 0 \tag{3.70}$$

In this stationary case there are no problems with instabilities, and the Navier-Stokes equations can be solved by a linear finite-element or finite-differences method combined with a Picard-iteration for the nonlinear part.

3.8.3 Computational Fluid Dynamics (CFD)

Given the importance of solving the Navier-Stokes equation for engineering the numerical solution of these equations has become an important field of engineering called Computational Fluid Dynamics (CFD). For further details we thus refer to the special courses offered in CFD.

3.9 Solitons and the Korteveg-de Vries equation

As the final application of partial differential equations for this semester – quantum mechanics and the Schrödinger equation will be discussed in the summer semester – we will discuss the Korteveg-de Vries equations and solitons.

3.9.1 Solitons

John Scott Russell, a Scottish engineer working on boat design made a remarkable discovery in 1834:

> *I was observing the motion of a boat which was rapidly drawn along a narrow channel by a pair of horses, when the boat suddenly stopped - not so the mass of water in the channel which it had put in motion; it accumulated round the prow of the vessel in a state of violent agitation, then suddenly leaving it behind, rolled forward with great velocity, assuming the form of a large solitary elevation, a rounded, smooth and well-defined heap of water, which continued its course along the channel apparently without change of form or diminution of speed. I followed it on horseback, and overtook it still rolling on at a rate of some eight or nine miles an hour, preserving its original figure some thirty feet long and a foot to a foot and a half in height. Its height gradually diminished, and after a chase of one or two miles I lost it*

in the windings of the channel. Such, in the month of August 1834, was my first chance interview with that singular and beautiful phenomenon which I have called the Wave of Translation.

John Scott Russell's "wave of translation" is nowadays called a soliton and is a wave with special properties. It is a time-independent stationary solution of special non-linear wave equations, and remarkably, two solitions pass through each other without interacting.

Nowadays solitons are far from being just a mathematical curiosity but can be used to transport signal in specially designed glass fibers over long distances without a loss due to dispersion.

3.9.2 The Korteveg-de Vries equation

The Korteveg-de Vries (KdV) equation is famous for being the first equation found which shows soliton solutions. It is a nonlinear wave equation

$$\frac{\partial u(x,t)}{\partial t} + \epsilon u \frac{\partial u(x,t)}{\partial x} + \mu \frac{\partial^3 u(x,t)}{\partial x^3} = 0 \tag{3.71}$$

where the spreading of wave packets due to dispersion (from the third term) and the sharpening due to shock waves (from the non-linear second term) combine to lead to time-independent solitons for certain parameter values.

Let us first consider these two effects separately. First, looking at a linear wave equation with a higher order derivative

$$\frac{\partial u(x,t)}{\partial t} + c \frac{\partial u(x,t)}{\partial x} + \beta \frac{\partial^3 u(x,t)}{\partial x^3} = 0 \tag{3.72}$$

and solving it by the usual ansatz $u(x,t) = \exp(i(kx \pm \omega t)$ we find dispersion due to wave vector dependent velocities:

$$\omega = \pm ck \mp \beta k^3 \tag{3.73}$$

Any wave packet will thus spread over time.

Next let us look at the nonlinear term separately:

$$\frac{\partial u(x,t)}{\partial t} + \epsilon u \frac{\partial u(x,t)}{\partial x} = 0 \tag{3.74}$$

The amplitude dependent derivative causes taller waves to travel faster than smaller ones, thus passing them and piling up to a large shock wave, as can be seen in the Mathematica Notebook provided on the web page.

Balancing the dispersion caused by the third order derivative with the sharpening due to the nonlinear term we can obtain solitions!

3.9.3 Solving the KdV equation

The KdV equation can be solved analytically by making the ansatz $u(x,t) = f(x-ct)$. Inserting this ansatz we obtain an ordinary differential equation

$$\mu f^{(3)} + \epsilon f f' - c f' = 0, \tag{3.75}$$

which can be solved analytically in a long and cumbersome calculation, giving e.g. for $\mu = 1$ and $\epsilon = -6$:

$$u(x,t) = -\frac{c}{2} sech^2 \left[\frac{1}{2}\sqrt{c}\,(x - ct - x_0)\right] \tag{3.76}$$

In this course we are more interested in numerical solutions, and proceed to solve the KdV equation by a finite difference method

$$\begin{aligned} u(x_i, t+\Delta t) &= u(x_i, t-\Delta t) \\ &\quad -\frac{\epsilon}{3}\frac{\Delta t}{\Delta x}\left[u(x_{i+1},t) + u(x_i,t) + u(x_{i-1},t)\right]\left[u(x_{i+1},t) - u(x_{i-1},t)\right] \\ &\quad -\mu\frac{\Delta t}{\Delta x^3}\left[u(x_{i+2},t) + 2u(x_{i+1},t) - 2u(x_{i-1},t) - u(x_{i-2},t)\right]. \end{aligned} \tag{3.77}$$

Since this integrator requires the wave at two previous time steps we start with an initial step of

$$\begin{aligned} u(x_i, t_0+\Delta t) &= u(x_i, t_0) \\ &\quad -\frac{\epsilon}{6}\frac{\Delta t}{\Delta x}\left[u(x_{i+1},t) + u(x_i,t) + u(x_{i-1},t)\right]\left[u(x_{i+1},t) - u(x_{i-1},t)\right] \\ &\quad -\frac{\mu}{2}\frac{\Delta t}{\Delta x^3}\left[u(x_{i+2},t) + 2u(x_{i+1},t) - 2u(x_{i-1},t) - u(x_{i-2},t)\right] \end{aligned} \tag{3.78}$$

This integrator is stable for

$$\frac{\Delta t}{\Delta x}\left[|\varepsilon u| + 4\frac{|\mu|}{\Delta x^2}\right] \leq 1 \tag{3.79}$$

Note that as in the case of the heat equation, a progressive decrease of space steps or even of space and time steps by the same factor will lead to instabilities!

Using this integrator, also provided on the web page as a Mathematica Notebook you will be able to observe:

- The decay of a wave due to dispersion
- The creation of shock waves due to the nonlinearity
- The decay of a step into solitons
- The crossing of two solitons

Chapter 4

The classical N-body problem

4.1 Introduction

In this chapter we will discuss algorithms for classical N-body problems, whose length scales span many orders of magnitudes

- the universe ($\approx 10^{26}$m)
- galaxy clusters ($\approx 10^{24}$m)
- galaxies ($\approx 10^{21}$m)
- clusters of stars ($\approx 10^{18}$m)
- solar systems ($\approx 10^{13}$m)
- stellar dynamics ($\approx 10^{9}$m)
- climate modeling ($\approx 10^{6}$m)
- gases, liquids and plasmas in technical applications ($\approx 10^{-3} \dots 10^{2}$m)

On smaller length scales quantum effects become important. We will deal with them later.

The classical N-body problem is defined by the following system of ordinary differential equations:

$$\begin{aligned} m_i \frac{d\vec{v}_i}{dt} &= \vec{F}_i = -\nabla_i V(\vec{x}_1, \dots, \vec{x}_N) \\ \frac{d\vec{x}_i}{dt} &= \vec{v}_i, \end{aligned} \tag{4.1}$$

where m_i, $\vec{v}_i$ and $\vec{x}_i$ are the mass, velocity and position of the i-the particle.

The potential $V(\vec{x}_1, \dots, \vec{x}_N)$ is often the sum of an external potential and a two-body interaction potential:

$$V(\vec{x}_1, \dots, \vec{x}_N) = \sum_i V_{\text{ext}}(\vec{x}_i) + \sum_{i<j} U_{ij}(|\vec{x}_i - \vec{x}_j|) \tag{4.2}$$

The special form $U(|\vec{x}_i - \vec{x}_j|)$ of the two-body potential follows from translational and rotational symmetry.

4.2 Applications

There are many different forms of the potential U:

1. In astrophysical problems gravity is usually sufficient, except in dense plasmas, interiors of stars and close to black holes:

$$U_{ij}^{(\text{gravity})}(r) = -G\frac{m_i m_j}{r}. \tag{4.3}$$

2. The simplest model for non-ideal gases are hard spheres with radius a_i:

$$U_{ij}^{(\text{hard sphere})}(r) = \begin{cases} 0 & \text{for } r >= a_i + a_j \\ \infty & \text{for } r < a_i + a_j \end{cases} \tag{4.4}$$

3. Covalent crystals and liquids can be modeled by the Lennard-Jones potential

$$U_{ij}^{(\text{LJ})}(r) = 4\epsilon_{ij}\left[(\frac{\sigma}{r}^{12}) - (\frac{\sigma}{r})^6\right]. \tag{4.5}$$

   The r^{-6}-term describes the correct asymptotic behavior of the covalent van der Waals forces. The r^{-12}-term models the hard core repulsion between atoms. The special form r^{-12} is chosen to allow for a fast and efficient calculation as square of the r^{-6} term. Parameters for liquid argon are $\epsilon = 1.65 \times 10^{-21}$J and $\sigma = 3.4 \times 10^{-10}$m.

4. In ionic crystals and molten salts the electrostatic forces are dominant:

$$U_{ij}^{(\text{ionic})}(r) = b_{ij}r^{-n} + e^2\frac{Z_i Z_j}{r}, \tag{4.6}$$

   where Z_i and Z_j are the formal charges of the ions.

5. The simulation of large biomolecules such as proteins or even DNA is a big challenge. For non-bonded atoms often the 1-6-12 potential, a combination of Lennard-Jones and electrostatic potential is used:

$$U_{ij}^{(1-6-12)}(r) = e^2\frac{Z_i Z_j}{r} + 4\epsilon_{ij}\left[(\frac{\sigma}{r}^{12}) - (\frac{\sigma}{r})^6\right]. \tag{4.7}$$

   For bonded atoms there are two ways to model the bonding. Either the distances between two atoms can be fixed, or the bonding can be described by a harmonic oscillator:

$$U_{ij}^{(\text{bond})}(r) = \frac{1}{2}K_{ij}(r - b_{ij})^2. \tag{4.8}$$

   The modeling of fixed angles between chemical bonds (like in water molecules) is a slightly more complex problem. Again, either the angle can be fixed, or modeled by a harmonic oscillator in the angle θ. Note that the angle θ is determined by the location of three atoms, and that this is thus a *three-body-interaction*! Students who are interested in such biomolecules are referred to the research group of Prof. van Gunsteren in the chemistry department.

6. More complex potentials are used in the simulation of dense plasmas and of collisions of heavy atomic nuclei.

7. The Car-Parrinello method combines a classical simulation of the molecular dynamics of the motion of atomic nuclei with a quantum chemical ab-initio calculation of the forces due to electronic degrees of freedom. This gives more accurate forces than a Lennard-Jones potential but is possible only on rather small systems due to the large computational requirements. If you are interested in the Car-Parrinello method consider the research group of Prof. Parrinello in Lugano.

4.3 Solving the many-body problem

The classical many-body problem can be tackled with the same numerical methods that we used for the few-body problems, but we will encounter several additional difficulties, such as

- the question of boundary conditions
- measuring thermodynamic quantities such as pressure
- performing simulations at constant temperature or pressure instead of constant energy or volume
- reducing the scaling of the force calculation for long-range forces from $O(N^2)$ to $O(N \ln N)$
- overcoming the slowing down of simulations at phase transitions

4.4 Boundary conditions

Open boundary conditions are natural for simulations of solar systems or for collisions of galaxies, molecules or atomic nuclei. For simulations of crystals, liquids or gases on the other hand, effects from open boundaries are not desired, except for the investigation of surface effects. For these systems periodic boundary conditions are better. As we discussed earlier, they remove all boundary effects.

In the calculation of forces between two particle all periodic images of the simulation volume have to be taken into account. For short range forces, like a Lennard-Jones force, the "minimum image" is the method of choice. Here the distance between a particle and the nearest of all periodic images of a second particle is chosen for the calculation of the forces between the two particles.

For long range forces on the other hand (forces that as r^{-d} or slower) the minimum image method is not a good approximation because of large finite size effects. Then the forces caused by all the periodic images of the second particle have to be summed over. The electrostatic potential acting on a particle caused by other particles with charge q_i at sites $\vec{r}_i$ is

$$\Phi_p = \sum_{\vec{n}} \sum_i \frac{q_i}{|\vec{r}_{\vec{n}} - \vec{r}_i|}, \tag{4.9}$$

where $\vec{n}$ is an integer vector denoting the periodic translations of the root cell and $\vec{r}_{\vec{n}}$ is the position of the particle in the corresponding image of the root cell.

This direct summation converges very slowly. It can be calculated faster by the Ewald summation technique[1], which replaces the sum by two faster converging sums:

$$\begin{aligned}\Phi_p &= \sum_{\vec{n}}\sum_i q_i \frac{\mathrm{erfc}(\alpha|\vec{r}_{\vec{n}} - \vec{r}_i|)}{|\vec{r}_{\vec{n}} - \vec{r}_i|} + \\ &+ \frac{1}{\pi L}\sum_i \sum_{\vec{h}\neq 0} q_i \exp\left(\frac{-\pi|h|^2}{\alpha L^2}\right)\cos\left(\frac{2\pi}{L}\vec{h}\cdot(\vec{r}_o - \vec{r}_i)\right). \end{aligned} \tag{4.10}$$

In this sum the $\vec{h}$ are integer reciprocal lattice vectors. The parameter α is arbitrary and can be chosen to optimize convergence.

Still the summation is time-consuming. Typically one tabulates the differences between Ewald sums and minimum image values on a grid laid over the simulation cell and interpolates for distances between the grid points. For details we refer to the detailed discussion in M.J. Sangster and M. Dixon, Adv. in Physics **25**, 247 (1976).

4.5 Molecular dynamics simulations of gases, liquids and crystals

4.5.1 Ergodicity, initial conditions and equilibration

In scattering problems or in the simulation of cosmological evolution the initial conditions are usually given. The simulation then follows the time evolution of these initial conditions. In molecular dynamics simulations on the other hand one is interested in thermodynamic averages $\langle A\rangle$. In an *ergodic* system the phase space average is equivalent to the time average:

$$\langle A\rangle := \frac{\int A(\Gamma)P[\Gamma]d\Gamma}{\int P[\Gamma]d\Gamma} = \lim_{\tau\to\infty}\frac{1}{\tau}\int_0^\tau A(t)dt. \tag{4.11}$$

Initial conditions are best chosen as a regular crystal lattice. The velocities are picked randomly for each component, according to a Maxwell distribution

$$P[v_\alpha] \propto \exp\left(\frac{-mv_\alpha^2}{2k_BT}\right). \tag{4.12}$$

Finally the velocities are all rescaled by a constant factor to obtain the desired total energy.

An important issue is that the system has to be equilibrated (thermalized) for some time before thermal equilibrium is reached and measurements can be started. This thermalization time is best determined by observing time series of physical observables, such as the kinetic energy (temperature) or other quantities of interest.

[1] P.P. Ewald, Ann. Physik **64**, 253 (1921).

4.5.2 Measurements

A simple measurement is the self-diffusion constant D. In a liquid or gaseous system it can be determined from the time dependence of the positions:

$$\Delta^2(t) = \frac{1}{N}\sum_{i=1}^{N}[\vec{r}_i(t) - \vec{r}_i(0)]^2 = 2dDt + \Delta_0^2 \tag{4.13}$$

In a crystal the atoms remain at the same location in the lattice and thus $D = 0$. A measurement of D is one way to observe melting of a crystal.

Another quantity that is easy to measure is the mean kinetic energy

$$\langle E_k \rangle = \frac{1}{2}\langle \sum_{i=1}^{N} m_i \vec{v}_i^2 \rangle. \tag{4.14}$$

$\langle E_k \rangle$ is proportional to the mean temperature

$$\langle E_k \rangle = \frac{G}{2} k_B T, \tag{4.15}$$

where $G = d(N-1) \approx dN$ is the number of degrees of freedom.

In a system with fixed boundaries the particles are reflected at the boundaries. The pressure P is just the force per area acting on the boundary walls of the system In the case of periodic boundary conditions there are no walls. The pressure P can then be measured using the following equation, derived from the virial theorem:

$$P = \frac{N k_B T}{V} + \frac{1}{dV}\sum_{i<j} \vec{r}_{ij} \cdot \vec{F}_{ij}(t), \tag{4.16}$$

where $\vec{F}_{ij}$ denotes the force between particles i and j and $\vec{r}_{ij}$ is their distance.

The first term of equation (4.16) is the kinetic pressure, due to the kinetic energy of the particles. This term alone gives the ideal gas law. The second term is the pressure (force per area) due to the interaction forces.

More information can usually be extracted from the pair correlation function

$$g(\vec{r}) = \frac{1}{\rho(N-1)} \left\langle \sum_{i \neq j} \delta(\vec{r} + \vec{r}_i - \vec{r}_j) \right\rangle \tag{4.17}$$

or its Fourier transform, the static structure factor $S(\vec{k})$

$$g(\vec{r}) - 1 = \frac{1}{(2\pi)^d \rho} \int [S(\vec{k}) - 1] \exp(i\vec{k} \cdot \vec{r}) d\vec{k} \tag{4.18}$$

$$S(\vec{k}) - 1 = \rho \int [g(\vec{r}) - 1] \exp(-i\vec{k} \cdot \vec{r}) d\vec{r} \tag{4.19}$$

If the angular dependence is of no interest, a radial pair correlation function

$$g(r) = \frac{1}{4\pi} \int g(\vec{r}) \sin\theta d\theta d\phi \tag{4.20}$$

and corresponding structure factor

$$S(k) = 4\pi\rho \int_0^\infty \frac{\sin kr}{kr}[g(r) - 1]r^2 dr \tag{4.21}$$

can be used instead.

This structure factor can be measured in X-ray or neutron scattering experiments. In a perfect crystal the structure factor shows sharp δ-function like Bragg peaks and a periodic long range structure in $g(\vec{r})$. Liquids still show broad maxima at distances of nearest neighbors, second nearest neighbors, etc., but these features decay rapidly with distance.

The specific heat at constant volume c_V can in principle be calculated as a temperature derivative of the internal energy. Since such numerical derivatives are numerically unstable the preferred method is a calculation from the energy fluctuations

$$c_v = \frac{\langle E^2 \rangle - \langle E \rangle^2}{k_B T^2}. \tag{4.22}$$

4.5.3 Simulations at constant energy

The equations of motion of a disspiationless system conserve the total energy and the simulation is thus done in the microcanonical ensemble. Discretization of the time evolution however introduces errors in the energy conservation, and as a consequence the total energy will slowly change over time. To remain in the microcanonical ensemble energy corrections are necessary from time to time. These are best done by a rescaling of all the velocities with a constant factor. The equations are easy to derive and will not be listed here.

4.5.4 Constant temperature

The canonical ensemble at constant temperature is usually of greater relevance than the microcanonical ensemble at constant energy. The crudest, ad-hoc method for obtaining constant temperature is a rescaling like we discussed for constant energy. This time however we want rescale the velocities to achieve a constant kinetic energy and thus, by equation (4.15) constant temperature. Again the equations can easily be derived.

A better method is the Nosé-Hoover thermostat. In this algorithm the system is coupled reversibly to a heat bath by a friction term η:

$$\begin{aligned} m_i \frac{d\vec{v}_i}{dt} &= \vec{F}_i - \eta \vec{v}_i \\ \frac{d\vec{r}_i}{dt} &= \vec{v}_i \end{aligned} \tag{4.23}$$

$$\tag{4.24}$$

The friction term η is chosen such that constant temperature is achieved on average. We want this term to heat up the system if the temperature is too low and to cool it down if the temperature is too high. One way of doing this is by setting

$$\frac{d\eta}{dt} = \frac{1}{m_s}\left(E_k - \frac{1}{2}Gk_B T\right), \tag{4.25}$$

where m_s is the coupling constant to the heat bath.

4.5.5 Constant pressure

Until now we always worked at fixed volume. To perform simulations at constant pressure we need to allow the volume to change. This can be done by rescaling the coordinates with the linear size L of the system:

$$\vec{r} = L\vec{x}. \tag{4.26}$$

The volume of the system is denoted by $\Omega = L^D$. We extend the Lagrangian by including an external pressure P_0 and an inertia M for pressure changes (e.g. the mass of a piston):

$$\mathcal{L} = \sum_{i=1}^{N} \frac{m_i}{2} L^2 \left(\frac{d\vec{x}_i}{dt}\right)^2 - \sum_{i<j} V(L(\vec{x}_i - \vec{x}_j)) + \frac{M}{2}\left(\frac{d\Omega}{dt}\right)^2 + P_0\Omega \tag{4.27}$$

The Euler equations applied to above Lagrangian give the equations of motion:

$$\begin{aligned} \frac{d^2\vec{x}_i}{dt^2} &= \frac{1}{m_i L}\vec{F}_i - \frac{2}{D\Omega}\frac{d\Omega}{dt}\frac{d\vec{x}_i}{dt} \\ \frac{d^2\Omega}{dt^2} &= \frac{P - P_0}{M}, \end{aligned} \tag{4.28}$$

where P turns out to be just the pressure defined in equation (4.16). These equations of motion are integrated with generalizations of the Verlet algorithm.

Generalizations of this algorithm allow changes not only of the total volume but also of the shape of the simulation volume.

4.6 Scaling with system size

The time intensive part of a classical N-body simulation is the calculation of the forces. The updating of positions and velocities according to the forces is rather fast and scales linearly with the number of particles N.

For short range forces the number of particles within the interaction range is limited, and the calculation of the forces, while it might still be a formidable task, scales with $O(N)$ and thus poses no big problems.

Rapidly decaying potentials, like the Lennard-Jones potential can be cut off at a distance r_c. The error thus introduced into the estimates for quantities like the pressure can be estimated from equations (4.16) using equation (4.17) as:

$$\Delta P = -\frac{2\pi\rho^2}{3}\int_{r_c}^{\infty} \frac{\partial V}{\partial r} g(r) r^3 dr \tag{4.29}$$

where a common two-body potential $V(r)$ between all particle pairs was assumed. If $V(R)$ decays faster than r^{-3} (in general r^{-d}, where d is the dimensionality) this correction becomes small as r_c is increased.

Long range forces, like Coulomb forces or gravity, on the other hand, pose a big problem. No finite cut-off may be introduced without incurring substantial errors.