A Critical Study of the Finite Difference and Finite Element Methods for the Time Dependent Schrödinger Equation

Simen Kvaal

Thesis Submitted for the Degree
of Candidatus Scientiarum

Department of Physics
University of Oslo

March 2004

Preface

This thesis is perhaps a bit lengthy compared to the standards of a *cand. scient.* degree (or Master Degree as it will be called in the future in order to comply with international standards). It is however aimed at a broad audience; from my fellow physics students to mathematicians and other non-physicists that may have interests in the area. I have tried to cut away material in order to make it a little bit shorter, and some less central material is moved to appendices for the specially interested readers.

The title of the thesis pretty much describes the aim of this *cand. scient.* project. The finite difference and finite element methods are two widely used approaches to solving partial differential equations. Traditionally the finite element method has been reserved for engineering projects and not that much in basic research fields such as atomic physics where the finite difference method is the main method. One reason may be that finite element methods are very complicated to implement and that they utilize a wide range of complicated numerical tools, such as sparse matrices and iterative solvers for linear equations. For this reason finite element solvers are usually expensive commercial products (such as FemLab, see Ref. [1]) whose operation hides the numerical details for the user, an approach that in a way makes scientists feel that they lose control of the situation.

Some work has been done on the Schrödinger equation with finite element methods early in the eighties and nineties, see for example Refs. [2–4], but for some reason the development has seemed to stagnate. One reason might be the above mentioned complexity in implementation. There is a huge threshold to climb if one wants to generalize a simple formulation which can be coded in a few hundred lines, experiment with different element types and so on. Fortunately, the programming library Diffpack is tailored for scientific problems, making available very powerful and flexible class hierarchies, interfaces and visualization support.

The newly initiated Centre for Mathematics with Applications (CMA) tries to join the forces of mathematics, physics and informatics among others, and a thorough yet basic exposition into quantum mechanics might be in the spirit of such collaborative work. Chapters 1 and 2 contain the fundamentals of quantum mechanics, and physicists may skip these chapters and go directly to chapter 3, in which I discuss more specific physical problems. Others may find the preceding chapters illuminating and interesting. A lot of work has been put into making an understandable discussion aimed at practitioners of natural sciences with a foundation in mathematics. It places itself in the middle of an intermediate course of quantum mechanics and a guide for mathematically trained people with an interest in physics.

Chapter 4 deals with ordinary and partial differential equations and numerical methods. A recipe-based approach is taken, retaining a touch of rigor and analysis along the way. Chapter 5 is a short review of numerical methods for problems from linear algebra; more specifically square linear systems of equations and eigenvalue problems. We also discuss the implementations of these methods in Diffpack, at the same time giving an introduction to this powerful programming library for partial differential equations and finite element methods.

Chapter 6 turns to quantum mechanical eigenvalue problems and discusses their

importance and their applications in time dependent problems, which are presented in chapter 7. Some numerical experiments are performed and analyzed, yielding an insight into the behavior of both the finite difference and finite element discretizations of the eigenvalue problem. Chapter 7 is the pinnacle of the thesis, around which all other material build up. We discuss the solution of the time dependent Schrödinger equation for a two-dimensional hydrogen atom with a laser field perturbation; a system which is both numerically, physically and conceptually interesting. The finite element implementation is flexible and allows for generalizations such as different gauges, different geometries, new time integration methods and so on.

Chapter 8 concludes and sums up the contents of the thesis, discussing important results and ideas for further work. There are a few appendices; the first one picks up some technical details of calculations and formalism, while appendix B contains complete program listings for every program written for the thesis.

As is often the case with the work on a *cand. scient.*-degree the creative process accelerates enormously towards the final days before deadline. Ideas pop up, intensive research is being done and many good (and some bad) ideas emerge. Alas, one cannot follow each and every idea. Rather late in the work with the thesis I realized that the mass matrix of finite element discretizations is in fact (obviously!) positive definite. This allows for converting every generalized eigenvalue problem arising from finite element discretizations into a standard eigenvalue problem by means of the Cholesky decomposition. Furthermore, incorporating this into the `HydroEigen` class definition should only amount to another hundred lines of code, but with proper testing rituals and debugging sessions it would require another week of work while not adding to the flexibility of the program. On the other hand, this optimization would allow use of much faster eigenvalue searching, which in turn would mean more numerical experiments and more detailed results.

There are several computer programs written for this thesis in addition to essential downloadable documentation and texts. For this reason I have created a web page, see Ref. [5], giving access to all source code, reports, animations, this text and more.

I would like to thank my supervisors Prof. Morten Hjorth-Jensen and Prof. Hans Petter Langtangen for invaluable support and guidance during the work with the thesis. Morten has spent quite a few hours of valuable time on interesting discussions, providing constructive criticism and proofreading of the text, and making available his deep and thorough insight into both numerical methods and quantum mechanics.

Hans Petter was co-supervisor and is responsible for this thesis being possible at all with his thorough knowledge of Diffpack, his offering summer holiday work relevant to my thesis and his constant encouragement and confidence in my abilities as a student and programmer.

I would also like to thank Prof. Ragnar Winther at CMA for valuable help with section 4.6 on stability analysis of finite element schemes. In addition, 1st Am. Per Christian Moan (also at CMA) has provided interesting discussions on gauge invariance and time integration.

During moments of frustration, slight despair and writer's block my girlfriend, my family and my friends have (as always!) put up with me. They have provided immense encouragement and support for which I am truly thankful. I dedicate this work to all of you, for as they say, no man is an island.

Oslo, March 2004
Simen Kvaal

Contents

Chapter 1

A Brief Introduction to Quantum Mechanics

We will present an introduction to the quantum mechanics of a single particle in this and the subsequent two chapters; its background, formalism and application on simple systems and examples. It will serve as an introduction to the real topic of this thesis, which is the numerical solution of the time dependent Schrödinger equation (which we will introduce later).

In this chapter, we will start with some basic classical mechanics to provide means to compare quantum physical concepts to "ordinary" Newtonian concepts, such as measurements and observables, equations of motion and probabilistic features.

Then we proceed with the postulates of quantum mechanics. By "postulates" we shall mean concepts that are not obtainable by means of mathematics or deduction from other areas of physics, but on the other hand are necessary and sufficient for developing all quantum mechanical results.

1.1 A Quick Look at Classical Mechanics

A particle described by classical mechanics behaves in a perfectly ordinary way. That is, it obeys the everyday mechanics of golf balls, planets and coffee-cups, also known as Newtonian mechanics.

The central force behind Newtonian mechanics is *Newton's second law*. This is the well-known differential equation stating the relation between a particle's acceleration and the force exerted on it by its surroundings:[1]

$$m\ddot{\boldsymbol{x}} = \boldsymbol{F}. \tag{1.1}$$

Although Newtonian mechanics is seemingly a well-established theory in this form, mathematicians and physicists have developed other equivalent and in some respects more complicated ways of formulating it. On the other hand, the reformulations yield deep insight into classical mechanics. In fact, the mathematical framework of classical mechanics is much deeper than we will ever be able to present in even a lengthy exposition, and it represents a huge area of mathematics.

For an excellent account of classical mechanics, see Ref. [6].

We will give a brief introduction to the Hamiltonian formulation of classical mechanics. Quantum mechanics may be built on this formalism, and the similarities and

[1] Note the dot-notation for the time derivative: n dots means n differentiations with respect to time.

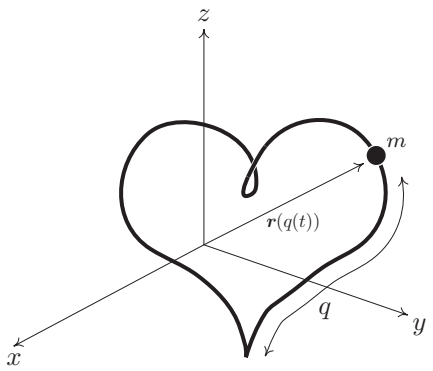

Figure 1.1: Particle in three dimensions with constraints

parallels are quite instructive. (There is also another and equivalent way of introducing quantum mechanics through the Lagrangian formulation of classical mechanics, see for example Ref. [7].)

Let us first introduce the *material system*: A system consisting of n point-shaped particles moving in D-dimensional space. By point-shaped we mean that its configuration is completely determined by a single position in D-space, and that each particle has a non-zero mass m_i. The configuration of a material system is then specified by nD real numbers $q = (q_1, q_2, \ldots, q_{nD})$, the set of which constitutes the so-called configuration space.

To specify the *state* of the material system however, we need to specify the motion of the system at an instant, and this is done through the momenta $p = (p_1, p_2, \ldots, p_{nD})$. To put it simple, the momenta represents the velocity of each particle in each direction in D-space.

The configuration and momentum of the system are not necessarily the usual cartesian coordinates (x_i, y_i and z_i, $i = 1 \ldots n$) and momenta ($m_i v_{x,i}$, $m_i v_{y,i}$ and $m_i v_{z,i}$), but rather so-called *generalized coordinates and momenta*. They are also called *canonical variables* and p and q are called *canonical conjugates* of each other.

For a single particle moving in three-dimensional space, there are initially three coordinates x, y and z in addition to the momenta p_x, p_y and p_z, but if we impose constraints on the motion, such as forcing the particle to move on an arbitrary shaped wire like in Fig. 1.1, the number may be reduced. In this case we have only one generalized coordinate q. The cartesian coordinates are then given as a function of q.

We will not state the general definition of generalized momenta. For the purpose of the applications in this thesis it suffices to think of the p_i as the velocity of q_i.

However, the momenta are defined in such a way that the equations of motion in Hamiltonian mechanics equivalent to Newton's second law are given by *Hamilton's equations of motion:*

$$\dot{q}_i = \frac{\partial H}{\partial p_i} \tag{1.2}$$

$$\dot{p}_i = -\frac{\partial H}{\partial q_i} \tag{1.2'}$$

The so-called Hamiltonian is a function of the individual coordinates and momenta and possibly explicitly of time, viz.,

$$H = H(q, p, t).$$

The Hamiltonian may usually be thought of as the system's energy function, i.e., its total energy. In some systems we consider this is however not the case. We will

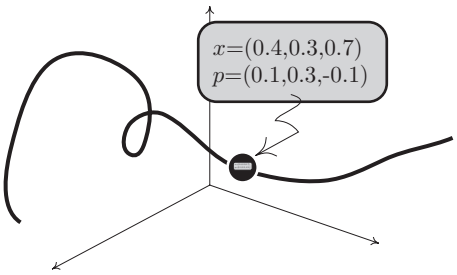

Figure 1.2: A classical particle and measuring its state

introduce electromagnetic forces to the system, and this alters the Hamiltonian and the definition of the canonical momenta p_i.

If H is the total energy of the system, and if it is explicitly independent of time, we may easily derive an important conservation law:

$$\frac{\mathrm{d}H}{\mathrm{d}t} = \frac{\partial H}{\partial t} + \sum_i \frac{\partial H}{\partial q_i}\dot{q}_i + \sum_i \frac{\partial H}{\partial p_i}\dot{p}_i = \frac{\partial H}{\partial t} + \sum_i (-\dot{p}_i\dot{q}_i + \dot{q}_i\dot{p}_i) = \frac{\partial H}{\partial t}.$$

Thus, the energy is conserved for a material system if the Hamiltonian does not explicitly depend on time. We shall see examples of this in chapter 2 when we discuss simple quantum mechanical systems and their classical analogies.

An important special case of Hamiltonian systems are systems composed of a single particle with mass m moving in three-dimensional space and under influence of an external potential $V(\boldsymbol{x})$, where $\boldsymbol{x}$ is the cartesian coordinates of the particle, viz.,

$$H(\boldsymbol{x}, \boldsymbol{p}, t) = T + V = \frac{\boldsymbol{p}^2}{2m} + V(\boldsymbol{x}, t).$$

Here, T is the total kinetic energy. Writing out Hamilton's equations of motion yields

$$\begin{aligned} \dot{\boldsymbol{x}} &= \frac{\boldsymbol{p}}{m} \\ \text{and} \quad \dot{\boldsymbol{p}} &= -\nabla V(\boldsymbol{x}, t), \end{aligned}$$

which is exactly Newton's second law. (Recall that in a potential field the force is given by $-\nabla V$.)

We will not elaborate any further on Hamilton's equations until chapter 2, but only note that since they are a set of ordinary differential equations governing the time evolution of a classical system, classical dynamics is *perfectly deterministic*, in the sense that given initial conditions $\boldsymbol{x}(0)$ and $\boldsymbol{p}(0)$ for the location and momentum of the particle, one may (at least in principle) calculate its everlasting trajectory through configuration space by solving these equations.

Furthermore, the concept of ideal measurements in classical mechanics is rather simple. If we have some dynamical quantity $\omega(\boldsymbol{x}, \boldsymbol{p})$ it may be calculated at all times using the coordinates and momenta at that time. There is no interference with the actual system. We can picture the particle as if it had a display attached, with $\boldsymbol{x}$ and $\boldsymbol{p}$ continuously updated and displayed for us to read, as in Fig. 1.2. Functions ω of the canonical coordinates and momenta are called classical *observables*. A measurable quantity can only be defined in terms of the state of the system, hence it must be such a function. On the other hand, $\boldsymbol{x}$ and $\boldsymbol{p}$ are measurable so that any function of these also must be an observable.

Newtonian mechanics is in most respects very intuitive and agrees with our ordinary way of viewing things around us. Things exist in a definite place, moving along definite paths with definite velocities. Indeed, this mechanistic viewpoint was securely founded

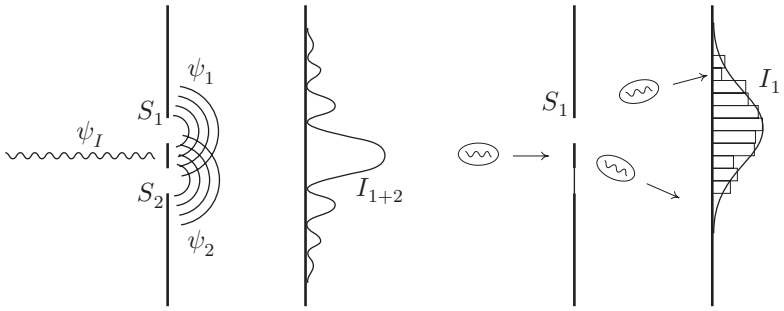

Figure 1.3: Schematic view of double-slit experiment (left) and discovery of photons (right)

already in the 18th century, when Pierre Simon De Laplace (1749–1827) proposed his deterministic world view, see Ref. [8]:

> *"We may regard the present state of the universe as the effect of its past and the cause of its future. An intellect which at any given moment knew all of the forces that animate nature and the mutual positions of the beings that compose it, if this intellect were vast enough to submit the data to analysis, could condense into a single formula the movement of the greatest bodies of the universe and that of the lightest atom; for such an intellect nothing could be uncertain and the future just like the past would be present before its eyes."*

This view was however profoundly shaken through the advent of quantum mechanics. One simply cannot measure the positions and velocities with infinite accuracy. In addition, the new science of *chaos* makes the Laplacian determinism somewhat hollow. If we consider Hamilton's equations of motion that govern many physical systems, they turn out to exhibit chaotic behavior, i.e., small variations in initial conditions increase exponentially with time, making long-term predictions of the system's configuration impossible, see Ref. [9].

1.2 Why Quantum Mechanics?

We will not go into the historical reasons of how and why quantum mechanics came to be, although this is very interesting in its own respect. Quantum mechanics arose in the first decades of the 20th century when modern experiments and measurements contradicting the classical theories created a crisis in the physical communities. In Refs. [10, 11] excellent and entertaining accounts of these matters are given.

We will instead make an illustration that is quite popular, explaining why we need quantum mechanics and at the same time introducing some of the concepts, making it easier to follow the introduction of the postulates of quantum mechanics in the next section.

Imagine that we direct a beam of monochromatic light described by a wave ψ_I onto a plate with two slits S_1 and S_2. At some distance behind the plate we place a photosensitive screen. The experimental setup of this double-slit experiment is shown in Fig. 1.3. Some of the light will pass through the plate. The outgoing waves ψ_1 and ψ_2 coming from S_1 and S_2, respectively, will interfere with each other, producing an interference pattern I_{1+2} at the screen. (The subscript indicates what the slits are open.)

This is very well understood if we assume the wave-nature of electromagnetic fields; an assumption that has been made for more than a century, after Maxwell proposed

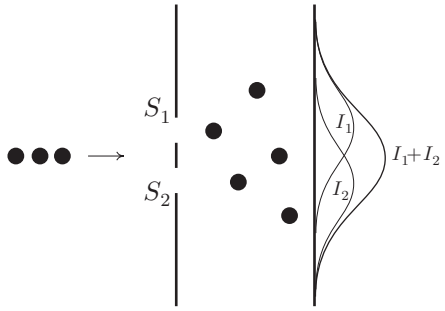

Figure 1.4: Classical particles scattered by double-slits, and their combined arrival distributions

his famous equations that show that light propagates as waves in a way very similar to waves on the surface of water.[2] In fact we may imagine an analogous experiment with water-waves and observe the results above.

Suppose now that we close S_2 and lower the intensity of the beam ψ_I. Then we will notice that the light no longer arrives in a continuous fashion at the screen; we observe small bursts of light. We have made the monumental discovery that light actually comes in small bundles – they have a particle nature. If we observe a large number of these so-called photons and make a histogram of their arrival-coordinate x, we will of course produce the pattern I_1, see Fig. 1.3.

The fact that light turned out to be particles was surprising, but not in direct opposition to classical physics. Physicists are used to the idea that phenomena appearing continuous in reality are composed of small and discrete components. A drop of water is for example composed of tiny molecules. But classical physics was anyway in for a surprise.

Experimentally we may find that each photon carries the same energy $E = h\nu$, where ν is the frequency of the light waves. Furthermore, we may find that they all share the same momentum $p = h/\lambda$, where λ is the wavelength, related to the frequency by $\lambda\nu = c$, the speed of light. The number h is called Planck's constant, and

$$h \approx 6.626\,068\,076 \cdot 10^{-34}\ \text{Js},$$

a very small number.[3]

What happens when we open S_2 again? If the photons are particles, they will pass either S_1 or S_2, creating the pattern $I_1 + I_2$. The photons passing S_1 will contribute to I_1, and the photons passing S_2 will contribute to I_2. This picture of the photons as classical particles is shown in Fig. 1.4. Instead, we observe that the pattern is indeed very *different* from $I_1 + I_2$, even with the very low intensity, which can only mean that the photons cannot pass through one single slit in a well-defined way. In some strange way they must pass both! It turns out that the classical particle picture has some flaws after all.

The same experiment may be used with *electrons;* physical objects that until the 1920s were solely regarded as particles in the classical sense. But as the joint diffraction patterns $I_1 + I_2$ fails to reproduce I_{1+2} in this case as well, we must abandon the Newtonian particle nature of the electrons; at least when they pass the slits S_1 and S_2. In fact, electron diffraction experiments have been carried out with an electron intensity as low as one electron per minute, see Ref. [13].

In 1926 De Broglie proposed a daring hypothesis (see Refs. [10, 14]), stating that

[2] See section 1.6.

[3] The uncertainty of the cited h lies in the two last digits. Cited from Ref. [12].

any physical object can be ascribed a wavelength λ given by the relation

$$\lambda = \frac{h}{p}, \tag{1.3}$$

which is the same expression as for the wavelength of photons, i.e.,

$$\lambda = \frac{c}{\nu} = \frac{hc}{h\nu} = \frac{hc}{E} = \frac{hc}{pc} = \frac{h}{p}.$$

Indeed, with De Broglie's hypothesis the results from the double-slit experiment make very much sense when we assume that a *wave* – not a particle – passes the slit, but that a *particle* is being found at the screen.

Before the ground-shaking surprise the wave ψ could be ascribed because the electrons and photons were so large in number. Quantum mechanics says that *each particle is also a wave*, in the sense that it is ascribed a wave $\psi(x)$, whose square amplitude $|\psi(x)|^2$ is the probability density that the particle will be found at the position x.

Each particle is of course observed at some specific place x along the screen when the wave arrives. However, we cannot tell what the wave itself looks like. This is because the wave is only a probability density for one single particle's position. Only if a large ensemble of particles starting out the same way hit the screen we may deduce from a histogram what $|\psi(x)|^2$ looks like. On the other hand an intense electron beam will completely obscure the particle-nature, and we will observe a continuous wave-front of particles, very much like as in the case with light. We see that the probabilistic nature of measurements is fundamental to quantum mechanics.

So why do we have Newtonian physics in the first place? The answer lies in the number h, which is a very small number. If we calculate the wavelength of, say, a 1 kg basket ball moving with 1 m/s, we will get $\lambda \approx 6 \times 10^{-27}$ m, a number which is extremely small compared to the size of the basketball. To observe diffraction patterns this wavelength must be comparable to the size of the slits which is absurd. It is very difficult to come up with an experiment showing the wavelike nature of basketballs.

In summary we have experimentally turned down the concept of microscopic particles moving along definite trajectories. Instead we say that they have a wave-like nature and that their motion is associated with some wave $\psi(x)$. Classical Newtonian physics cannot be used, at least in cases where the De Broglie wavelength h/p is comparable to the size of the system considered.

1.3 The Postulates of Quantum Mechanics

In this section we will introduce the postulates of non-relativistic quantum mechanics. Even though we until now have been talking about waves distributed in space we shall make a formulation in more abstract mathematical terms. The connection between these abstract terms and the concrete example of the double-slit experiment will become clear as the discussion progresses.

The language of quantum mechanics is linear algebra, with all its implications. This connection is clearly seen through the first three postulates.

Postulate 1
The state of a quantum mechanical particle is described by a vector Ψ in a complex Hilbert space $\mathcal{H}$. On the other hand, all the possible states of the particle are exactly $\mathcal{H}$ minus the zero vector.

Not every vector of $\mathcal{H}$ corresponds to *distinct* states. Every vector

$$\Psi' = \alpha\Psi$$

with α a (complex) scalar, corresponds to the same state. The different "rays" or directions in Hilbert space thus corresponds to distinct physical states. We will show this explicitly later, when discussing observables and measurements.

Particles exist in physical space, and this is reflected through the fact that our Hilbert space almost always contains $L^2(\mathbb{R}^3)$, the set of square-integrable complex-valued functions on $\mathbb{R}^3$. Indeed, the wave-shape of the double-slit experiment is just the vector Ψ represented as an L^2-function. (See Refs. [15–17] for a discussion of L^2.)

But unlike in classical physics a particle in quantum mechanics has not only spatial degrees of freedom. It may also have so-called *spin*, which is an additional intrinsic degree of freedom found in many elementary particles. In that case our Hilbert space looks like

$$\mathcal{H} = L^2(\mathbb{R}^3) \otimes \mathbb{C}^n,$$

where n depends on the fundamental particle in consideration. In other words, the function $\Psi(\boldsymbol{x})$ has n complex components $\Psi^{(\sigma)}$, $\sigma = 1, 2, \ldots, n$. For so-called spin-$\frac{1}{2}$ particles we have $n = 2$. The most fundamental examples of such particles are electrons, protons and neutrons, the constituents of all atoms and also most of the known universe. Photons, by the way, are spin-1 particles, with $n = 3$, but they do not obey non-relativistic quantum mechanics as they have zero rest mass.

Sometimes we reduce our Hilbert space (in some appropriate way) and consider particles in one or two dimensions. We may also neglect the spin degrees of freedom as in the double-slit illustration, or even neglect L^2 when our particle is confined to a very small volume in space.[4] In addition, other ways of approximating $\mathcal{H}$ by neglecting some parts of it are used. Indeed, the finite element methods to be discussed can be viewed in this way.

There are obviously striking differences between the classical way of describing a particle's state and the quantum mechanical way. The classical state is fully specified with momentum $\boldsymbol{x}$ and position $\boldsymbol{p}$; six degrees of freedom in total. But in quantum mechanics the particle's state is only completely specified with an infinity of complex numbers, as L^2 is an infinite-dimensional vector space. This difference stems from the probabilistic interpretation of the wave function hinted at in the previous section; we need a probability density for all points in space.

The second postulate concerns physically observable quantities.

Postulate 2
Every measurable quantity (i.e., observable) is represented by an Hermitian linear operator in $\mathcal{H}$.

For every classical dynamical variable $\omega(\boldsymbol{x}, \boldsymbol{p})$ there corresponds an operator Ω obtained by operator substitution of the fundamental position and momentum operators $\boldsymbol{X}$ and $\boldsymbol{P}$, respectively:

$$\Omega = \omega(\boldsymbol{x} \to \boldsymbol{X}, \boldsymbol{p} \to \boldsymbol{P}).$$

The components of $\boldsymbol{X}$ and $\boldsymbol{P}$ are operators defined through the fundamental quantum commutation relation

$$[X_i, P_j] = i\hbar\delta_{ij}.$$

When Ψ is represented as an L^2-function, the position operator X_i is just a multiplication of $\Psi(\boldsymbol{x})$ with x_i. The momentum operator then becomes

$$P_i = -i\hbar\frac{\partial}{\partial x_i},$$

Note however that this choice is not unique, as shown in Ref. [7]. Furthermore, as it stands now it really makes an assumption on the differentiability of Ψ.

[4] See chapter 3 for a further discussion.

We have simplified things somewhat. Given ω, the operator Ω may be ambiguous. As an example consider the dynamical quantity $\omega = xp$ for a particle in one dimension. Since X and P do not commute we cannot tell what operator XP or PX to prescribe, and the physical results are not equivalent when using the different operators. In addition, neither operator is Hermitian. The remedy is most often to use the so-called Weyl product

$$\Omega = \frac{1}{2}(XP + PX)$$

instead, but there is no universal recipe in more general cases where this fails. We will not elaborate anymore on this, but for a more thorough discussion, see Ref. [18].

The observables Ω described above have a classical counterpart. But quantum particles have degrees of freedom that have no classical meaning. In this part of Hilbert space we also have linear Hermitian operators, and these cannot be interpreted classically. Examples of such observables are the spin operators that are used to measure the direction of the intrinsic spin of spin-$\frac{1}{2}$ particles.

The third postulate concerns (ideal) quantum mechanical measurements of observables. The meaning of ideal measurements are delicate, see page 122 of Ref. [7], and we will only consider them in a mathematical and in an operational way, without discussing the deeper physical meanings of them, such as if ideal measurements are possible at all.

Postulate 3
The only possible values obtainable in an (ideal) measurement of Ω *are its eigenvalues* ω_n*. They have a probability*

$$P(\omega_n) \propto (\Psi, \Pi_n \Psi) \tag{1.4}$$

of occurring, with Π_n *the projection operator onto the eigenspace of* ω_n*. Immediately after a measurement of* Ω*, the quantum state collapses into* $\Pi_n\Psi$*.*

This postulate contains a great deal of information, vital in order to understand how quantum mechanics work. We assume for simplicity that the spectrum ω_n of Ω is discrete.

First of all we have the *collapse of the state vector*. When we have obtained a value ω_n for the observable Ω the state of the particle changes into $\Pi_n\Psi$, the projection of Ψ onto the eigenspace of ω_n. Contrary to a classical system, measuring an observable quantity interferes with the system, and it is inevitable as well. This is perhaps the most fundamental difference between classical physics and quantum physics.

An implication of this in terms of the double-slit experiment, is that we *cannot follow the particle's trajectory through the slits*. Indeed, there is no such thing as a trajectory. When we measure the position we destroy the wave-pattern that the interaction with the slits created, and the particle will never reach the screen and contribute to the distribution dictated by the same wave-pattern. Thus, we cannot observe *both* the particle nature and the wave nature at the same time – an important fact.

But what about the proportionality factor in Eqn. (1.4)? This is where we find out about which vectors in $\mathcal{H}$ correspond to distinct physical states.

Since the only obtainable values for Ω are the ω_ns, we must have

$$\sum_n P(\omega_n) = 1.$$

That is, the total probability of getting one of the eigenvalues when performing a measurement is unity.

Since Ω is an Hermitian operator, it has a complete set of orthonormal eigenvectors Φ_n so that we may expand Ψ in this basis, viz.,

$$\Psi = \sum_n (\Phi_n, \Psi)\Phi_n.$$

For simplicity we assume that we have no degeneracy of the eigenvalues ω_n, that is they are all distinct. Since Φ_n are orthonormal, we get

$$\Pi_n \Psi = (\Phi_n, \Psi)\Phi_n,$$

and

$$1 = \sum_n P(\omega_n) = C \sum_n (\Psi, \Pi_n \Psi) = C \left(\Psi, \sum_n (\Phi_n, \Psi)\Phi_n \right) = C(\Psi, \Psi).$$

For this to hold, we must choose $C = \|\Psi\|^{-2}$. In that case, we have an explicit formula for $P(\omega_n)$:

$$P(\omega_n) = \frac{(\Psi, \Pi_n \Psi)}{(\Psi, \Psi)}.$$

Scaling Ψ with some arbitrary nonzero complex number α, we see that the probabilities are all conserved. This must hold for *every* observable, and thus the probability distribution of every possible observable is conserved when we scale Ψ. This is why each *direction* in Hilbert space corresponds to distinct physical states, rather than each individual vector.[5] We simply cannot *observe* any difference among the different vectors along the ray.

We only have a probabilistic way of forecasting what value an observable Ω will have if we measure it. The double-slit experiment demonstrated this clearly when we measured the position of each particle.

We have assumed here that the spectrum of Ω is discrete. This is not always the case, and in the case of a continuous spectrum, we have to change our formulation a little bit: The only obtainable values becomes $\omega(x)$, where x is a parameter in some interval (instead of a discrete index), and $P(\omega(x))$ becomes a *probability density* instead of a probability.

So far we have discussed the state Ψ of a particle and the description of observable quantities. What is left is the time development of the system, and this is exactly what is given in the fourth postulate.

Postulate 4
The time development of the quantum state Ψ is given by time dependent Schrödinger equation, viz.,

$$i\hbar \frac{\partial}{\partial t} \Psi(t) = H\Psi(t). \tag{1.5}$$

We may interpret this postulate for the double-slit example. Eqn. (1.5) tells us in what way the particle-wave will move through the slits. Compare the Schrödinger equation with Hamilton's equations of motion and Newton's second law, which are two equivalent classical counterparts of the fourth postulate. They dictate the unique evolution in time of the state of a material system.

Since Eqn. (1.5) is a first order differential equation, knowledge of the wave Ψ at one instant makes us able to forecast the evolution at all times, past and future. We may demonstrate this explicitly. Note that when we describe the state Ψ as a function of spin and the spatial coordinates, Eqn. (1.5) becomes a coupled set partial differential equations of second order in space. We will investigate (partial and ordinary) differential equations later in chapter 3 and 4.

We will assume that H is independent of time. Given $\Psi(t_1)$, suppose we want to find $\Psi(t_2)$. Even though $\Psi(t)$ is a vector, we may create a Taylor expansion of $\Psi(t_2)$

[5] There is also a reason connected to the fourth postulate, on time development of Ψ. Does $\alpha\Psi$ develop differently in time compared to Ψ? The answer is negative due to the Schrödinger equation being linear.

around $t = t_1$:[6]

$$\Psi(t_2) = \sum_{n=0}^{\infty} \frac{\Delta t^n}{n!} \left.\frac{\partial^n \Psi}{\partial t^n}\right|_{t=t_1},$$

where $\Delta t = t_2 - t_1$. But according to Eqn. (1.5),

$$\left.\frac{\partial \Psi(t)}{\partial t}\right|_{t=t_1} = -\frac{i}{\hbar} H \Psi(t_1),$$

so that our Taylor expansion becomes:

$$\Psi(t_2) = \sum_{n=0}^{\infty} \frac{1}{n!} \left(-\Delta t \frac{i}{\hbar} H\right)^n \Psi(t_1) = \exp\left(-\Delta t \frac{i}{\hbar} H\right) \Psi(t_1),$$

where we have used the definition of the exponential of an operator.

Defining the so called *propagator*, or time evolution operator,

$$\mathcal{U}(t_2, t_1) = \exp\left(-(t_2 - t_1)\frac{i}{\hbar} H\right), \tag{1.6}$$

we may write

$$\Psi(t_2) = \mathcal{U}(t_2, t_1)\Psi(t_1).$$

In other words we have found a linear operator $\mathcal{U}$ that solves the time dependent Schrödinger equation in the sense that it takes a quantum state $\Psi(t_1)$ into itself at time t_2, at least formally.

Note that since H is Hermitian, that is $H^\dagger = H$, we have

$$\mathcal{U}^\dagger(t_2, t_1) = \exp\left((t_2 - t_1)\frac{i}{\hbar} H\right) = \mathcal{U}(t_1, t_2) = \mathcal{U}(t_2, t_1)^{-1},$$

so that $\mathcal{U}$ is a unitary operator. Alternatively, we may note that the adjoint is simply a Taylor series *backwards in time*, and thus unitarity follows immediately.

For time dependent Hamiltonians things are a little bit more complicated, and we defer its discussion until appendix A. The operator $\mathcal{U}$ is still a unitary operator from the Taylor expansion argument.

An important implication of the unitarity of the propagator, is that the norm of $\Psi(t)$ is conserved in time, viz.,

$$\|\Psi(t_2)\|^2 = \|\mathcal{U}(t_2, t_1)\Psi(t_1)\|^2 = (\Psi(t_1), \mathcal{U}^\dagger(t_2, t_1)\mathcal{U}(t_2, t_1)\Psi(t_1)) = \|\Psi(t_1)\|^2.$$

Conservation of norm is equivalent to conservation of probability. Recall that we may decompose a state Ψ into its components along an orthonormal basis, viz.,

$$\Psi = \sum_n c_n \Phi_n,$$

where the Φ_n may be precisely the eigenvectors of some observable Ω, again assumed to be non-degenerate and discrete for simplicity. Then, the probability of obtaining the eigenvalue ω_n from a measurement of Ω is

$$P(\omega_n) = \frac{(\Psi, \Pi_n \Psi)}{\|\Psi\|^2} = \frac{|c_n|^2}{\|\Psi\|^2} = \frac{|(\Phi_n, \Psi)|^2}{\|\Psi\|^2},$$

[6] This requires, of course, that the solution to the differential equation exists, which we silently assume here.

and all these add up to unity, viz.,

$$\sum_n P(\omega_n) = \sum_n \frac{|c_n|^2}{\|\Psi\|^2} = \|\Psi\|^{-2} \sum_n |c_n|^2 = \|\Psi\|^{-2} \sum_n (\Psi, \Phi_n)(\Phi_n, \Psi)$$
$$= \|\Psi\|^{-2} (\Psi, \left(\sum_n \Pi_n \right) \Psi) = \|\Psi\|^{-2} \|\Psi\|^2 = 1.$$

We have used that the eigenvectors of Ω constitute a basis, that is the sum of all the projection operators on the eigenspaces is the identity, viz.,

$$\sum_n \Pi_n = \mathbf{1}.$$

Thus, when $\|\Psi\|$ is conserved, so is the total probability for obtaining some of the eigenvalues ω_n when measuring Ω.

In short, unitarity of $\mathcal{U}(t_2, t_1)$ allows for probabilistic interpretation of Ψ.

1.4 More on the Solution of the Schrödinger Equation

In this rather compact section we will take a closer look at the solution of the Schrödinger equation (1.5) for time independent Hamiltonians, i.e., where the Hamiltonian does not contain an explicit dependence on time, viz.,

$$\frac{\partial}{\partial t} H \equiv 0.$$

Since H is Hermitian, we may find a basis $\mathcal{H}$ of eigenvectors of H, viz.,

$$H\Phi_n = E_n \Phi_n, \tag{1.7}$$

and we assume for simplicity that the energy spectrum (i.e., the eigenvalues of H) are discrete and non-degenerate. Eqn. (1.7) is referred to as *the time independent Schrödinger equation*. We expand the wave function $\Psi(t)$ in this basis, viz.,

$$\Psi(t) = \sum_n c_n(t) \Phi_n.$$

Inserting this expansion into the time independent Schrödinger equation yields

$$i\hbar \sum_n \frac{\mathrm{d}c_n}{\mathrm{d}t} \Phi_n = \sum_n E_n c_n(t) \Phi_n.$$

Both sides represent a non-zero element in $\mathcal{H}$ in terms of a basis. Hence, the terms must be equal as well, i.e.,

$$i\hbar \frac{\mathrm{d}c_n}{\mathrm{d}t} = E_n c_n(t)$$

which implies

$$c_n(t) = c_n(0) e^{-iE_n t/\hbar}.$$

The solution to the time dependent Schrödinger equation then reads

$$\Psi(t) = \sum_n c_n(0) e^{-iE_n t/\hbar} \Phi_n, \tag{1.8}$$

where $\Psi(0) = \sum_n c_n(0) \Phi_n$.

The coefficients c_n along the energy basis rotate with angular frequency $E_n/\hbar$. The magnitude $|c_n|$ is easily seen to be constant, and hence the probability of finding the system in the state Φ_n remains constant. It is then easy to see why Φ_n is called a *stationary state*. If $\Psi = \Phi_n$ for some n the system remains in the eigenstate at all t.

Notice that the solution (1.8) is simply the application of the time evolution operator from Eqn. (1.6) when expressed in a basis of eigenvectors for H, i.e., when H is diagonal, viz.,

$$\Psi(t) = e^{-\frac{i}{\hbar}tH}\Psi(0).$$

A consequence of Eqn. (1.8) is that solving the time dependent Schrödinger equation for time independent Hamiltonians is equivalent to diagonalizing H; that is equivalent to finding its eigenvalues and eigenvectors. This may be looked upon as a simpler problem than solving the original Schrödinger equation (1.5), and indeed in light of Eqn. (1.8) it is not surprising that finding an algorithm for solving the Schrödinger equation numerically for time dependent Hamiltonians is more difficult than for time independent ones.

1.5 Important Consequences of the Postulates

The postulates together with the facts of linear algebra in Hilbert spaces lay the foundation for quantum mechanics, and now is the time for gaining some physical insight based on the postulates. The goal of this section is to present some important results whose contents are vital for understanding and working with quantum mechanics.

1.5.1 Quantum Measurements

The second and third postulate tell us about observables in quantum mechanics. When measuring an observable ideally, in a sense that we shall not define, the quantum state collapses onto the eigenvector (or rather onto the eigenspace) corresponding to the eigenvalue found. Thus, we destroy our perhaps painstakingly constructed quantum system in the process.

As we have noted, quantum mechanics is probabilistic in nature. Some things we do not know for certain, only with a certain probability. Measuring a quantum state destroys it, making further investigations useless. It is clear that statistical methods will become in handy.

In statistics, if we have some quantity A taking on the values A_n, where n is some index, with some probability distribution P_n, then the expectation value $\langle A\rangle$ is defined as:

$$\langle A\rangle := \sum_n P_n A_n.$$

When performing a very large sequence of experiments, each obtaining one of the values A_n, we will on average get the value $\langle A\rangle$.

Applying this on a quantum observable Ω and its eigenvalues with their probabilities, we get, when we assume $\|\Psi\| = 1$:

$$\langle\Omega\rangle = \sum_n P(\omega_n)\omega_n = \sum_n (\Psi,\Phi_n)(\Phi_n,\Psi)\omega_n = \sum_n (\Psi,\Omega\Phi_n)(\Phi_n,\Psi) = (\Psi,\Omega\Psi),$$

where we have used that the eigenvectors Φ_n constitute an orthonormal basis for our Hilbert space. If Ψ is not normed to unity, then we must introduce an additional factor $\|\Psi\|^{-2}$ as is easily seen.

Theorem 1
The expectation value of some observable A in the state Ψ is given as:

$$\langle A \rangle = \frac{(\Psi, A\Psi)}{(\Psi, \Psi)}.$$

Although we have only shown this for a discrete, non-degenerate spectrum, it also holds in general for continuous and degenerate spectra, the key reason being the orthonormality of the eigenvectors and the fact that they span the whole Hilbert space $\mathcal{H}$.

Finding the average value of an observable thus amounts to applying the observable (i.e., its operator representation) to the state, and projecting the result back onto the state. This is the best prediction we can make of the outcome of a measurement.

1.5.2 Sharpness and Commuting Observables

When we measure an observable Ω the state collapses onto the eigenspace of the obtained value ω_n. If we try to measure the same observable immediately after the previous measurements, we will *with certainty* get ω_n as result, because by the third postulate we know that $\Psi = \Phi_n$, the eigenstate corresponding to ω_n. We say that ω_n is a *sharply determined value of* Ω *in the state* Ψ. In other words, the probability is 1 that we find ω_n when measuring the observable for the state Ψ.

If the system in consideration is in an eigenstate of an observable Ω, then we know with certainty that a measurement will yield the corresponding eigenvalue. Conversely, if we want $P(\omega_n) = 1$, then $(\Phi_m, \Psi) = 0$ for $m \neq n$. Thus, having a sharp eigenvalue of some observable for a system is equivalent to the system being in an eigenstate of the observable.

An important question arises, and indeed it is one of the most important: What conditions must be fulfilled for two observables to have sharply determined values at the same time?

Theorem 2
A sufficient and necessary condition for two observables A and B to have a common set of orthonormal eigenvectors is that A and B commute, i.e., if

$$[A, B] := AB - BA = 0.$$

Proof: Again we show this for a discrete spectrum, but this time degeneracy is also included.

An immediate requirement for two observables to have sharp values in the same state at the same time is that the observables must have common eigenvectors, because a sharp value is only obtained in the case of the system being in an eigenstate.

Assume that a_n and b_n are the eigenvalues of A and B, respectively, and that Φ_n are the corresponding eigenvectors of both operators:

$$\begin{aligned} A\Phi_n &= a_n\Phi_n \\ B\Phi_n &= b_n\Phi_n \end{aligned}$$

Operating on the first relation with B and the second with A, and subtracting yields

$$(AB - BA)\Phi_n = (a_n b_n - b_n a_n)\Phi_n = 0,$$

and since Φ_n is not the zero vector, we get

$$[A, B] = 0.$$

For the converse, assume that $[A, B] = 0$, and assume that we are given the eigenvalues and eigenvectors of A, viz.,

$$A\Phi_n = a_n\Phi_n.$$

Using commutability we get

$$A(B\Phi_n) = B(A\Phi_n) = a_n(B\Phi_n),$$

implying that $B\Phi_n$ is an eigenvector of A with the eigenvalue a_n.

Assume that a_n is a non-degenerate eigenvalue. Then $B\Phi_n$ must be some scalar multiple of Φ_n, because the eigenspace has only dimension one. Let us call this scalar b_n, viz.,

$$B\Phi_n = b_n\Phi_n,$$

and we are done.

Assume then that a_n is degenerate, and that the eigenspace has (infinite or finite) dimension g. Assume that Φ_n^m, $m = 1 \dots g$ is a basis for the eigenspace. $B\Phi_n$ must be in the eigenspace and thus be a linear combination of the basis vectors, viz.,

$$B\Phi_n^m = \sum_i M_{mi}\Phi_n^i.$$

This vector is not necessarily an eigenvector of B. But a linear combination Ψ might be, viz.,

$$\Psi = \sum_i c_i\Phi_n^i. \tag{1.9}$$

We assume that Ψ is an eigenvector for B with eigenvalue b, viz.,

$$B\Psi = b\Psi,$$

and writing out the left hand side, we get

$$B\Psi = B\sum_i c_i\Phi_n^i = \sum_i\sum_j c_i M_{ij}\Phi_n^j.$$

On the right hand side we have

$$b\Psi = b\sum_j c_j\Phi_n^j.$$

Since the vectors Φ_n^m are linearly independent, the right and left hand side must be equal term by term also, viz.,

$$\sum_i M_{ij}c_i = bc_j,$$

which is nothing more than the jth component of the matrix equation

$$M\boldsymbol{c} = b\boldsymbol{c}.$$

This is a new eigenvalue problem of dimension g. But since B is Hermitian, so is the $g \times g$ matrix M, and thus we may find g eigenvalues b_j and eigenvectors $\boldsymbol{c}$. This gives coefficients c_i for Eqn. (1.9), and thus we have found g (orthonormal) eigenvectors of B which also are eigenvectors of A. We have found that A and B have a common set of eigenvectors, and we are finished. ■

Note that even though Φ_n^m are eigenvectors corresponding to the *same eigenvalue* a_n for A, the eigenvalues b_j found for B may be *different*.

1.5.3 Uncertainty Relations: Heisenberg's Uncertainty Principle

The converse of sharpness is that Ψ is *not* an eigenstate for our observable A, and thus there is some probability that we will get another value than a_n when measuring the observable. As a measure of the degree of sharpness we may use the standard deviation ΔA, viz.,

$$\Delta A := \sqrt{\langle A^2 - \langle A\rangle^2\rangle}. \tag{1.10}$$

For an eigenstate Φ_n of A, we trivially get

$$\Delta A^2 = (\Phi_n, A^2\Phi_n) - (\Phi_n, A\Phi_n)^2 = 0,$$

so we have zero uncertainty in the case of a sharp value for the observable A.

If A has standard deviation ΔA (in a given state), what is the optimal standard deviation ΔB for another observable B? The question has fundamental importance, and the answer is one of the most striking facts of quantum mechanics.

Theorem 3
Given two observables A and B, we have the uncertainty relation

$$\Delta A \Delta B \geq \frac{1}{2}\left|\langle [A, B]\rangle\right|. \tag{1.11}$$

Proof: The standard text-book proof is a rather elegant one. Define two new operators $\hat{A}$ and $\hat{B}$ by:

$$\hat{A} = A - \langle A\rangle \mathbf{1}, \quad \text{and} \quad \hat{B} = B - \langle B\rangle \mathbf{1}.$$

Of course, $\hat{A}$ and $\hat{B}$ are also Hermitian. Note that $[\hat{A}, \hat{B}] = [A, B]$. Let Ψ be an arbitrary vector in $\mathcal{H}$, and define

$$H = \hat{A} + i\alpha\hat{B},$$

where $\alpha \in \mathbb{R}$ is arbitrary. We must have

$$\|H\Psi\|^2 = (\Psi, H^\dagger H\Psi) \geq 0,$$

by definition of the norm. Calculating the product $H^\dagger H$ yields

$$H^\dagger H = \hat{A}^2 + i\alpha[\hat{A}, \hat{B}] + \alpha^2\hat{B}^2,$$

and thus

$$\|H\Psi\|^2 = \langle \hat{A}^2\rangle + i\alpha\langle [\hat{A}, \hat{B}]\rangle + \alpha^2\langle \hat{B}^2\rangle.$$

Note that we must have $\langle i[\hat{A}, \hat{B}]\rangle \in \mathbb{R}$ since the norm is a real number.[7] We then choose

$$\alpha = -\frac{\langle i[\hat{A}, \hat{B}]\rangle}{2\langle \hat{B}^2\rangle}.$$

Insertion into the norm gives us

$$\langle \hat{A}^2\rangle \geq -\frac{\langle [\hat{A}, \hat{B}]\rangle^2}{4\langle \hat{B}^2\rangle}.$$

When noting that $\langle \hat{A}^2\rangle = \Delta A^2$ and similarly for ΔB, we immediately get the desired result (1.11). ■

A very important special case is the *Heisenberg uncertainty principle*:

$$\Delta X_i \Delta P_j \geq \frac{\hbar}{2}\delta_{ij}. \tag{1.12}$$

[7] It is easy to show that $[A, B]^\dagger = -[A, B]$, and thus $i[A, B]$ is Hermitian.

This relation states that if the particle is localized to degree ΔX_i in the ith spatial direction, then the corresponding momentum is sharp to a degree limited by the Heisenberg uncertainty principle.

Contrast this result to the classical principle depicted in Fig. 1.2. The classical particle offers complete information of its state at all times through the coordinates and momenta. It has $\Delta X_i \cdot \Delta P_i = 0$ in all situations. The quantum particle behaves however in a more complex way. A quantum particle localized perfectly like this will have infinite ΔP_i!

1.5.4 Ehrenfest's Theorem

We will state and prove the so-called Ehrenfest's theorem, which in some sense connects the time development of the expectation values of X_i and P_i to Hamilton's equations and the movement of a classical particle.

But first we need a more general result concerning time dependencies of expectation values.

Theorem 4
Given an observable A, its expectation value develops in time according to the differential equation

$$\frac{\mathrm{d}}{\mathrm{d}t}\langle A\rangle = \frac{i}{\hbar}\langle [H,A]\rangle + \langle \frac{\partial A}{\partial t}\rangle. \tag{1.13}$$

The last term accounts for the explicit dependence on time of A.

Proof: We begin by noting that with an expression such as

$$(\Psi, A\Psi),$$

we may use the familiar product rule for differentiation (with respect to t). This can be done since we may expand both operators and states in Taylor series in time,[8] viz.,

$$\Psi(t+\Delta t) = \Psi(t) + \Delta t\frac{\partial \Psi(t)}{\partial t} + \mathcal{O}(\Delta t^2)$$
$$\text{and} \quad A(t+\Delta t) = A(t) + \Delta t\frac{\partial A(t)}{\partial t} + \mathcal{O}(\Delta t^2).$$

Using these expansions when differentiating for example $A\Psi$ we get

$$\begin{aligned}\frac{\partial}{\partial t}(A(t)\Psi(t)) &= \lim_{\Delta t\to 0}\frac{1}{\Delta t}\left(A(t+\Delta t)\Psi(t+\Delta t) - A(t)\Psi(t)\right)\\ &= \lim_{\Delta t\to 0}\frac{1}{\Delta t}\left((A(t)+\Delta t\frac{\partial A}{\partial t})(\Psi(t)+\Delta t\frac{\partial \Psi}{\partial t}) - A(t)\Psi(t)\right)\\ &= \frac{\partial A(t)}{\partial t}\Psi(t) + A(t)\frac{\partial \Psi(t)}{\partial t}.\end{aligned}$$

If we are able to expand some time dependent quantity in Taylor series, this argument will hold.

Thus, we get

$$\begin{aligned}\frac{\partial\langle A\rangle}{\partial t} &= \frac{\partial}{\partial t}(\Psi, A\Psi) = (\frac{\partial\Psi}{\partial t}, A\Psi) + (\Psi, \frac{\partial A}{\partial t}\Psi) + (\Psi, A\frac{\partial\Psi}{\partial t})\\ &= (-\frac{i}{\hbar}H\Psi, A\Psi) + (\Psi, -\frac{i}{\hbar}AH\Psi) + \langle\frac{\partial A}{\partial t}\rangle\\ &= (\Psi, \frac{i}{\hbar}HA\Psi) + (\Psi, -\frac{i}{\hbar}AH\Psi) + \langle\frac{\partial A}{\partial t}\rangle\\ &= \frac{i}{\hbar}\langle[H,A]\rangle + \langle\frac{\partial A}{\partial t}\rangle.\end{aligned}$$

[8] Again we are assuming that the solution $\Psi(t)$ exists.

This completes the proof. ■

The important special case named Ehrenfest's theorem concerns the operators X_i and P_i in particular.

Theorem 5
Assume $\mathcal{H} = L^2(\mathbb{R}^3)$, that is we are working with square integrable functions and without spin degrees of freedom. Assume that $\nabla^2\Psi \in \mathcal{H}$. Assume that the Hamiltonian is given by

$$H = \frac{\boldsymbol{P}^2}{2m} + V(\boldsymbol{X}),$$

where V is a differentiable function of $\boldsymbol{X}$. Then,

$$\frac{\partial}{\partial t}\langle X_i\rangle = \frac{1}{m}\langle P_i\rangle \tag{1.14}$$

$$\text{and} \quad \frac{\partial}{\partial t}\langle P_i\rangle = -\langle\frac{\partial V}{\partial X_i}\rangle. \tag{1.15}$$

Proof: We begin by computing a commutator:

$$[H, X_i] = \frac{\boldsymbol{P}^2}{2m}X_i + VX_i - X_i\frac{\boldsymbol{P}^2}{2m} - X_iV = \frac{1}{2m}[\boldsymbol{P}^2, X_i] = \frac{1}{2m}[P_i^2, X_i]$$
$$= \frac{1}{2m}\left(P_i[P_i, X_i] + [P_i, X_i]P_i\right) = \frac{\hbar}{2im}P_i.$$

This gives us Eqn. (1.14) upon insertion into Eqn. (1.13).

Next, we compute another commutator:

$$[H, P_i] = [V, P_i] = -i\hbar[V, \frac{\partial}{\partial X_i}] = i\hbar\frac{\partial V}{\partial X_i},$$

when we regard the operators as operating on functions. This immediately gives Eqn. (1.15), when we insert the commutator into Eqn. (1.13). ■

In other words: Hamilton's equations of motion (1.2) are satisfied for the expectation values. This is not a strong link to classical theory as one might think. Consider a particle with a high degree of localization. Then we may take $\Delta X_i \approx 0$. In addition, if the particle has low speed (or high mass) we may take $\Delta P_i \approx 0$. The distribution for position and velocity exhibit the features of the state of a classical particle. Furthermore, the expectation value on the right hand side of Eqn. (1.15) become simply a sampling of the gradient of V at the particle's now well-defined position. For a general quantum particle one must sample the gradient of the potential *in whole space*, and not locally as in Newtonian mechanics. This is a very big difference. Similar results exist regarding semiclassical electrodynamics, as considered in section 1.6. The expectation values do not only sample the electromagnetic forces locally through a gradient or a curl, but through an integration of these over whole space.

1.5.5 Angular Momentum

An important class of observables comprises the *angular momentum operators*. They are defined as three Hermitian operators J_1, J_2 and J_3 constituting a vector operator $\boldsymbol{J}$ obeying the following commutation relations:

$$\begin{aligned}[J_1, J_2] &= i\hbar J_3 \\ [J_2, J_3] &= i\hbar J_1 \\ [J_3, J_1] &= i\hbar J_2\end{aligned} \tag{1.16}$$

In short, the three operators J_i obey

$$[J_i, J_j] = i\hbar\epsilon_{ijk}J_k,$$

where we have assumed Einstein's summation convention (summation over repeated indices) and the Levi-Civita symbol ϵ_{ijk} which is equal to the sign of the permutation of the numbers 1, 2 and 3, if ijk is a permutation of these, and zero otherwise.

The angular momentum operators get their names from the fact that the observables associated with each of the three components of the classical orbital momentum obeys the relations (1.16). The orbital momentum is given by

$$\boldsymbol{L} := \boldsymbol{r} \times \boldsymbol{p} = \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ x & y & z \\ p_x & p_y & p_z \end{vmatrix}$$
$$= (yp_z - zp_y)\boldsymbol{i} + (zp_x - xp_z)\boldsymbol{j} + (xp_y - yp_x)\boldsymbol{k}. \quad (1.17)$$

We will return to this later, when discussing the solution of the hydrogen atom.

Let us define J^2 as the operator given by

$$J^2 := J_iJ_i = J_1^2 + J_2^2 + J_3^2.$$

We call J^2 *the square length (or square norm) of the angular momentum.*

The components of the quantum mechanical angular momentum cannot be measured sharply at the same time, as we know from Theorem 3. For example,

$$\Delta J_1 \Delta J_2 = \frac{\hbar}{2}|\langle J_3\rangle|.$$

However, $\hbar$ is a very small number in macroscopic terms, actually essentially zero according to the correspondence principle. Thus, for the classical orbital angular momentum of Newtonian mechanics we may measure each component L_i simultaneously with in reality perfect accuracy. However, there are angular momentum operators that are *not* derived from classical quantities yielding important quantum mechanical effects. The intrinsic spin of particles such as electrons will display this, as we will see later.

It is rather straightforward to show that J^2 commutes with all three J_i.

Theorem 6
Given three angular momentum operators J_i, $i = 1, 2, 3$, the square norm $J^2 = J_iJ_i$ obeys the following commutation relation:

$$[J^2, J_i] = 0, \quad i = 1, 2, 3.$$

Proof: We begin by writing out the first argument of the commutator, viz.,

$$[J^2, J_i] = [J_jJ_j, J_i] = J_jJ_jJ_i - J_iJ_jJ_j = J_jJ_jJ_i - ([J_i, J_j] + J_jJ_i)J_j$$
$$= J_j[J_j, J_i] - i\hbar\epsilon_{ijk}J_kJ_j = i\hbar\epsilon_{jik}J_jJ_k - i\hbar\epsilon_{ijk}J_kJ_j$$

In the last expression, we are summing over both j and k. Swapping their names in one of the terms makes the expression vanish identically, viz.,

$$i\hbar\epsilon_{jik}J_jJ_k = i\hbar\epsilon_{kij}J_kJ_j = i\hbar\epsilon_{ijk}J_kJ_j.$$

We have used that $\epsilon_{ijk} = \epsilon_{kij}$ since cycling the symbols does not change the sign of the permutation. ■

According to Theorem 2, J^2 has a common set of eigenvectors with each of the components J_i. However, since the components by definition do not commute among

themselves, we cannot have a common set of eigenvectors for J^2 and *more* than one component at the same time.

We may then have a sharp value of J^2 and for example J_3, but J_1 and J_2 will not be sharply determined. Comparing this to the classical orbital momentum, this is like saying that we may know the absolute value of the angular momentum and the z-component, but not the x and y component.

We will now state a very important and striking theorem concerning the angular momentum operators. The theorem puts restrictions on the possible eigenvalues of J_i and J^2. Its proof rests solely on the defining relations (1.16).

In the proof, we use a *ladder operator* technique very similar to that of the solution of the harmonic oscillator in section 2.2 later. Thus it might be of interest to read these sections in conjunction. The ladder operators are defined as

$$\begin{aligned} J_+ &:= J_1 + iJ_2, \\ \text{and} \quad J_- &:= J_1 - iJ_2. \end{aligned}$$

Note that $J_- = J_+^\dagger$; they are Hermitian adjoints of each other.

Theorem 7
The square of the norm of the angular momentum, J^2, has a common set of orthonormal eigenvectors Φ_{jm} with J_3. The eigenvalues of J^2 and J_3 may only be given by

$$\begin{aligned} J^2\Phi_{jm} &= j(j+1)\hbar^2\Phi_{jm}, \\ \textit{and} \quad J_3\Phi_{jm} &= m\hbar\Phi_{jm}, \end{aligned}$$

where $j \geq 0$ is integral or half-integral, and where m takes the values

$$m = -j, -j+1, \ldots, j-1, j.$$

Thus, m is also integral or half-integral, and the eigenvalues of J^2 are degenerate with degeneracy (at least) $2j+1$.

Proof: At this stage, m and l are arbitrary, and we will gradually reduce their freedom to the restrictions stated in the theorem.

First of all j and m are dimensionless numbers, since $\hbar$ has the same dimension as L_i, as can be seen from Eqn. (1.16). Furthermore, we see that $j(j+1) \geq 0$, since L^2 is positive definite, and we may define $j \geq 0$ since the equation

$$j(j+1) = x$$

has a non-negative solution j for all $x \geq 0$.

We claim that $-j \leq m \leq j$. We prove this by considering the norm of the vectors Ψ_- and Ψ_+, which we define as

$$\Psi_\pm := J_\pm\Phi_{jm}.$$

The norm of $\Psi_\pm$ is easily calculated:

$$\begin{aligned} \|\Psi_\pm\|^2 &= (\Phi_{jm}, J_\pm J_\mp \Phi_{jm}) = (\Phi_{jm}, (J_1 \pm iJ_2)(J_1 \mp iJ_2)\Phi_{jm}) \\ &= (\Phi_{jm}, (J_1^2 + J_2^2 \pm i[J_2, J_1])\Phi_{jm}) = (\Phi_{jm}, (J - J_3^2 \pm \hbar J_3)\Phi_{jm}) \\ &= \hbar^2(j(j+1) - m(m \mp 1)) \|\Phi_{jm}\|^2 . \end{aligned}$$

Since the norm must be non-negative, we get

$$-j \leq m \leq j+1, \quad \text{and} \quad -j-1 \leq m \leq j,$$

for the plus and the minus sign, respectively. Hence,

$$-j \leq m \leq j,$$

as were claimed.

Next, we claim that m can only differ from j by an integer, and that j is either integral or half-integral. Equivalently, j is integral or half-integral, and so is m.

To prove this, we begin by noting that J^2 commutes with both J_- and J_+, so

$$J^2\Psi_\pm = \hbar^2 j(j+1)\Psi_\pm,$$

and $\Psi_\pm$ is also an eigenvector for J^2 with the same eigenvalue as Φ_{jm}. In other words, $\Phi_\pm$ lies in the same eigenspace for the J^2 operator's eigenvalue as Φ_{jm}. We then operate with J_3 on $\Psi_\pm$:

$$\begin{aligned} J_3\Psi_\pm &= J_3(J_1 \pm iJ_2)\Phi_{jm} = ([J_3, J_1] + J_1J_3 \pm i[J_3, J_2] \pm iJ_2J_3)\Phi_{jm} \\ &= (i\hbar J_2 \pm iJ_2J_3 \mp \hbar J_1 + J_1J_3)\Phi_{jm} = (i\hbar J_2 \pm i\hbar m J_2 \mp \hbar J_1 + \hbar m J_1)\Phi_{jm} \\ &= \hbar(m \pm 1)(J_1 \pm iJ_2)\Phi_{jm} = \hbar(m \pm 1)\Psi_\pm. \end{aligned}$$

That is, $\Psi_\pm$ is an eigenvector for J_3 with eigenvalue $\hbar(m \pm 1)$, that is

$$\Psi\pm = J_\pm\Phi_{jm} = C\Phi_{j(m\pm1)}.$$

By operating successively on one single Φ_{jm}, we get a ladder of increasing and decreasing eigenvalues and corresponding eigenvectors, but these cannot have eigenvalues outside the range $-j \ldots j$. Observe that

$$|C| = \hbar\sqrt{j(j+1) - m(m \pm 1)},$$

so that the sequence terminates at $j = m$ and $j = -m$ for J_+ and J_-, respectively. For both sequences to terminate properly, we must have j integral or half integral, because this is the only way to make the difference between the maximum and minimum value of m integral. Our claim is proven, and in fact we have also proved our theorem. ■

To sum up: j may only take the values

$$j = 0, \frac{1}{2}, 1, \frac{3}{2}, \ldots,$$

while m ranges like

$$m = -j, -j+1, \ldots, j-1, j.$$

We see that the eigenvalue $\hbar j(j+1)$ for J^2 is has a degeneracy of $2j+1$.[9] For a fixed value of j, the eigenspace has dimension $2j+1$. This eigenspace has a basis of eigenvectors of each J_i, but since they do not commute their eigenvectors are not the same. It is clear that if j is fixed we may describe the operators J_i as square Hermitian matrices of dimension $2j+1$, once we have chosen a basis for the eigenspace.

It is rather surprising that we may get so much information just from the commutation relations (1.16), and Theorem 7 has a wide range of applications; from finding the energy eigenstates of the hydrogen atom and other spherical symmetric systems to the theory on eigenspin of elementary particles such as electrons, the latter which we will turn to right now.

[9] *At least* $2j+1$ is more correct to say; it may happen that the eigenvalues of J_3 again are degenerated.

1.5.6 Eigenspin of the Electron

A surprising feature of the electron theory is the necessity of introducing an additional degree of freedom called *eigenspin* or just *spin*. As mentioned in section 1.3, the Hilbert spaces for electrons, protons, neutrons and many other particles are not only composed of $L^2(\mathbb{R}^3)$, but also $\mathbb{C}^n$, the latter ascribed to the spin degrees of freedom.

One of the experimental facts that lead to the discovery of electron spin was the famous Stern-Garlach experiment, in which silver ions were accelerated through a magnetic field. If the electrons where spinning in a classical sense, they would be deflected according to the distribution of the spin. It turned out however, that the deflected ions were concentrated into two spots, indicating a discrete nature of the eigenspin. See for example Ref. [11] for a complete account of the eigenspin discovery.

The eigenspin is described through angular momentum operators acting on vectors in $\mathbb{C}^n$. For a particle of spin s, we have the spin operator $\boldsymbol{S}$, and S^2 has by definition only one eigenvalue, namely $\hbar^2 s(s+1)$. The eigenspace of each S_i is then $n = 2s+1$ dimensional, according to Theorem 7, and we may choose $\mathbb{C}^n$ as the Hilbert space connected to eigenspin. The spin components becomes spin matrices, and we choose S_3 to be diagonal. In other words, the eigenvectors of S_3 are chosen as a basis for the spin part of Hilbert space, and the spin matrices are described in this representation.

Let us turn to the spin-1/2 particles, such as the electron. Since $s = 1/2$ we get $\mathbb{C}^2$ as our Hilbert space. Let S_3 be diagonal and write

$$\begin{aligned} \chi_+ &:= \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \\ \text{and} \quad \chi_- &:= \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \end{aligned}$$

Defining S_3 as the 2×2 matrix

$$S_3 = \frac{\hbar}{2} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix},$$

makes χ_+ have an S_3 eigenvalue of $+\hbar/2$ and χ_- an eigenvalue of $-\hbar/2$, viz.,

$$\begin{aligned} S_3\chi_+ &= \frac{\hbar}{2}\chi_+ \\ \text{and} \quad S_3\chi_- &= -\frac{\hbar}{2}\chi_-. \end{aligned}$$

Taking S_3 as diagonal is just a conventional choice: Later, when discussing the hydrogen atom, we use spherical coordinates in which the z-axis has a special meaning. In addition, interactions of both orbital momentum and eigenspin with a magnetic field becomes simpler to describe mathematically if we direct the field along the diagonal axis of spin, see chapter 3.

Using the raising and lowering operators of Theorem 7, we may find S_1 and S_2 as well:

$$\begin{aligned} S_1 &= \frac{\hbar}{2} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\ \text{and} \quad S_2 &= \frac{\hbar}{2} \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}. \end{aligned}$$

The matrices are easily seen to obey Eqn. (1.16).

Note that S_i does not act on any spatial degree of freedom. Thus, S_i commutes with both X_i and P_i. If the Hamiltonian does not contain some interaction term such as a magnetic field, then the spin operators also commute with H. Then the spin state

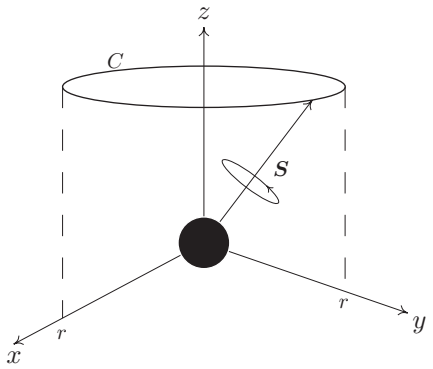

Figure 1.5: The "spinning top electron"

of the particle does not affect the time development of the spatial wave function, and we may ignore it altogether as it has no physical significance.

On the other hand, if H contains a magnetic field, then the magnetic moment associated with $\boldsymbol{S}$ will interact with the magnetic field and disturb this ignorance of the spatial wave function of the spin, such as demonstrated in the Stern-Garlach experiment.

Let us explain this a little further. Classically, when a charged particle of orbital momentum $\boldsymbol{L}$ is placed in a magnetic field $\boldsymbol{B}$, it gains an additional potential energy

$$V = -\frac{qg}{2mc}\boldsymbol{L}\cdot\boldsymbol{B} = -\boldsymbol{\mu}\cdot\boldsymbol{B},$$

where q is the particle charge. The dimensionless number g is called *the gyromagnetic factor* and classically this is related to the charge distribution of the particle. We write $\Gamma = qg/2mc$ as a shorthand later when discussing the particular form of the time dependent Schrödinger equation to be solved, in chapter 3.

The quantity $\boldsymbol{\mu}$ defined through

$$\boldsymbol{\mu} := \Gamma\boldsymbol{L}$$

is called the *magnetic moment*. When this is nonzero we have coupling between the magnetic field that may be present and the orbital momentum of the electron.

We assume that the eigenspin also has an associated magnetic moment $\boldsymbol{\mu}$ and a gyromagnetic factor to be determined experimentally. After all, we imagine the spin as some spinning property (albeit non-classical) of the electron which is a charged particle. Perhaps we may view the electron as a pure impossible-to-perceive spinning top. Thus, when electromagnetic forces are present in our system we must account for the spin state of the particle as well as the spatial state.

Can we get some sort of picture of the "spinning top"? If we imagine that the electron is in the state χ_+, then the z component of the eigenspin has a sharp value of $\hbar/2$. The x and y components however, are not possible to determine. But the norm of the spin is $3\hbar^2/4$. Consider Fig. 1.5, where we have depicted everything we know about the spin state of the electron. Even though the spin vector $\boldsymbol{S}$ is not possible to measure directly, we may imagine that this "actual spin" lies on the circle C with radius $r = \hbar/\sqrt{2}$ at a distance $\hbar/2$ above the xy plane.

1.5.7 Picture Transformations

In this section we shall consider so-called *picture transformations*, in which the vectors are transformed by a (possibly time dependent) unitary operator $T(t)$. It corresponds

to changing the frame of reference in Newtonian mechanics. We view our quantum states from another point of view, so to speak.

Assume that $T(t)$ is a unitary operator on Hilbert space with a possible explicit time dependence. Consider the transformation

$$\Psi(t) \longrightarrow \Psi_T(t) := T(t)\Psi(t). \tag{1.18}$$

This transformation is invertible due to $T(t)$ being unitary, i.e., $T^{-1} = T^\dagger$. The time dependence of Ψ is governed by the time dependent Schrödinger equation

$$i\hbar\frac{\partial}{\partial t}\Psi(t) = H(t)\Psi(t), \tag{1.19}$$

and we ask: What is the governing equation for $\Psi_T(t)$? The left hand side of Eqn. (1.19) may be rewritten as

$$i\hbar\frac{\partial}{\partial t}\left(T^\dagger(t)\Psi_T(t)\right) = i\hbar\left(\frac{\partial T^\dagger}{\partial t}\Psi_T(t) + T^\dagger\frac{\partial}{\partial t}\Psi_T(t)\right).$$

We rewrite the right hand side in a similar way, viz.,

$$H(t)\Psi(t) = H(t)T^\dagger(t)\Psi_T(t).$$

Multiplying Eqn. (1.19) with $T(t)$ from the left and reorganizing the equation we get

$$i\hbar\frac{\partial}{\partial t}\Psi_T(t) = H_T(t)\Psi_T(t), \tag{1.20}$$

with

$$H_T(t) := T(t)H(t)T^\dagger(t) + i\hbar\frac{\partial T}{\partial t}T^\dagger(t).$$

We have also used that

$$\frac{\partial T}{\partial t}T^\dagger = -T^\dagger\frac{\partial T}{\partial t},$$

which follows from differentiating $TT^\dagger = \mathbf{1}$ with respect to t.

We may also find the picture transformation of an arbitrary operator A. Let

$$\Phi = A\Psi,$$

and using Eqn. (1.18) we get

$$\Phi_T = T\Phi = TA\Psi = TAT^\dagger\Psi_T,$$

and thus

$$A(t) \longrightarrow T(t)A(t)T^\dagger(t), \tag{1.21}$$

is the picture transformation of an operator A.

As a consequence of the definition of the picture transformed operator, we get conservation of expectation values in the different pictures:

$$\langle A\rangle = (\Psi, A\Psi) = (T^\dagger\Psi_T, AT^\dagger\Psi_T) = (\Psi_T, TAT^\dagger\Psi_T) = \langle A\rangle_T.$$

We summarize these findings in a theorem.

Theorem 8
Consider a picture transformation given by

$$\Psi(t) \longrightarrow \Psi_T(t) := T(t)\Psi(t)$$

where $T(t)T^\dagger(t) = \mathbf{1}$. Then Ψ_T obeys the picture transformed Schrödinger equation

$$i\hbar\frac{\partial}{\partial t}\Psi_T(t) = H_T(t)\Psi_T(t),$$

and the picture transformed operators obey

$$A(t) \longrightarrow T(t)A(t)T^\dagger(t).$$

The two pictures are physically equivalent, in the sense that expectation values are conserved, viz.,

$$\langle A\rangle = \langle A\rangle_T.$$

Note that there is no reason to think of one picture as more fundamental than another. The original "Schrödinger picture" may be obtained from the picture given by $T(t)$ with the picture transformation

$$\Psi_T(t) \longrightarrow \Psi(t) = T^\dagger(t)\Psi_T(t).$$

If T is unitary, so is of course $T^\dagger$!

In other words: For every unitary operator there exists a picture. And when a unitary operator acts on an orthonormal basis, we get another orthonormal basis. Hence, changing the picture may be looked upon as simply changing the frame of reference, if we consider the frame of reference to be the basis we describe our states in.

To conclude we remark that through the following chain of identities,

$$\Psi_T(t') = T(t')\Psi(t') = T(t')\mathcal{U}(t',t)\Psi(t) = T(t')\mathcal{U}(t',t)T^\dagger(t)\Psi_T(t),$$

we obtain a relation for the transformed propagator $\mathcal{U}_T$, viz.,

$$\mathcal{U}_T(t',t) = T(t')\mathcal{U}(t',t)T^\dagger(t).$$

1.5.8 Many Particle Theory

Quantum mechanics is not only able to describe a single particle. It is also applicable to systems with many particles (i.e., material systems) in the same way as Hamiltonian mechanics is. We will here give a brief introduction to the non-relativistic quantum description of many particles. A thorough exposition is out of scope for this thesis; see Ref. [7]. Many-body theory is one of the most fundamental ingredients of various disciplines in physics such as nuclear physics, solid state physics and particle physics.

The quantum mechanical formalism for many particles is obtained from assuming that the wave function depends on *all* the particle coordinates $\boldsymbol{x}_i \in \mathbb{R}^3$, and the Hilbert space for N particles then becomes

$$\mathcal{H} = L^2(\mathbb{R}^{3N}) \otimes \mathbb{C}^{2s_1+1} \otimes \mathbb{C}^{2s_2+1} \otimes \ldots \otimes \mathbb{C}^{2s_N+1},$$

The fundamental commutator is

$$[X_{i,r}, P_{j,s}] = i\hbar\delta_{ij}\delta_{rs}, \quad i,j = 1\ldots N \text{ and } r,s = 1\ldots 3,$$

and we must bear in mind that there are $3N$ degrees of freedom. The commutators are thus defined for coordinates and momenta belonging to different particles as well.

Most applications of many-body theory involves so-called *identical particles*. Identical particles have the property that there is nothing distinguishing the states of systems obtained from one another by permuting the positions of the particles.

For a quantum mechanical system we may describe it in this way. If we by S_N denote the set of permutations of N symbols, and if we by $\sigma(\boldsymbol{x}_1, \ldots \boldsymbol{x}_N)$ denote this permutation applied to the N coordinate vectors, then the physical properties of $\Psi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$

should be identical to those of $\Psi(\sigma(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N))$. Here, we include the spin degrees of freedom in $\boldsymbol{x}_i$, since it is a coordinate on equal footing as the cartesian position even though it is just an integer.

Note that this is an additional postulate to the four already given in section 1.3.

It is difficult to make this notion precise. Most textbooks use the above definition of identical particles based on the permutations of positions in configuration space. Unfortunately this is physically inconsistent and should be avoided. Indeed, if the particles are distinguishable, the permutations of the particles is a *non-observable concept.*

Indistinguishability is fundamental. Introducing the postulate leads to theoretical predictions in perfect agreement with experiments. The physical identification of states obtained by permuting the particles is equivalent to the statement

$$\Psi(p_{ij}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)) = \pm\Psi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N),$$

where p_{ij} is a transposition of particle i and j. In other words, Ψ is either symmetric or anti-symmetric with respect to particle exchanges. Particles of nature can thus be divided into two categories; those with symmetric wave functions and those with anti-symmetric wave functions. The former particles are referred to as *bosons* and the latter as *fermions*. This is an experimental fact not to be overseen and indeed it is fundamental in most branches of microphysics.

A further experimental fact is that bosons always have integral spin and fermions always half-integral spin. Thus electrons, protons and neutrons are fermions, while photons are bosons.

For fermions we may immediately identify the so-called *Pauli principle*, stating that two (identical) fermions cannot occupy the same quantum state. If we assume that two particles i and j occupy the same place in space, then

$$\Psi(p_{ij}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)) = \Psi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) = -\Psi(p_{ij}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)),$$

so that

$$\Psi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) = 0$$

whenever two of the coordinates coincide. It is not obvious that this implies that two fermions are not allowed to be in the same state in general (and what is meant by "the same state" for two particles in a state for an N-particle system), but we have not yet developed the necessary quantum mechanical results to establish this.

In Ref. [19], J.M. Leinaas and J. Myrheim shows that the identical particle postulate is actually superfluous. By carefully treating the transition from classical mechanics to quantum mechanics they managed to obtain the quantum mechanical notion of identical particles from the corresponding classical notion which is easier to describe accurately.

The basic idea is that because of the indistinguishability of the classical particles their configuration space is *not* $\mathbb{R}^{3N}$ but rather $\mathbb{R}^{3N}/S_N$; the quotient space obtained by identifying configurations which differ only by a permutation of the positions of the particles. Such quotient spaces are more complicated than the Euclidean space we normally use, as they may have non-zero curvature and singularities.

For classical systems particles are not usually allowed to inhabit the same locations in space, simply because they are mutually repelling, such as electrons. In that case the effect of using the modified configuration space is not seen. In quantum mechanics the effect is profound, and for three dimensional systems the notion coincides with the one sketched above. For lower-dimensional systems however, Leinaas and Myrheim showed that a different behavior was possible as well; the sign change when transposing two particles instead becomes a phase change e^{ia}, with a real.

1.5.9 Entanglement

A famous quote by Einstein says that "God does not play dice." There are many possible ways to interpret this, but the original context in which it was uttered was theoretical physics. In this short section we will discuss some of the rather esoteric features of quantum physics. Ref. [20] contains a concise yet precise treatment of the subjects of this section.

Considering the fourth postulate, quantum physics is deterministic in the sense that the state evolves in time according to a differential equation. On the other hand, it is non-deterministic in the way that the outcome of an experiment is completely random. In 1935, Einstein, Podolsky and Rosen proposed a famous gedanken experiment in which the non-deterministic features yielded somewhat absurd consequences. Because of this Einstein felt that quantum physics must be *incomplete*; that there had to be a bigger "super-theory" that included the present form of quantum theory. To understand this paradox we first need to furnish the concept of *entanglement* for many-body systems.

Consider a particle with Hamiltonian H, which we for simplicity assume is independent of time. We diagonalize H, i.e., find orthonormal ϕ_n such that

$$H\phi_n = \epsilon_n \phi_n,$$

and ϕ_n form a basis for Hilbert space $\mathcal{H}$. For arbitrary one-particle functions we have

$$\psi = \sum_n c_n \phi_n.$$

If we now consider an N-particle system with a Hamiltonian given by

$$H' = \sum_{i=1}^{N} H(i),$$

where $H(i)$ denotes that only operators concerning degrees of freedom belonging to particle number i is present in the term, the state

$$\Phi = \phi_{n_1}(1) \otimes \phi_{n_2}(2) \otimes \cdots \otimes \phi_{n_N}(N),$$

is easily seen to be an eigenstate of H', viz.,

$$H'\Phi = (\epsilon_{n_1} + \cdots + \epsilon_{n_N})\Phi.$$

The Hilbert space for the N-particle system may be written as

$$\mathcal{H}' = \underbrace{\mathcal{H} \otimes \mathcal{H} \otimes \cdots \otimes \mathcal{H}}_{N \text{ terms}},$$

where $\mathcal{H}$ is the Hilbert space associated with one particle. The states Φ above is easily seen to constitute a basis for $\mathcal{H}'$. Thus we may construct quite general states of the N-particle system by considering the direct product of arbitrary one-particle functions, viz.,

$$\Psi = \psi_1(1) \otimes \psi_2(2) \otimes \cdots \otimes \psi_N(N).$$

These states are however not the most general ones. Indeed, a simple superposition of two such states cannot be factorized in this way. If we considered product states exclusively, then we would actually simply study the one-particle system. But *any* square-integrable function of the different coordinates may be used as a state and such non-product states are called *entangled states*.

As an example let us consider a one-particle Hamiltonian with only two eigenstates, viz.,

$$H\psi_i = \epsilon_i \psi_i, \quad i = 1, 2.$$

A basis of eigenstates for the two-particle Hamiltonian $H' = H(1) + H(2)$ is then given by the four states

$$\psi_1(1)\psi_1(2), \quad \psi_1(1)\psi_2(2), \quad \psi_2(1)\psi_1(2) \quad \text{and} \quad \psi_2(1)\psi_2(2).$$

(We have omitted the direct product symbols to economize.) Their energies are $2\epsilon_1$, $\epsilon_1+\epsilon_2$ (for two states) and $2\epsilon_2$, respectively. If we measure the total energy $H(1)+H(2)$ of the system, then these are the values we may obtain. The energy of one single particle, say $H(1)$, is also an observable, and upon measurement it will yield either ϵ_1 or ϵ_2. Consider then the state

$$\Psi = \frac{1}{\sqrt{2}} \left(\psi_1(1)\psi_2(1) - \psi_2(1)\psi_1(2) \right),$$

which is not an eigenfunction for the total energy, but rather an (anti-symmetric) linear combination of such. It is an entangled state.

If we measure the energy of particle 1 and obtain ϵ_1 the state collapses into

$$\Psi \longrightarrow \Psi_1(1)\Psi_2(2).$$

But then we know the energy of particle 2 because the new state is an eigenstate for $H(2)$. In other words, there is a perfect correlation between the energies of the particles. Measuring one particle's energy will also determine the energy of the other.

This is the essential content of the Einstein-Podolsky-Rosen paradox, but what is so paradoxical about this result, besides the fact that operating on one particle's degrees of freedom seemingly affects the other, even though $H(1)$ and $H(2)$ commute? One may prepare a quantum system in many ways. For example, one may prepare two particles to be localized in space; one here on earth and one on the moon. We may also prepare these particles' spins to be in an entangled state at the same time. Substituting "energy" with "spin" in the above argument then leads to the fact that there may be perfect correlation between the spins of two particles very far away from each other. A measurement on the particle at home will *instantaneously* affect the one at the moon. Einstein somewhat humorously called this phenomenon "spooky action at a distance."

Einstein and others felt that such behavior was absurd; that quantum mechanics should be *local*. An operation at a point should not immediately affect other points in space due to the limited speed of light. On these grounds, one may hope for some kind of local "super-theory" that contains quantum mechanics as a special case. Mathematically, such a super-theory is called a *hidden variable theory.*

The British mathematician John Bell derived a series of inequalities that any hidden variable theory must obey. These may actually be tested experimentally, and indeed experimental results tend to invalidate Bell's inequalities for quantum mechanics. Then quantum mechanics is a complete theory and the "spooky action" must be a real phenomenon.

1.6 Electromagnetism and Quantum Physics

An interesting class of time dependent Hamiltonians are the ones describing the interaction between a charged particle such as an electron and a classical electromagnetic field. In this section we will give a very brief summary of this semiclassical theory. It is called semiclassical because only the charged particle is described in quantum mechanical terms; the electromagnetic field is still a classical field which really should

be quantized to make a consistent theory. However, when the electromagnetic field has macroscopic magnitudes, the quantum behavior becomes neglible compared to the classical features, as dictated by the correspondence principle, see section 2.4.

1.6.1 Classical Electrodynamics

Let us first describe the electrodynamic theory in classical terms. On one hand we have a charged particle of charge q. This particle interacts with an electric field $\boldsymbol{E}$ and a magnetic field $\boldsymbol{B}$. In this context, a field means a vector quantity with three components varying in space. We refer to $\boldsymbol{E}$ and $\boldsymbol{B}$ jointly as the electromagnetic field.

The response of a charged particle of charge q due to the electromagnetic field is given by the Lorentz force:[10]

$$\boldsymbol{F} = q\left(\boldsymbol{E} + \frac{1}{c}\boldsymbol{v}\times\boldsymbol{B}\right). \tag{1.22}$$

The number c is the speed of light in vacuum.

On the other hand the famous Maxwell's equations give the response of the electromagnetic field due to a charge, or a charge distribution ρ in general:

$$\nabla\cdot\boldsymbol{E} = 4\pi\rho \tag{1.23}$$

$$\nabla\times\boldsymbol{E} + \frac{1}{c}\frac{\partial\boldsymbol{B}}{\partial t} = 0 \tag{1.24}$$

$$\nabla\cdot\boldsymbol{B} = 0 \tag{1.25}$$

$$\nabla\times\boldsymbol{B} - \frac{1}{c}\frac{\partial\boldsymbol{E}}{\partial t} = \frac{4\pi}{c}\boldsymbol{j} \tag{1.26}$$

The charge density $\rho(\boldsymbol{x})$ represents charges distributed in space. For a point particle of charge q situated at $\boldsymbol{r} = \boldsymbol{r}_0$ we have[11]

$$\rho = q\delta(\boldsymbol{r} - \boldsymbol{r_0}),$$

since the total charge then becomes

$$\int_{\text{all space}} \rho\, \mathrm{d}^3 r = q.$$

The current density $\boldsymbol{j}$ defines the movement of the charge density through the continuity equation

$$\frac{\partial\rho}{\partial t} + \nabla\cdot\boldsymbol{j} = 0. \tag{1.27}$$

This equation must hold if Maxwell's equations is to be fulfilled at the same time. For a charged particle we obtain

$$\boldsymbol{j} = q\boldsymbol{v}\delta(\boldsymbol{r} - \boldsymbol{r}_0),$$

where $\boldsymbol{v} = \dot{\boldsymbol{r}}_0$ is the velocity of the particle.

Note that the Lorentz force and Maxwell's equations are connected through the appearance of the charge density and the current density.

If we assume that no charges are present, i.e., $\rho = \boldsymbol{j} = 0$, we may easily combine Maxwell's equations and obtain the well-known wave equations

$$\frac{\partial^2\boldsymbol{E}}{\partial x^2} = \frac{1}{c^2}\frac{\partial^2\boldsymbol{E}}{\partial t^2},$$

$$\text{and}\quad \frac{\partial^2\boldsymbol{B}}{\partial x^2} = \frac{1}{c^2}\frac{\partial^2\boldsymbol{B}}{\partial t^2},$$

[10] In this thesis we employ Gaussian units for the electromagnetic field, which are defined through the proportionality constants in front of each term in Eqn. (1.22). See Ref. [21].

[11] Here we use the three dimensional Dirac delta function $\delta(\boldsymbol{x}) = \delta(x_1)\delta(x_2)\delta(x_3)$.

which shows that the electromagnetic fields propagate as waves through charge-free space with velocity c.

Working with the fields $\boldsymbol{E}$ and $\boldsymbol{B}$ directly may be cumbersome, and more insight is gained if we introduce the potentials ϕ and $\boldsymbol{A}$ as follows: Eqn. (1.24) implies that $\boldsymbol{B}$ may be written as a curl, viz.,

$$\boldsymbol{B} = \nabla \times \boldsymbol{A}.$$

We insert this into Eqn. (1.25), and get

$$\nabla \times \left(\boldsymbol{E} + \frac{1}{c}\frac{\partial \boldsymbol{A}}{\partial t} \right) = 0,$$

which implies that the expression in parenthesis may be written as a gradient, viz.,

$$-\nabla\phi = \boldsymbol{E} + \frac{1}{c}\frac{\partial \boldsymbol{A}}{\partial t},$$

or

$$\boldsymbol{E} = -\frac{1}{c}\frac{\partial \boldsymbol{A}}{\partial t} - \nabla\phi.$$

Note that we have reduced the six degrees of freedom of $\boldsymbol{E}$ and $\boldsymbol{B}$ to four degrees of freedom, a considerable simplification.

The potentials $\boldsymbol{A}$ and ϕ are not unique. We may add to $\boldsymbol{A}$ the gradient of an arbitrary function Λ and still get the same fields $\boldsymbol{E}$ and $\boldsymbol{B}$, and thus no physical distinction between the transformed potentials and the old ones. Let us prove this claim.

Let

$$\boldsymbol{A} \longrightarrow \boldsymbol{A}' = \boldsymbol{A} - \nabla\Lambda,$$

and let

$$\phi \longrightarrow \phi' = \phi + \frac{1}{c}\frac{\partial \Lambda}{\partial t}.$$

This transformation is called a *gauge transformation* and the function Λ is called a *gauge parameter*. The physical field $\boldsymbol{B}'$ of the transformed potential $\boldsymbol{A}'$ becomes

$$\begin{aligned} \boldsymbol{B}' &= \nabla \times (\boldsymbol{A} - \nabla\Lambda) = \nabla \times \boldsymbol{A} - \nabla \times \nabla\Lambda = \nabla \times \boldsymbol{A} \\ &= \boldsymbol{B}, \end{aligned}$$

since the curl of a gradient vanishes identically. For the transformed electric field $\boldsymbol{E}$ we obtain

$$\begin{aligned} \boldsymbol{E}' &= -\frac{1}{c}\frac{\partial}{\partial t}(\boldsymbol{A} - \nabla\Lambda) - \nabla\left(\phi + \frac{1}{c}\frac{\partial \Lambda}{\partial t}\right) \\ &= -\frac{1}{c}\frac{\partial \boldsymbol{A}}{\partial t} - \nabla\phi + \frac{1}{c}\nabla\frac{\partial \Lambda}{\partial t} - \frac{1}{c}\nabla\frac{\partial \Lambda}{\partial t} = \boldsymbol{E}. \end{aligned}$$

Thus we have shown the *gauge invariance* of the electric fields, and hence the physical laws, under a *gauge transformation*. It is then clear that the gauge parameter Λ cannot have any physical significance. Gauge invariance is a necessity of any physical theory utilizing the potentials $\boldsymbol{A}$ and ϕ directly. (If only the electromagnetic fields $\boldsymbol{E}$ and $\boldsymbol{B}$ themselves occur, then gauge invariance is automatically obtained.)

The choice of Λ is rather arbitrary. The proof rested on the differentiability of Λ and the commutability of differentiation with respect to time and with respect to space. Hence, at least continuously differentiable functions Λ may be used.

The gauge parameter may be chosen in different applications at our convenience to simplify calculations and derivations. We will return to this later on, when discussing the interaction of a charged quantum particle such as an electron and the classical

electromagnetic field. We mention here however that two gauges are quite common in classical electromagnetic theory and in the quantum description of this (also called QED; quantum electro-dynamics). These are the Coulomb-gauge and the Lorentz-gauge. In the former we require that $\nabla \cdot \boldsymbol{A} = 0$ and in the latter that $c\nabla \cdot \boldsymbol{A} = \partial\phi/\partial t$. These conditions may always be fulfilled if we have full freedom over Λ. See Ref. [7] for a discussion.

We state gauge invariance as a theorem.

Theorem 9
The electromagnetic fields $\boldsymbol{E}$ *and* $\boldsymbol{B}$ *consistent with Maxwell's equations* (1.23) *can be represented by the vector potential* $\boldsymbol{A}$ *and scalar potential* ϕ *such that*

$$\begin{aligned} \boldsymbol{B} &= \nabla \times \boldsymbol{A} \\ \text{and} \quad \boldsymbol{E} &= -\nabla\phi - \frac{1}{c}\frac{\partial \boldsymbol{A}}{\partial t}. \end{aligned}$$

Furthermore, the electromagnetic fields are gauge invariant, i.e., they are invariant under the transformation

$$\begin{aligned} \boldsymbol{A} &\longrightarrow \boldsymbol{A} - \nabla\Lambda \\ \text{and} \quad \phi &\longrightarrow \phi + \frac{1}{c}\frac{\partial \Lambda}{\partial t}, \end{aligned}$$

where $\Lambda(\boldsymbol{x}, t)$ *is any continuously differentiable real function.*

We may formulate the Lorentz force within the Hamiltonian framework in addition to the Newtonian framework, in which the force originally was given. In order to achieve this we must redefine our (classical) Hamiltonian and the canonical momenta. (This is due to the fact that the electromagnetic forces are not conservative.)

$$H_{em} := \frac{1}{2m}\left(\boldsymbol{p}_{em} - \frac{q}{c}\boldsymbol{A}\right)^2 + V(\boldsymbol{x}) + q\phi(\boldsymbol{x}), \tag{1.28}$$

where

$$\boldsymbol{p}_{em} := \boldsymbol{p} + \frac{q}{c}\boldsymbol{A} \tag{1.29}$$

is the canonical momentum for the electromagnetic Hamiltonian. Thus, the canonical momentum is no longer the regular linear momentum, but has an additional term proportional to the vector potential. It is easy to prove that this reproduces Newton's second law and the Lorentz force for the charged particle. Note also that the first term of H_{em} is exactly the kinetic energy, so that

$$H_{em} = \text{total energy}$$

as we are used to.

1.6.2 Semiclassical Electrodynamics

The quantization process, i.e., the process of applying the postulates to the semiclassical system, follows easily. The postulates were formulated for *canonical variables*, i.e., generalized coordinates and their corresponding momenta, and we now have acquired momenta that look just a little bit different than usual. The canonical coordinates for the electromagnetic Hamiltonian, H_{em}, are $p_{i,em}$ and x_i, where $p_{i,em}$ is *different* from the linear momentum mv_i (in Cartesian coordinates). This means that the quantum mechanical operator $P_{i,em}$ does *not* correspond to the ith linear momentum component, but rather

$$P_{i,em} = P_i + \frac{q}{c}A_i.$$

Thus for the position representation, when we view our quantum states as L^2 functions,

$$P_i \longrightarrow -i\hbar\frac{\partial}{\partial x_i} - \frac{q}{c}A_i \tag{1.30}$$

is the ith component of the linear momentum in semiclassical electrodynamics.

What about gauge invariance in this semiclassical theory? It is not obvious that a gauge transformation will leave the physics invariant. Indeed, the vector potential $\boldsymbol{A}$ appears explicitly in Eqn. (1.30), the expression for the linear momentum operator in the position representation. Of course, when we transform $\boldsymbol{A}$ and ϕ, we transform the Hamiltonian, so the Schrödinger equation also changes. We hope that this leaves room for gauge invariance, and indeed it is the case. To be more precise, if

$$\boldsymbol{A} \longrightarrow \boldsymbol{A}_\Lambda = \boldsymbol{A} - \nabla\Lambda$$

and

$$\phi \longrightarrow \phi_\Lambda = \phi + \frac{1}{c}\frac{\partial\Lambda}{\partial t}$$

are gauge transformations of the potentials, this induces a gauge transformation of the Hamiltonian, viz.,

$$H = H(\boldsymbol{A}, \phi) \longrightarrow H_\Lambda = H(\boldsymbol{A}_\Lambda, \phi_\Lambda).$$

If the original Schrödinger equation read

$$H\Psi = i\hbar\frac{\partial}{\partial t}\Psi,$$

the new gauge-transformed Schrödinger equation will read

$$H_\Lambda\Psi_\Lambda = i\hbar\frac{\partial}{\partial t}\Psi_\Lambda.$$

The theorem below states that the gauge transformation is just a picture transformation, and we identify the unitary operator $T(t)$ in this case. Then the physics of the different gauges cannot be different, since picture transformations conserve physical measurements according to Theorem 8.

Theorem 10
Let Ψ be the solution to the time dependent Schrödinger equation with the semiclassical Hamiltonian H. Let Ψ_Λ be the solution to the gauge transformed Schrödinger equation corresponding to the gauge transformed $H \longrightarrow H_\Lambda$. Then Ψ_Λ is related to Ψ by a unitary transformation, viz.,

$$\Psi \longrightarrow \Psi_\Lambda = \exp\left(-\frac{iq}{\hbar c}\Lambda(\boldsymbol{X}, t)\right)\Psi.$$

Furthermore,

$$H_\Lambda(t) = T(t)H(t)T^\dagger(t) + i\hbar\frac{\partial T}{\partial t}T^\dagger(t), \tag{1.31}$$

with

$$T(t) = \exp\left(-\frac{iq}{\hbar c}\Lambda(\boldsymbol{X}, t)\right).$$

In short: The gauge transformation corresponds to a picture transformation.

Proof: We will prove the theorem by working in the position representation, i.e., expressing the state vector in the coordinate basis, because it is the natural choice as Λ is expressed as a function of $\boldsymbol{X}$.

It is sufficient to prove that H_Λ is given by Eqn. (1.31), because then $T(t)\Psi$ is governed by the corresponding Schrödinger equation, and then Ψ_Λ must be given by this unitary transformation.

Consider the picture transformation

$$H_T = THT^\dagger + i\hbar\frac{\partial T}{\partial t}T^\dagger.$$

The first term yields

$$THT^\dagger = \frac{1}{2m}T\left(\boldsymbol{P} - \frac{q}{c}\boldsymbol{A}\right)^2 T^\dagger + qT\phi T^\dagger,$$

where the first part is the kinetic energy. The second term is easy to calculate, since ϕ commutes with T as they are pure functions of $\boldsymbol{X}$. The kinetic energy, however, is more complicated. We consider each term in the kinetic energy when the square is expanded:

$$T\left(\boldsymbol{P} - \frac{q}{c}\boldsymbol{A}\right)^2 T^\dagger = T\left(\boldsymbol{P}^2 - \frac{q}{c}(\boldsymbol{AP} + \boldsymbol{PA}) + \frac{q^2}{c^2}\boldsymbol{A}^2\right)T^\dagger. \tag{1.32}$$

The last $\boldsymbol{A}^2$-term commutes with $T^\dagger$ because it is also a function of $\boldsymbol{X}$. We will then have to compute the two remaining terms. These two terms will, as we shall see, give rise to the terms arising from the gauge transformation.

We calculate the operator $\boldsymbol{P}T^\dagger$ by operating on a L^2-function Ψ:

$$\boldsymbol{P}T^\dagger\Psi = -i\hbar\nabla(T^\dagger\Psi) = -i\hbar\left(\frac{iq}{\hbar c}T^\dagger(\nabla\Lambda)\Psi + T^\dagger\nabla\Psi\right) = \left[T^\dagger\frac{q}{c}(\nabla\Lambda) + T^\dagger\boldsymbol{P}\right]\Psi.$$

Hence,

$$\boldsymbol{P}T^\dagger = T^\dagger\frac{q}{c}(\nabla\Lambda) + T^\dagger\boldsymbol{P}.$$

The middle term in Eqn. (1.32) is then easily calculated:

$$\frac{q}{c}\left(\boldsymbol{P}T^\dagger\boldsymbol{A} + \boldsymbol{A}\boldsymbol{P}T^\dagger\right) = T^\dagger 2\frac{q^2}{c^2}\boldsymbol{A}(\nabla\Lambda) + \frac{q}{c}T^\dagger(\boldsymbol{AP} + \boldsymbol{PA}).$$

For the $\boldsymbol{P}^2$ term we note first that

$$\boldsymbol{P}^2T^\dagger = \boldsymbol{P}(T^\dagger\frac{q}{c}(\nabla\Lambda) + T^\dagger\boldsymbol{P}).$$

The second term yields

$$\boldsymbol{P}T^\dagger\boldsymbol{P} = T^\dagger\frac{q}{c}(\nabla\Lambda)\boldsymbol{P} + T^\dagger\boldsymbol{P}^2,$$

while the first term is found by operating on Ψ, viz.,

$$\frac{q}{c}\boldsymbol{P}(\nabla\Lambda)T^\dagger\Psi = \left[\frac{-i\hbar q}{c}T^\dagger(\nabla^2\Lambda) + \frac{q^2}{c^2}T^\dagger(\nabla\Lambda)^2 + \frac{q}{c}T^\dagger(\nabla\Lambda)\boldsymbol{P}\right]\Psi.$$

Collecting all the terms we obtain

$$\begin{aligned}T(\boldsymbol{P} - \frac{q}{c}\boldsymbol{A})^2T^\dagger = {} & TT^\dagger\boldsymbol{P}^2 - \frac{i\hbar q}{c}TT^\dagger(\nabla^2\Lambda) \\ & + \frac{q^2}{c^2}TT^\dagger(\nabla\Lambda)^2 + 2TT^\dagger\frac{q}{c}(\nabla\Lambda)\boldsymbol{P} \\ & + \frac{q^2}{c^2}TT^\dagger\boldsymbol{A}^2 - 2TT^\dagger\frac{q^2}{c^2}(\nabla\Lambda)\boldsymbol{A} \\ & - \frac{q}{c}TT^\dagger(\boldsymbol{AP} + \boldsymbol{PA})\end{aligned}$$

Using the unitarity of T and reorganizing we get

$$T(\boldsymbol{P}-\frac{q}{c}\boldsymbol{A})^2T^\dagger = \boldsymbol{P}^2 - \frac{q}{c}(\boldsymbol{AP}+\boldsymbol{PA}) + \frac{q^2}{c^2}\boldsymbol{A}^2 + 2\frac{q}{c}(\nabla\Lambda)\boldsymbol{P} - \frac{i\hbar q}{c}(\nabla^2\Lambda) - 2\frac{q^2}{c^2}\boldsymbol{A}(\nabla\Lambda) + \frac{q^2}{c^2}(\nabla\Lambda)^2.$$

Note that

$$(\nabla\Lambda)\boldsymbol{P} = \boldsymbol{P}(\nabla\Lambda) + i\hbar(\nabla^2\Lambda),$$

which we use to transform one of the $(\nabla\Lambda)\boldsymbol{P}$ terms into $\boldsymbol{P}(\nabla\Lambda)$. This cancels the $\nabla^2\Lambda$ term. Then the expression reads

$$\begin{aligned} T(\boldsymbol{P}-\frac{q}{c}\boldsymbol{A})^2T^\dagger &= \boldsymbol{P}^2 - \frac{q}{c}(\boldsymbol{AP}+\boldsymbol{PA}) + \frac{q^2}{c^2}\boldsymbol{A}^2 \\ &\quad + \frac{q}{c}(\nabla\Lambda)\boldsymbol{P} + \frac{q}{c}\boldsymbol{P}(\nabla\Lambda) - 2\frac{q^2}{c^2}\boldsymbol{A}(\nabla\Lambda) + \frac{q^2}{c^2}(\nabla\Lambda)^2 \\ &= \left[(\boldsymbol{P}-\frac{q}{c}\boldsymbol{A}) + \frac{q}{c}(\nabla\Lambda)\right]^2. \end{aligned}$$

The gauge transformed Hamiltonian reads

$$H_\Lambda = \frac{1}{2m}\left(\boldsymbol{P}^2 - \frac{q}{m}(\boldsymbol{A}-\nabla\Lambda)\right) + q\left(\phi + \frac{1}{c}\frac{\partial\Lambda}{\partial t}\right),$$

and we notice that we have successfully reproduced the gauge transformed kinetic energy. The scalar potential has an additional term proportional to $\partial\Lambda/\partial t$, and this comes from the term $(\partial T/\partial t)T^\dagger$ in H_T, viz.,

$$i\hbar\frac{\partial T}{\partial t}T^\dagger = i\hbar\frac{-iq}{\hbar c}\frac{\partial\Lambda}{\partial t}TT^\dagger = \frac{q}{c}\frac{\partial\Lambda}{\partial t}.$$

Thus, we have proved that

$$H_\Lambda = H(\boldsymbol{A}_\Lambda,\phi_\Lambda) = THT^\dagger + i\hbar\frac{\partial T}{\partial t}T^\dagger = H_T,$$

and the proof is complete. ■

We have established that performing a gauge transformation is in fact equivalent to performing a picture transformation. Hence, the gauge parameter Λ has no physical meaning in itself as in classical electrodynamics.

However, the potentials $\boldsymbol{A}$ and ϕ have a physical meaning with somewhat surprising consequences. In classical mechanics a particle moves along a definite trajectory, sampling the electromagnetic fields $\boldsymbol{E}$ and $\boldsymbol{B}$ along the path, i.e., sampling $\nabla\times\boldsymbol{A}$ and $\nabla\phi$. The quantum particle however moves according to a partial differential equation, namely the Schrödinger equation, and thus $\boldsymbol{A}$ and ϕ influence the movement of the particle (i.e., wave) *globally*. Thus in an area of space where the fields vanish and where a classical particle would feel no electromagnetic force, the quantum particle may be affected from non-vanishing potentials, both in the same area of space and other areas of space!

The simplest example of this striking fact is perhaps the so-called *Ahoronov-Bohm effect* occurring in a modified double slit experiment. See Ref. [7] for a treatment.

Gauge invariance is an important concept when we deal with numerical calculations. If our calculations show different results in different gauges, this indicates numerical errors arising in time. Indeed, we have an artificial physical effect of the gauge parameter. If on the other hand our results turn out to be gauge independent, then we have an indication of a highly accurate calculation.

Chapter 2

Simple Quantum Mechanical Systems

In this chapter we will have a look at some analytically solvable quantum mechanical systems. We do this to gain some further insight into the physics described in the previous chapter, and to show that even the simplest quantum mechanical systems are rather difficult to solve. This points in the direction of numerical methods.

We will have a look at the free particle; a particle propagating with no external forces acting on it. We will study the quantum mechanical harmonic oscillator, perhaps the most notorious and important system ever, regardless of physical discipline. The hydrogen atom is the most complicated system we will solve, and the techniques used are common.

More complicated systems can be solved, but it should be clear after solving the hydrogen atom that numerical methods are well justified. In addition, when solving time dependent systems things gets even worse. There are very few systems with time dependent Hamiltonians that can be attacked with analytical means.

In addition to the systems solved here, systems that are important when testing a numerical approach to the time independent Schrödinger equation are considered in chapter 6. These include a particle-in-box in two dimensions, a two-dimensional free particle in a constant magnetic field and a two-dimensional hydrogen atom.

2.1 The Free Particle

We begin with the free particle, which is the simplest spatial problem to solve. The free particle provides the first system comparable to classical Newtonian mechanics and also offers some insight into the behavior of the quantum particle.

2.1.1 The Classical Particle

A free particle is a particle free to move without influence from external forces. The classical Hamiltonian reads

$$H = T = \frac{\boldsymbol{p}^2}{2m},$$

i.e., is it consists of kinetic energy only. Classically, this system has a very simple solution.

Writing out Hamilton's equations of motion (1.2), we get

$$\dot{\boldsymbol{x}} = \frac{\boldsymbol{p}}{m}, \quad \text{and} \quad \dot{\boldsymbol{p}} = 0.$$

Thus, the particle moves with constant momentum $\boldsymbol{p}$ and traces out a straight line (or a single point) in space, viz.,

$$\boldsymbol{x}(t) = \boldsymbol{x}(0) + \frac{\boldsymbol{p}}{m}t.$$

2.1.2 The Quantum Particle

Let us turn to the quantum mechanical system and its solution. Let us first note that the particle may have spin degrees of freedom, but since H commutes with any spin-dependent operator, we need not consider spin at all. Thus,

$$\mathcal{H} = L^2(\mathbb{R}^3).$$

We will use the technique of section 1.4 to solve the Schrödinger equation. We then begin with the time independent Schrödinger equation, which is just the eigenvalue problem for the Hamiltonian, viz.,

$$\frac{\boldsymbol{P}^2}{2m}\psi = E\psi.$$

We have separated the time dependence $e^{-iEt/\hbar}$ and the space dependence $\psi(\boldsymbol{x})$.

Note that the Hamiltonian is proportional to the square of $\boldsymbol{P}$. Thus, any eigenfunction of $\boldsymbol{P}$ is also an eigenfunction for H. Let us try to find eigenfunctions $\psi(\boldsymbol{x})$ of P_i:

$$\frac{\partial}{\partial x_i}\psi = \frac{i}{\hbar}p_i\psi.$$

The solution to this differential equation is easy do deduce:

$$\psi(x_i) = \frac{1}{\sqrt{2\pi\hbar}}e^{\frac{i}{\hbar}p_i x_i},$$

which are the eigenfunctions of the ith component of $\boldsymbol{P}$ with eigenvalue $p_i \in \mathbb{R}$. The factor $(2\pi\hbar)^{-1/2}$ is chosen to achieve orthonormal eigenvectors, in the sense described in appendix A. The function may be scaled by an arbitrary number (that is, independent of x_i). Such a constant may be one of the eigenfunctions corresponding to other directions in space than the ith, viz.,

$$\psi(\boldsymbol{x})_{\boldsymbol{p}} = e^{\frac{i}{\hbar}p_1 x_1}e^{\frac{i}{\hbar}p_1 x_1}e^{\frac{i}{\hbar}p_1 x_1} = e^{\frac{i}{\hbar}\boldsymbol{p}\cdot\boldsymbol{x}}.$$

We have added a subscript to indicate the eigenvalue belonging to the eigenfunction. Note that this function is an eigenfunction of *all* the components of $\boldsymbol{P}$ simultaneously.

For the Hamiltonian we get

$$H\psi_{\boldsymbol{p}} = \frac{1}{2m}(P_1^2 + P_2^2 + P_3^2)\psi_{\boldsymbol{p}} = \frac{1}{2m}\boldsymbol{p}^2\psi_{\boldsymbol{p}},$$

so that the eigenvalues of H become

$$E = \frac{\boldsymbol{p}^2}{2m}, \quad \boldsymbol{p} \in \mathbb{R}^3.$$

Thus, the energies corresponding to $\boldsymbol{p}$ of constant length are the same. The time dependence of $\psi_{\boldsymbol{p}}$ is given by

$$\psi_{\boldsymbol{p}}(t) = \psi_{\boldsymbol{p}}e^{-\frac{i}{\hbar}Et} = \frac{1}{(2\pi\hbar)^{3/2}}e^{\frac{i}{\hbar}\boldsymbol{p}\cdot\boldsymbol{x}}e^{-\frac{i}{\hbar}Et} = \frac{1}{(2\pi\hbar)^{3/2}}e^{i(\boldsymbol{k}\cdot\boldsymbol{x}-\omega t)}, \tag{2.1}$$

where

$$\boldsymbol{k} := \frac{\boldsymbol{p}}{\hbar} \quad \text{and}$$
$$\omega := \frac{E}{\hbar} = \frac{\boldsymbol{p}^2}{2m\hbar} = \frac{\hbar \boldsymbol{k}^2}{2m}.$$

We recognize Eqn. (2.1) as a *plane wave* solution; a wave that travels with constant speed through space with wave number $\boldsymbol{k}$. The speed of the wave is $\boldsymbol{p}/m$, as in the classical case.

The eigenvectors of the momentum operator are plane waves that are free to move with constant velocity in a fixed direction. Compare this to the classical particle moving in a straight line with constant speed. In both cases we know the momentum $\boldsymbol{p}$ sharply – it has no uncertainty. Heisenberg's uncertainty principle (1.12) then states that the position $\boldsymbol{x}$ of the particle in for the quantum mechanical wave should have infinite uncertainty. Indeed, we have:

$$|\psi_{\boldsymbol{p}}(\boldsymbol{x})| = \text{a constant},$$

so that every point in space has the same probability density for finding the particle there upon measurement! Note that the eigenfunctions are not proper vectors of L^2, because they cannot be normalized to get unit norm. As described in appendix A this is because they are instead so-called improper vectors of Hilbert space. Improper vectors are not normalizable to a number, but they are distributions.

Let us discuss arbitrary states of the free particle. We have found the eigenfunctions of H for the free particle, and we know that these constitute a basis for the Hilbert space in question. An arbitrary solution may then be found by superpositioning these, viz.,

$$\Psi(\boldsymbol{x}) = \frac{1}{(2\pi\hbar)^{3/2}} \int \Phi(\boldsymbol{p}) e^{\frac{i}{\hbar}\boldsymbol{p}\cdot\boldsymbol{x}} \, \mathrm{d}^3 p$$

As described in appendix A, we recognize this as the *inverse Fourier transform* of $\Phi(\boldsymbol{p})$. Then we may write

$$\Phi(\boldsymbol{p}) = \frac{1}{(2\pi\hbar)^{3/2}} \int \Psi(\boldsymbol{x}) e^{-\frac{i}{\hbar}\boldsymbol{p}\cdot\boldsymbol{x}} \, \mathrm{d}^3 x,$$

which gives the momentum distribution in terms of the spatial distribution.

It is important to be aware of the fact that the momentum distribution $\Phi(\boldsymbol{k})$ and the spatial distribution $\Psi(\boldsymbol{p})$ are equivalent. They both fully determine the wave function, the difference is that they are described in different bases, i.e.,

$$\Psi(\boldsymbol{x}) = (\psi_{\boldsymbol{x}}, \Psi)$$

and

$$\Phi(\boldsymbol{p}) = (\psi_{\boldsymbol{p}}, \Psi),$$

These expressions are *the components of the vector Ψ along the eigenvector basis for the position and momentum operator,* respectively.

2.1.3 The Gaussian Wave Packet

The plane waves are not particularly suitable for describing a particle as we are used to experience it. Hardly ever do we deal with particles with equal probability of being found at any place in the universe. We need some kind of localization of the particle. Then it is natural to discuss the Gaussian wave packet. We will confine this discussion to one dimension for simplicity and ease of visualization.

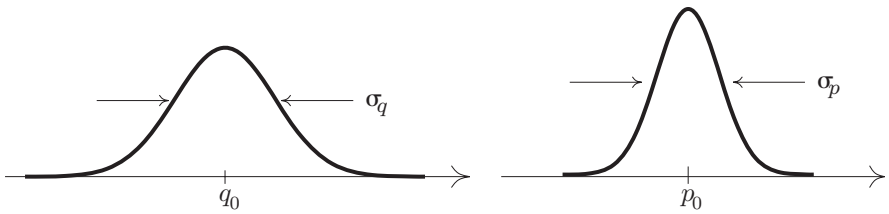

Figure 2.1: Gaussian probability density for position and momentum

The Gaussian wave packet is a spatial wave function whose absolute square is a Gaussian distribution. The wave function is given by

$$\Psi(q) = \frac{1}{(2\pi\sigma_q^2)^{1/4}} e^{ip_0 q/\hbar} \exp\left(-\frac{(q-q_0)^2}{4\sigma_q^2}\right). \tag{2.2}$$

The factor $\exp(ip_0 x/\hbar)$ gives rise to an average momentum p_0, as we shall see in a minute. The probability distribution in space is given by

$$P(q) = |\Psi(q)|^2 = \frac{1}{\sqrt{2\pi\sigma_q^2}} \exp\left(-\frac{(q-q_0)^2}{2\sigma_q^2}\right).$$

Performing an (inverse) Fourier transform of Ψ to obtain the momentum distribution yields

$$\Phi(p) = \frac{1}{(2\pi\sigma_p^2)^{1/4}} e^{-i(p-p_0)q/\hbar} \exp\left(-\frac{(p-p_0)^2}{4\sigma_p^2}\right), \tag{2.3}$$

where $\sigma_p = \hbar/2\sigma_q$ is the width of the momentum distribution which also happens to be a Gaussian distribution. Fig. 2.1 depicts these distributions, i.e., their absolute squares.

Thus, the Gaussian wave packet represents a particle localized around the mean $q = q_0$ with variance σ_q^2. The momentum has mean p_0 and variance $\sigma_p^2 = \hbar^2/4\sigma^2$. Hence

$$\Delta q \Delta p = \sigma_q \sigma_p = \frac{\hbar}{2},$$

and we have achieved optimal uncertainty in q and p in the case of a Gaussian distribution.

With Ehrenfest's theorem we get at once

$$\frac{\mathrm{d}}{\mathrm{d}t}\langle q\rangle = \frac{1}{m}\langle p\rangle$$

and

$$\frac{\mathrm{d}}{\mathrm{d}t}\langle p\rangle = 0.$$

Hence, the wave packet's mean position moves like a classical particle of constant momentum p_0. It can be shown that the time evolution of the free wave packet always assumes the shape of a gaussian.

One may calculate the time evolution of σ_q. The width σ_p is constant in time by Eqn. (1.13), since P^2 commutes with the Hamiltonian. The time evolution of σ_q is given by

$$\sigma_q(t) = \frac{\hbar^2}{4\sigma_p^2}\left(1 + \frac{4\sigma_p^4}{m^2\hbar^2}t^2\right) = \sigma_q(0)\left(1 + \frac{\hbar^2}{4m^2\sigma_q(0)^4}t^2\right). \tag{2.4}$$

Thus, the wave packet spreads out in space as time increases. This phenomenon is called *dispersion*: The different plane wave components of a wave packet have different

velocities (i.e., different phase velocities), so that the group velocity (the velocity of the envelope of the packet) differs from the phase velocity, viz.,

$$v_{\text{phase}} := \frac{\omega}{k} = \frac{p}{2m} \neq \frac{p}{m} = \frac{\mathrm{d}\omega}{\mathrm{d}k} =: v_{\text{group}}.$$

In classical terms we may view the Gaussian wave packet as an ensemble of travelling plane wave particles. Since they have differing velocities their distribution in space must change with time: The faster waves outrun the slower ones. On the other hand, the particles are all free, so that their velocities does not change. Hence, the momentum distribution does not change with time.

The Schrödinger equation is reversible, that is the time dependence of Eqn. (2.4) also holds for *negative* t. This means that it is possible to start with in some respects a more exotic wave packet $\mathcal{U}(-t_0, 0)\Psi$ whose space distribution gets *sharper* before it spreads out again.

2.2 The Harmonic Oscillator

We will now study an ever-returning problem in both classical and quantum mechanics: The harmonic oscillator. The harmonic oscillator is in some respects the most fundamental physical mechanical system, because in very many applications it is a good first approximation. There are dozens of examples of systems performing oscillatory behavior, and in very many of these the harmonic oscillator is a good choice of approximation.

Consider Fig. 2.2 which shows some general one-dimensional potential $V(q)$ with a local minimum at q_0. A Taylor expansion around q_0 yields

$$\begin{aligned} V(q) &\approx V(q_0) + (q - q_0) \left.\frac{\mathrm{d}V}{\mathrm{d}q}\right|_{q_0} + \frac{1}{2} \left.\frac{\mathrm{d}^2V}{\mathrm{d}q^2}\right|_{q_0} (q - q_0)^2 \\ &= V(q_0) + \frac{1}{2} m\omega^2 (q - q_0)^2, \end{aligned}$$

with ω defined through

$$m\omega^2 = \left.\frac{\mathrm{d}^2V}{\mathrm{d}q^2}\right|_{q_0}$$

and where m is the mass of the particle moving in V. This approximation is the harmonic oscillator. (The constant term $V(q_0)$ is usually omitted.)

We will confine our discussion to the one-dimensional oscillator for clarity and simplicity. The full-blown general oscillator is readily obtained in more generalized arguments than those we present in this section.

2.2.1 The Classical Particle

As stated above the Hamiltonian is given by

$$H = \frac{p^2}{2m} + \frac{1}{2} m\omega^2 q^2.$$

Here $\omega \in \mathbb{R}$ is the frequency of oscillation in the classical sense. Sometimes the constant $k = m\omega^2$ is used instead, and one identifies ω after the solution has been found. The coordinate name q is preferred over x in this section, because harmonic oscillators often occur when q is not a cartesian coordinate.

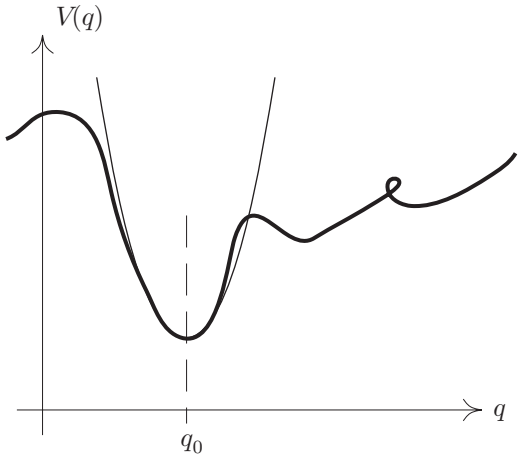

Figure 2.2: Harmonic oscillator approximation

Let us solve the harmonic oscillator with Hamilton's equations (1.2). Writing these out yields

$$\begin{aligned} \dot{q} &= \frac{p}{m} \\ \text{and} \quad \dot{p} &= -\omega^2 q, \end{aligned}$$

and thus

$$\ddot{q} = -\omega^2 q.$$

This is a standard differential equation, and we know that its solution is

$$q(t) = C\cos(\omega t + \phi),$$

where C is the amplitude of the oscillations and ϕ is an arbitrary phase shift. The system thus undergoes periodic oscillations with period $T = 2\pi/\omega$.

From section 1.1 we know that

$$\frac{\mathrm{d}H}{\mathrm{d}t} = \frac{\partial H}{\partial t} = 0,$$

and hence the total energy is conserved for the harmonic oscillator. The situation may be illustrated through Fig. 2.3, which depicts a particle of mass m trapped in the

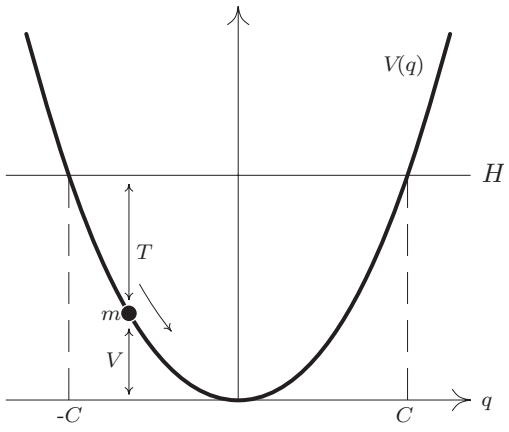

Figure 2.3: The classical harmonic oscillator as a roller-coaster

harmonic oscillator potential; we display it here as sliding on a vertical wire shaped as the potential $V(q)$ which makes a roller-coaster analogy.[1]

Since kinetic energy cannot be negative, the particle cannot be found at q coordinates outside the intersection of the graph of $V(q)$ and the horizontal line of constant total energy $T + V = H$. The region $\mathbb{R} - [-C, C]$ is called the *classically forbidden region*. In quantum mechanics we shall see that the particle may still have a finite probability of being found outside this region.

We have established the fact that a classical particle moving in an harmonic oscillator performs oscillations. Indeed, from the roller-coaster analogy of the system it should be obvious for anyone who has played with small marbles and fruit bowls. When we turn to the quantum mechanical system things get a little bit different. Instead of a clear oscillatory behavior we find discrete eigenstates of the Hamiltonian. These do not "move" in the classical sense, since they are stationary states, but for systems with very high energy (compared to $\hbar\omega$) they correspond to the average behavior of the classical system, as we shall see.

2.2.2 The Quantum Particle

We now have a good understanding of the classical harmonic oscillator. Turning to the quantum mechanical system, it turns out that it is a little bit more complicated to solve. Solving the time independent Schrödinger equation in position representation, that is as a differential equation eigenvalue problem, requires tedious work and knowledge of the so-called Hermite differential equation and the Hermite polynomials. We will instead choose an approach based on operator algebra which makes the solution readily obtainable.

The trick is to define a *lowering operator* a as

$$a := \sqrt{\frac{m\omega}{2\hbar}}Q + i\frac{P}{\sqrt{2m\hbar\omega}}. \tag{2.5}$$

The Hermitian adjoint of a, called the *raising operator* is easily seen to be

$$a^\dagger = \sqrt{\frac{m\omega}{2\hbar}}Q - i\frac{P}{\sqrt{2m\hbar\omega}}. \tag{2.6}$$

The names of the operators will be justified in Theorem 11.

A number x^2+y^2 may be factorized in the complex plane as $(x-iy)(x+iy)$, and this is partly the idea behind the definition of a and $a^\dagger$: Could we write the Hamiltonian as a product in the same way?

We define the Hermitian *number operator* N as

$$N := a^\dagger a.$$

The name will be justified in Theorem 12 below. When we compare H with N, we see that since a and $a^\dagger$ do not commute, we get an extra term proportional to $[Q, P]$, viz.,

$$N = a^\dagger a = \frac{P^2}{2m\hbar\omega} + \frac{m\omega^2}{2\hbar\omega}Q^2 + \frac{i}{\sqrt{2m\hbar\omega}}\sqrt{\frac{m\omega}{2\hbar}}[Q, P] = \frac{1}{\hbar\omega}H - \frac{1}{2}\mathbf{1}.$$

Fortunately this term is just an energy shift which we may safely ignore.[2] Analyzing the spectrum of N is much easier than analyzing H directly, as we will see. We calculate the product in reverse order, viz.,

$$aa^\dagger = \frac{1}{\hbar\omega}H + \frac{1}{2}\mathbf{1},$$

[1] This analogy is perfectly acceptable. The acceleration of a particle sliding on a surface shaped like $V(q)$ is proportional to $V'(q)$ as may easily be deduced.

[2] Eigenvectors are unchanged if we make the transition $H \to H - \sigma\mathbf{1}$, where σ is any scalar.

and this wields the commutator relation

$$[a, a^\dagger] = \mathbf{1}.$$

The following theorem justifies the names of the operators a and $a^\dagger$.

Theorem 11
Let Φ_n be orthonormal eigenvectors of $N := a^\dagger a$ with eigenvalue n. Assume $[a, a^\dagger] = \mathbf{1}$. Then,

$$a^\dagger \Phi_n = \sqrt{n+1}\Phi_{n+1} \tag{2.7}$$
$$\text{and} \quad a\Phi_n = \sqrt{n}\Phi_{n-1}. \tag{2.8}$$

In other words: The operator a transforms Φ_n into Φ_{n-1}, and $a^\dagger$ promotes Φ_n to Φ_{n+1}. The lowering and raising operators produce a whole lot of eigenvectors of N once we are given one of them. A collective term for a and $a^\dagger$ is *ladder operators*. *Proof:* Note that

$$\|a\Phi_n\|^2 = (\Phi_n a^\dagger a \Phi_n) = n \|\Phi_n\|^2,$$

and that

$$\left\|a^\dagger \Phi_n\right\|^2 = (\Phi_n a a^\dagger \Phi_n) = (\Phi, ([a, a^\dagger] + a^\dagger a)\Phi_n) = (n+1)\|\Phi_n\|^2.$$

We investigate the action of N on $a\Phi_n$ and $a^\dagger \Phi_n$:

$$Na\Psi = ([a^\dagger, a] + aa^\dagger)a\Psi = (-1 + aa^\dagger)a\Psi = (n-1)a\Psi$$
$$Na^\dagger\Psi = a^\dagger([a, a^\dagger] + a^\dagger a)\Psi = a^\dagger(\mathbf{1} + N)\Psi = (n+1)a^\dagger\Psi$$

Hence $a\Phi_n$ is an eigenvector of N with eigenvalue $n-1$, and $a^\dagger\Phi_n$ is an eigenvector of N with eigenvalue $n+1$. With the assumption that Φ_n are orthonormal, the result follows at once. ■

We will now prove that the *only* eigenvalues of N are exactly the non-negative integers. Note that Theorem 11 does *not* say this. It could happen that some eigenvalue existed between n and $n+1$. Furthermore, the theorem does not say whether Φ_n are degenerated or not, that is if there exists more than one Φ_n with eigenvalue n.

Theorem 12
Assume $[a, a^\dagger] = 1$. The eigenvalues n of the number operator $N := a^\dagger a$ are integral and non-negative.

Proof: For any $\Psi \in \mathcal{H}$, we define $\Phi = a\Psi$. The norm of Φ is

$$\|\Phi\|^2 = (\Psi, a^\dagger a \Psi) \geq 0,$$

so $a^\dagger a$ is positive definite. (That N is also Hermitian is obvious.)

Assume Φ_n to be an eigenvector of N with eigenvalue n (where n now is not necessarily integral.)

From Theorem 11 we get that repeated use of a on Φ_n yields

$$a^m \Phi_n \propto (n-m)\Phi_{n-m}.$$

But since N is positive definite the eigenvalue cannot be negative. The process must terminate so that

$$n = m,$$

for some integer m. Thus n is integral and non-negative. For the termination to occur, the lowest state must exist and have eigenvalue 0. ■

We still do not know whether *all* eigenvectors may be generated from only Φ_0 by repeated use of $a^\dagger$. It could happen that we missed some eigenvector in case of a degenerate eigenvalue n. However, in one dimension we cannot have degeneracy, such as when applying the theorem to $N = a^\dagger a$ in the present case.[3]

We have now deduced that the Hamiltonian has eigenvalues given by

$$E_n = \hbar\omega(n + \frac{1}{2}), \qquad n = 0, 1, 2, \ldots$$

Hence, the spectrum of the one-dimensional quantum mechanical harmonic oscillator is evenly spaced with spacing $\hbar\omega$.

Let us find the ground state Φ_0, i.e., the state with the lowest energy $E_0 = \hbar\omega/2$). As noted in the proof of Theorem 12, operating with a will annihilate it, viz.,

$$a\Phi_0 = \left(\sqrt{\frac{m\omega}{2\hbar}}Q + i\frac{P}{\sqrt{2m\hbar\omega}}\right)\Phi_0 = 0.$$

This yields a differential equation

$$\frac{\partial\Phi_0}{\partial q} = -\frac{m\omega}{\hbar}q\Phi_0(q),$$

whose solution is easily found by inspection:

$$\Phi_0 = \left(\frac{m\omega}{\pi\hbar}\right)^{1/4} e^{-\frac{m\omega}{2\hbar}q^2}.$$

To produce excited states (that is, states of higher energy than the ground state) of the harmonic oscillator we may operate repeatedly on Φ_0 with $a^\dagger$.

$$\Phi_n = \frac{1}{\sqrt{n!}}(a^\dagger)^n\Phi_0 = \frac{1}{\sqrt{n!}}\left(\sqrt{\frac{m\omega}{2\hbar}}q - \frac{ip}{\sqrt{2m\hbar\omega}}\right)^n\Phi_0$$

We get

$$\begin{aligned}\Phi_n(q) &= \frac{1}{\sqrt{n!}}\left(\frac{\hbar}{2m\omega}\right)^{n/2}\left(\frac{m\omega}{\hbar}q - \frac{\partial}{\partial q}\right)^n\Phi_0 \\ &= \frac{1}{\sqrt{n!}}\left(\frac{m\omega}{\pi\hbar}\right)^{1/4}\left(\frac{\hbar}{2m\omega}\right)^{n/2}\left(\frac{m\omega}{\hbar}q - \frac{\partial}{\partial q}\right)^n e^{-m\omega q^2/2\hbar}\end{aligned}$$

To simplify this expression, we introduce a change of variable from q to x, viz.,

$$x := \sqrt{\frac{m\omega}{\hbar}}q.$$

This yields

$$\Phi_n(x) = \frac{1}{\sqrt{2^n\pi}}\left(\frac{m\omega}{\pi\hbar}\right)^{1/4}\left(x - \frac{\partial}{\partial x}\right)^n e^{-x^2/2}.$$

Let us summarize the results in a theorem. In addition we want to relate our eigenfunctions to the so-called Hermite polynomials $H_n(x)$, see Ref. [22]. A basic relation concerning these is

$$\left(x - \frac{\mathrm{d}}{\mathrm{d}x}\right)^n e^{-x^2/2} = H_n(x)e^{-x^2/2}.$$

Using this, we state our summarizing theorem.

[3] A proof of this can be found in Ref. [7], and rests upon uniqueness of solutions of ordinary differential equations.

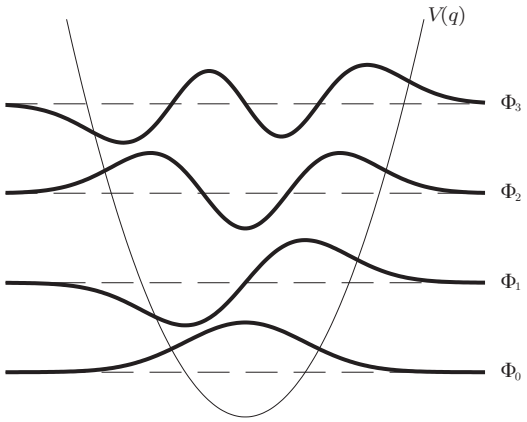

Figure 2.4: Lowest eigenstates of the harmonic oscillator

Theorem 13
The energies of the harmonic oscillator are given by

$$E_n = \hbar\omega(n + \frac{1}{2}),$$

and the corresponding eigenfunctions are given by

$$\begin{aligned}\Phi_n(x) &= \frac{1}{\sqrt{2^n \pi}} \left(\frac{m\omega}{\pi\hbar}\right)^{1/4} \left(x - \frac{\partial}{\partial x}\right)^n e^{-x^2/2} \\ &= \frac{1}{\sqrt{2^n \pi}} \left(\frac{m\omega}{\pi\hbar}\right)^{1/4} H_n(x) e^{-x^2/2},\end{aligned}$$

where $H_n(x)$ are the Hermite polynomials and

$$x = \sqrt{\frac{m\omega}{\hbar}} q.$$

Fig. 2.4 shows the first four eigenfunctions for the harmonic oscillator. The other functions look very similar: An even or odd number of squiggles that fade out at each side.

It is instructive to see what happens with the probability density $|\Phi_n(q)|^2$ as n grows. According to the correspondence principle (see section 2.4) classical behavior should be obtained in the limit of large quantum numbers, that is, for large n. When we say "behavior' in this context, we mean that the probability density should approach that of the classical harmonic oscillator. Let us derive this distribution.

Imagine that the classical harmonic oscillator performs very rapid oscillations. When we measure position, our eyes do not have any chance of catching exactly where it is, so we cannot anticipate the position. Then it becomes clear that the probability density $P(q)\mathrm{d}q$ for finding the particle between q and $q + \mathrm{d}q$ must be given by

$$P(q)\mathrm{d}q = \frac{|dt|}{T/2}.$$

This is because during one half period of oscillation which takes the time $T/2$, the particle will have visited all $q \in [-C, C]$. The time spent between q and $q + \mathrm{d}q$ is $|dt|$. Differentiating $q(t)$, we get

$$\frac{\mathrm{d}q}{\mathrm{d}t} = -C\omega \sin(\omega t) = -\omega\sqrt{C^2 - q(t)^2}.$$

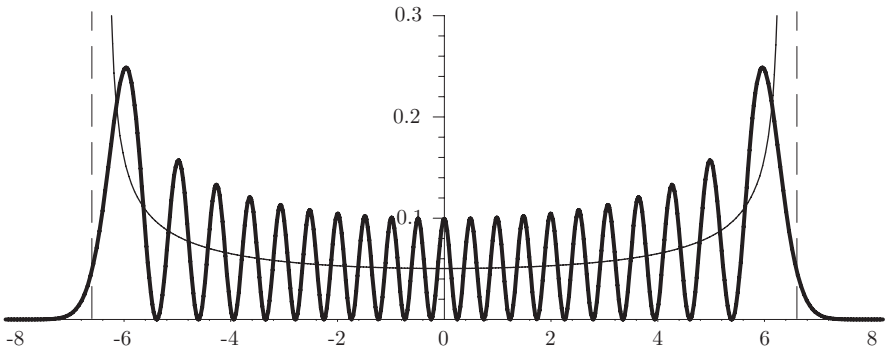

Figure 2.5: Probability density for $n = 20$ together with classical probability. Dimensionless units are applied. Horizontal unit is $(\hbar/m\omega)^{1/2}$ and vertical unit is $(m\omega/\hbar)^{1/2}$. Note the dashed lines that mark the borders of the classically allowed regime.

Thus,

$$|\mathrm{d}t| = \frac{\mathrm{d}q}{\omega\sqrt{C^2 - q(t)^2}}.$$

Remembering that $T = 2\pi/\omega$ we get

$$P(q)\mathrm{d}q = \frac{\mathrm{d}q}{\pi\sqrt{C^2 - q^2}} \tag{2.9}$$

for the probability density.

In Fig. 2.5 we see the classical probability density along with the probability density for $n = 20$. The probability density clearly approaches that of the classical oscillator as we hoped. As seen in Fig. 2.3 in the discussion of the classical harmonic oscillator, q is classically confined to the region $[-C, C]$. On the other hand we see that the probability density $|\Psi_{20}(q)|^2$ is *nonzero* outside this region for the quantum mechanical system. This always happens in quantum systems, when V is finite: There is always some probability that the particle will be found in the classically forbidden region dictated by the total energy H of the bound system.

This is an example of the characteristic phenomenon in quantum mechanics called *tunnelling*. Quantum particles tunnel through classically forbidden regions.

2.3 The Hydrogen Atom

The hydrogen atom is a two-body problem in which an electron orbits a proton. Classically the forces are given by the Coulomb force which is attractive for a system of two opposite charges.

To study the hydrogen atom is interesting in several ways. It is one of the few analytically solvable three-dimensional problems in quantum mechanics, and finding the structure of the eigenstates of a non-trivial system could be illuminating.

In 1913 Niels Bohr proposed a modified version of Rutherford's hydrogen atom in which quantization principles were incorporated (though in a rather *ad hoc* manner) to get rid of certain classical inconsistencies. Rutherford's model assumed a massive and dense nucleus of positive charge and classically orbiting electrons. (See Refs. [10, 11].) Orbiting classical electrons are accelerating. Accelerating charges emit radiation according to Maxwell's equations, and hence loss of energy is inevitable. The orbit is not stable, making the electron eventually collide with the heavy nucleus. Indeed, the large charge-to-mass ratio of the electron makes the lifetime of the atom rather short,

viz.,

$$\tau \approx 1.5 \cdot 10^{-11} \text{ s}.$$

See Ref. [23] for a discussion of the classical instability of the orbiting electron.

Bohr proposed a model in which the electron was allowed to orbit only in circular paths whose angular momentum was an integral multiple of $\hbar$, viz.,

$$L = n\hbar.$$

(For this reason $\hbar$ is also called *the quantum of action*.) The paths were still classical, as $\boldsymbol{x}$ and $\boldsymbol{p}$ were well-defined quantities. In addition to the quantization hypothesis Bohr assumed that radiation did not occur during orbital motion; only in *transitions* between different orbits. In that case, photons of energy equal to the energy difference between orbits were emitted or absorbed.

However unsatisfactory for various reasons, Bohr's model featured several important ideas, among these quantization of energy for a material system, interaction with a quantized electromagnetic field and excellent correspondence with experimental data.[4]

To define the problem at hand, assume that the nucleus, i.e., a proton with mass m_p, is at rest at the origin in $\mathbb{R}^3$. Let the electron with mass m_e be located at the coordinates $\boldsymbol{r}$ relative to the proton.

As the proton mass is much higher than the electron mass, viz.,

$$\frac{m_p}{m_e} \approx 1836,$$

we may neglect the motion of the proton on classical grounds. The large mass ratio leads to small errors. However, a two-body problem such as the hydrogen atom is best studied in the center of mass reference frame, i.e., the moving frame in which the center of mass is at rest. Describing the electron's position relative to the center is equivalent to using the so-called reduced mass μ instead of the electron mass in the equations of motion for the electron. Instead of neglecting the proton motion in our equations we may instead use the center of mass description, and thus improve or description both theoretically and numerically.

The center of mass Hamiltonian is identical to the original proton-neglecting Hamiltonian, but with the electron mass replaced with the reduced mass μ. In other words we will make the substitution

$$m_e \longrightarrow \mu := \frac{m_e \cdot m_p}{m_e + m_p}.$$

See for example Refs. [10, 22] for a thorough discussion of the center of mass system and the Hydrogen atom.

The Coulomb force is a central force, i.e., its line of action is the line joining the proton and the electron. Its value depends only on the distance between the particles. Such a conservative force may be expressed as a gradient, viz.,

$$F = -\nabla V(r),$$

in which the potential V only depends on r, the distance between the particles, i.e. the distance from the origin.

For the Coulomb force we have

$$V(r) = -ke^2 \frac{1}{r}, \tag{2.10}$$

[4]Notice that quantization of light was proposed *before* non-relativistic quantum mechanics even though photons belong to a super-theory incorporating relativity.

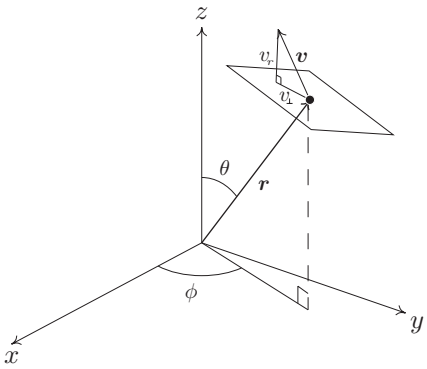

Figure 2.6: Spherical coordinates

where e is the fundamental charge and k is a constant whose value is such that

$$ke^2 \approx 1.44 \text{ eV} \cdot \text{nm}.$$

Spherical symmetry suggests that we use spherical coordinates in our description, viz.,

$$\begin{aligned} x &= r\cos\phi\sin\theta \\ y &= r\sin\phi\sin\theta \\ z &= r\cos\theta \end{aligned}$$

See Fig. 2.6 for a visualization.

We will study the general case with arbitrary potentials to emphasize that the methods are applicable to other systems as well, such as the three dimensional harmonic oscillator.

2.3.1 The Classical System

We will consider the classical system only briefly to provide means for comparing the quantum mechanical system to the classical system.

The Hamiltonian of the problem at hand is given by

$$H = \frac{\boldsymbol{p}^2}{2\mu} + V(r).$$

Let us find a constant of motion for such a system, namely the angular momentum $\boldsymbol{L}$. The torque on the particle is zero, viz.,

$$\boldsymbol{\tau} = \boldsymbol{r} \times \boldsymbol{F} = 0,$$

because the $\boldsymbol{F} = -\nabla V$ is parallel to $\boldsymbol{r}$. Thus, angular momentum is conserved, viz.,

$$\frac{\mathrm{d}\boldsymbol{L}}{\mathrm{d}t} = \boldsymbol{v} \times (m\boldsymbol{v}) + \boldsymbol{r} \times \boldsymbol{F} = 0.$$

We may decompose the velocity into a radial part $\boldsymbol{v}_r$ parallel to $\boldsymbol{r}$ and a part $\boldsymbol{v}_\perp$ in the plane orthogonal to $\boldsymbol{r}$, see Fig. 2.6. Then it is easily seen that

$$L = \mu r v_\perp,$$

and thus the Hamiltonian may be rewritten as

$$H = \frac{p_r^2}{2\mu} + \underbrace{\frac{L^2}{2\mu r^2} + V(r)}_{V_{\text{eff}}(r)}. \tag{2.11}$$

This is the Hamiltonian of a one-dimensional system in which r is the (generalized) coordinate and where $V_{\text{eff}}(r)$ is the potential.

Note that since $\partial H/\partial t = 0$ the total energy H is conserved in this system. It can be shown that the motion of the electron in the Coulomb potential lies on a conic section, depending on the total energy H: For $H < 0$ the motion traces an ellipse, for $H = 0$ the motion traces a parabola and for $H > 0$ the trace is a hyperbola. Thus all systems in which $H < 0$ yields bound motion with bounded conic sections, and $H \geq 0$ yields unbound motion.

2.3.2 The Quantum System

The derivation in this section will not be complete. We include only the most important aspects of the method and leave it to the reader to fill in the details referred to. A good account is given in Ref. [22], and is recommended for further reading.

Turning to the Hamiltonian again, we have in the position representation

$$H = -\frac{\hbar^2}{2\mu}\nabla^2 + V(r),$$

and we need the Laplacian in spherical coordinates. This is given by

$$\nabla^2 = \frac{\partial^2}{\partial r^2} + \frac{2}{r}\frac{\partial}{\partial r} + \frac{1}{r^2}\left[\frac{\partial^2}{\partial\theta^2} + \cot\theta\frac{\partial}{\partial\theta} + \frac{1}{\sin^2\theta}\frac{\partial^2}{\partial\phi^2}\right]$$

as can be looked up in for example Ref. [24]. The expression in brackets turns out to be the quantum mechanical orbital momentum operator L^2 as described in section 1.5.5.[5] Thus, our Hamiltonian reads

$$H = -\frac{\hbar^2}{2\mu}\left(\frac{\partial^2}{\partial r^2} + \frac{2}{r}\frac{\partial}{\partial r}\right) + \frac{L^2}{2\mu r^2} + V(r). \tag{2.12}$$

The first term is just the radial kinetic energy, viz.,

$$p_r^2 = -\hbar^2\left(\frac{\partial^2}{\partial r^2} + \frac{2}{r}\frac{\partial}{\partial r}\right).$$

We see from Eqn. (2.12) that L^2 commutes with H, that is we may find a set of simultaneous orthonormal eigenvectors to both operators. (Thus L^2 is also a quantum mechanical constant of motion according to Theorem 4, section 1.5.4.) Since L^2 commutes with the angular momentum components L_i, these are also constants of motion.

We know that the eigenstates of L^2 and L_z are parameterized with two integral (or half-integral) quantum numbers l and m. For the orbital angular momentum it turns out that l may take on all integral values and no half-integral values. To be more specific, the eigenfunctions $Y_{lm}(\theta,\phi)$ of L^2 and L_z are the so-called *spherical harmonics*; a basis for the square-integrable functions defined on a sphere. For a discussion of these functions see for example Refs. [10, 22].

We may try separation of variables to obtain the eigenstates Ψ_{nlm} of H. Let us write

$$\Psi_{nlm} = R(r)Y_{lm}(\theta,\phi).$$

This works perfectly well, and upon insertion into the time independent Schrödinger equation (1.7) we end up with the equation

$$\frac{\partial^2}{\partial r^2}(rR) + \left\{\frac{2\mu}{\hbar^2}[E - V(r)] - \frac{l(l+1)}{r^2}\right\}(rR) = 0.$$

[5] Given the differential operators in Cartesian coordinates, it is no difficult task (albeit tedious) to calculate the spherical coordinate version of L_i.

The auxiliary function $u = rR(r)$ therefore satisfies a modified time independent Schrödinger equation called *the radial equation*, viz.,

$$-\frac{\hbar^2}{2\mu}\frac{\partial^2 u}{\partial r^2} + \underbrace{\left[V(r) + \frac{\hbar^2 l(l+1)}{2\mu r^2}\right]}_{V_{\text{eff}}(r)} u(r) = Eu(r). \tag{2.13}$$

This is clearly the quantum mechanical counterpart of Eqn. (2.11).[6] The extra term in the effective potential V_{eff} is called *the centrifugal term*. We see that for different l we have different radial equations and hence different solutions $R(r)$. We expect therefore that R must be labelled both with n and l, the nature of the former may be discrete or continuous. It turns out that n is discrete for $E < 0$ and continuous otherwise.

We may at once devise some constraints on $u(r)$ for the actual solution Ψ to be physical (or rather mathematical). First of all we must have

$$u(0) = 0,$$

because we want $R(r) = u(r)/r$ not to be divergent. Second, if we let r approach infinity, then the radial equation becomes

$$\frac{\partial^2 u}{\partial r^2} \sim -\frac{2\mu E}{\hbar^2}u(r).$$

For $E < 0$ the only acceptable solution is

$$u(r) \sim e^{-\alpha r},$$

where $\alpha = \sqrt{-2\mu E\hbar^{-2}}$. If we on the other hand let r approach zero, and assume that $V(r) = \mathcal{O}(r^{-k})$ with $k < 2$, the centrifugal term will dominate the radial equation, giving

$$\frac{\partial^2 u}{\partial r^2} \sim \frac{l(l+1)}{r^2}u(r).$$

The acceptable solution giving bounded $R(r)$ in this case is

$$u(r) \sim r^{l+1},$$

or

$$R(r) \sim r^l, \quad r \text{ small}.$$

Let us summarize what we have found so far in a theorem.

Theorem 14
Assume that we have a central symmetric Hamiltonian on the form

$$H = \frac{\boldsymbol{p}^2}{2\mu} + V(r)$$

where

$$V(r) = \mathcal{O}(r^{-k}), \quad k < 2.$$

Then the solutions to the time independent Schrödinger equation are on the form

$$\Psi(r, \theta, \psi) = R(r)Y_{lm}(\theta, \phi),$$

[6] The correspondence between the classical radial equation and the quantum mechanical equation is a mathematical coincidence. As shown in Ref. [25] it is for example not true in two dimensions.

where Y_{lm} are the spherical harmonics and where $R(r)$ satisfies the radial equation

$$-\frac{\hbar^2}{2\mu}\frac{\partial^2(rR)}{\partial r^2}+\left[V(r)+\frac{\hbar^2 l(l+1)}{2\mu r^2}\right](rR)=E(rR).$$

Furthermore, the asymptotic behavior of $R(r)$ for bound states (i.e., $E<0$) is

$$\begin{aligned} R(r) &\sim r^l, \quad r \text{ small}, \\ \text{and} \quad R(r) &\sim \exp\left(-\sqrt{-2\mu E\hbar^{-2}}r\right), \quad r \text{ large}. \end{aligned}$$

We will now consider the Coulomb potential (2.10) in particular. We will consider bound states, states in which $E<0$. These are exactly the states that classically traces out a bounded orbit. We expect these to remain bounded, that is the solutions to the time independent Schrödinger equation are expected to be square integrable.

First we rewrite Eqn. (2.13) on dimensionless form, introducing the scalings

$$\rho = r\sqrt{-8\mu E/\hbar^2} \tag{2.14}$$

$$\text{and} \quad \lambda = ke^2\sqrt{\frac{\mu}{-2E}}. \tag{2.15}$$

This yields

$$\frac{\partial^2 u}{\partial\rho^2}+\left[\frac{\lambda}{\rho}-\frac{l(l+1)}{\rho^2}-\frac{1}{4}\right]u(\rho)=0. \tag{2.16}$$

The strategy uses Theorem 14: We try to incorporate the asymptotic behavior. For large r we have

$$u(\rho)\sim e^{-\rho/2},$$

and therefore we write

$$u(\rho)=e^{-\rho/2}v(\rho).$$

Inserting this into the new radial equation yields a new differential equation for $v(\rho)$, viz.,

$$\frac{\partial^2 v}{\partial\rho^2}-\frac{\partial v}{\partial\rho}+\frac{\lambda}{\rho}v-\frac{l(l+1)}{\rho^2}v=0.$$

We will not solve this equation explicitly as the arguments are a bit lengthy. See for example Ref. [22]. However, the acceptable solutions turn out to be polynomials of degree $\lambda-1$, thereby imposing a quantization on the energy E through Eqn. (2.15). We define the *principal quantum number* $n=\lambda$, and we have

$$n\geq l+1,$$

so that for each n, the orbital momentum quantum number is limited. Note that the radial function R depends on both n and l, but that the energy E *only* depends on n. For an arbitrary spherical symmetric potential we would expect a dependence on l as well.

We summarize the properties of the radial solution $R(r)$ in a theorem, which we leave without further proof.

Theorem 15
The hydrogen atom's radial wave function is on the form

$$R_{nl}(r)=C\rho^l e^{-\rho/2}L_{n+l}^{2l+1}(\rho),$$

where ρ is given by

$$\rho=r\cdot\frac{2}{a_0 n}, \qquad a_0:=\frac{\hbar^2}{ke^2\mu},$$

where a_0 is called the Bohr radius. *C is a normalization constant. The energy is given by*

$$E_n = -\frac{(ke^2)^2\mu}{2\hbar^2}\frac{1}{n^2}.$$

The function $L_{n+l}^{2l+1}(\rho)$ is the $2l+1$th associated polynomial of the $n+l$'th Laguerre polynomial, and it has degree $n-l+1$.

We have now found all the solutions to the time independent Schrödinger equation, viz.,

$$\Psi_{nlm}(r,\theta,\phi) = R_{nl}(r)Y_{lm}(\theta,\phi).$$

Visualizing these three dimensional functions is not very easy, but in Ref. [26] there is a whole chapter devoted to this.

This concludes our exploration of the hydrogen atom. There are however some features of the solution to notice which aids us with a general understanding of bounded states in three dimensions:

- The stationary states, i.e. the wave functions Ψ_{nlm}, and the energy eigenvalues are indexed by three *quantum numbers*. We choose a numbering such that higher quantum numbers yield higher energy. See also section 2.4 on the correspondence principle.
- There are an infinite number of bound states with $E < 0$ for the hydrogen atom. This is not a general fact. There are potentials without even one bound state even though the potential has a global minimum.
- We also have eigenstates for the hydrogen atom with energy $E > 0$. Classically, these are not bound, and this is also the case for the quantum mechanical system: The eigenstates of H turn out to be non-normalizable, just as the eigenstates of the free particle. For the unbound states the energy spectrum is continuous, contrary to the bound states.
- As for the one-dimensional harmonic oscillator there is a finite probability of finding the particle *at arbitrary large* r, thus violating the strict boundaries of the classically allowed regime.
- The energies obtained are actually identical to those found by Bohr in his atomic model.
- For large n, the energy difference $E_n - E_{n-1}$ approach zero, making in a sense a classical limit in which the energy is continuous. This is an example of the *correspondence principle*, which Bohr proposed for his quantum theory. We will say more on this in section 2.4.
- As we have explained, the square magnitude $|\Psi(\boldsymbol{x})|^2$ of the wave function is interpreted as the probability density of locating the electron at $\boldsymbol{x}$ upon measurement. In our case we have

  $$|\Psi_{nlm}(r,\theta,\phi)|^2 = R_{nl}(r)^2|Y_{lm}(\theta,\phi)|^2.$$

  Since $|Y_{00}(\theta,\phi)| = 1$, we get for the $l = m = 0$ states

  $$|\Psi_{nlm}(r,\theta,\phi)|^2 = R_{n0}(r)^2 = C^2e^{-\rho}[L_{n+1}^1(\rho)]^2.$$

  It is $|\Psi(\boldsymbol{x})|^2\mathrm{d}x\mathrm{d}y\mathrm{d}z$ that measures the probability as opposed to the probability density. With spherical coordinates we have

  $$|\Psi(\boldsymbol{x})|^2\mathrm{d}x\mathrm{d}y\mathrm{d}z = |\Psi(r,\theta,\phi)|^2r^2\mathrm{d}r\mathrm{d}\theta\mathrm{d}\phi,$$

and thus

$$|\Psi_{nlm}(r,\theta,\phi)|^2 r^2 \mathrm{d}r = R_{n0}(r)^2 r^2 \mathrm{d}r = C' r^2 e^{-\rho}[L^1_{n+1}(\rho)]^2 \mathrm{d}r$$

where C' is a normalization constant is the probability density of finding the electron at a distance r from the nucleus.

Recall that $\rho = 2r/a_0 n$, and thus the exponential term falls off slower with higher energy (i.e., higher n). Without analyzing the Laguerre-polynomial term further, it is reasonable to accept that the electron's average distance from the nucleus increases with energy, a feature expected from Bohr's model. Indeed, it turns out that $\langle r \rangle$ is identical to the model's postulated quantized radii.

2.4 The Correspondence Principle

After analyzing a few simple quantum mechanical systems we are in position to state *the correspondence principle*, first put forward by Bohr. The principle is quite simple: In the limit of high quantum numbers classical behavior should be reproduced. Alternatively, we may take the limit $\hbar = 0$ as the energies always will contain Planck's constant as a factor.

If we for instance consider the harmonic oscillator, the energies are given by

$$E_n = \hbar\omega(n + \frac{1}{2}),$$

and in letting $\hbar \to 0$ we must compensate with letting $n \to \infty$ to keep the energy of the system fixed.

As seen in Fig. 2.5 on page 45, the quantum mechanical probability density is approaching the classical density in the limit $n \to \infty$,[7] equivalently if $\hbar \to 0$ and the energy E is kept constant.

The principle simply expresses that quantum mechanics is supposed to be a super-theory for Newtonian mechanics in the sense that everything obeying Newton's laws is supposed to also obey non-relativistic quantum mechanics, but with the non-classical features of the dynamics obscured by the fact that $\hbar$ is very small compared to macroscopic quantities.

On the other hand, if the correspondence principle is violated for some system, then we may find a macroscopic realization of a system in which quantum effects is visible.

[7] We must choose a numbering of the energy eigenvalues such that energy is increasing with higher quantum numbers.

Chapter 3

The Time Dependent Schrödinger Equation

This chapter is a detailed description of the time dependent Schrödinger equation for a single particle in a classical electromagnetic field. This is the domain of the systems we wish to study with our numerical methods.

3.1 The General One-Particle Problem

We study the quantum description of a single particle. We shall in general let μ be the mass of the particle. In addition to existing in (at most) three dimensions the particle also has spin s. Hence, the wave function for our particle is a square integrable function of (at most) three real variables (i.e., $\Omega \subset \mathbb{R}^d$, $d \leq 3$) into the complex vector space $\mathbb{C}^n$, with $n = 2s + 1$. For each point in Ω, the wave function has n complex components. We denote each component function by $\Psi^{(\sigma)}$, where $\sigma = 1, \ldots, n$.

We let $\boldsymbol{S}$ be the spin operators S_i for the particle. In our chosen representation these are $n \times n$ matrices with components $S_i^{(\sigma,\tau)}$. We take the standard basis vectors to be eigenvectors of S_3 (i.e., the spin along the z-axis) with increasing eigenvalues. Hence, $|\Psi^{(\sigma)}(\boldsymbol{x}, t)|^2$ is the probability density of finding the particle at $\boldsymbol{x}$ at time t with a spin eigenvalue $\hbar m = \hbar(\sigma - 1 - s)$ along the z-axis.[1]

The particle will in general have charge q and move in an electromagnetic field, i.e., in an electric field $\boldsymbol{E}$ and a magnetic field $\boldsymbol{B}$. Equivalently, the fields are described by the potentials $\boldsymbol{A}$ and ϕ, as described in section 1.6.

A classically spinning particle gains an additional potential energy $-\Gamma \boldsymbol{B} \cdot \boldsymbol{S}$ from its rotational motion, where $\Gamma = qg/2\mu c$ is the scaled gyromagnetic factor. For an electron we have $g \approx 2$.[2] For a quantum particle we assume the same formal expression for the potential energy of a spin, and thus we *do* in a way envisage the electron as spinning, as described in section 1.5.6. In fact, the coupling with $\boldsymbol{B}$ is the *only* observable effect of the spin of the particle. We call this the Zeemann effect.[3]

The full-blown Hamiltonian of this system then reads

$$H = \frac{1}{2\mu}\left(-i\hbar\nabla - \frac{q}{c}\boldsymbol{A}(\boldsymbol{x}, t)\right)^2 + q\phi(\boldsymbol{x}, t) + V_{\text{ext}}(\boldsymbol{x}, t) - \Gamma \boldsymbol{B}(\boldsymbol{x}, t) \cdot \boldsymbol{S}. \tag{3.1}$$

[1] As $m = -s \ldots s$, $\sigma = m + s + 1 = 1 \ldots 2s + 1$.

[2] This may be determined experimentally or by means of relativistic corrections to the quantum theory.

[3] See Ref. [22].

Note that all but the last term commute with S_i, and it becomes natural to write

$$H = H_{\text{spatial}} + H_{\text{spin}},$$

with $H_{\text{spin}} = -\Gamma \boldsymbol{B} \cdot \boldsymbol{S}$. However, H_{spin} does not commute with neither the momentum nor the position operator due to the magnetic field which may vary in space. For each component $\Psi^{(\sigma)}$ of the quantum state we have the equation

$$i\hbar \frac{\partial \Psi^{(\sigma)}}{\partial t} = H_{\text{spatial}} \Psi^{(\sigma)} - \Gamma \sum_{\tau} \boldsymbol{B} \cdot \boldsymbol{S}^{(\sigma,\tau)} \Psi^{(\tau)}. \tag{3.2}$$

Clearly, H_{spin} couples the differential equations for each component wave function. If $\boldsymbol{B} = 0$ this coupling vanishes and the equations become identical. Hence, there is no need for implementing a simulator for coupled PDEs in this case. In the case of a constant (in both space and time) magnetic field $\boldsymbol{B}$ we may align it along the z-axis, and this makes H_{spin} diagonal and again the equations become decoupled.

Eqn. (3.2) is the most general problem we would wish to solve numerically. There are however many special cases yielding considerable simplifications to the problem such as the above-mentioned decoupling of the PDEs for $\Psi^{(\sigma)}$ or gauge-transformations of the potentials.

As discussed in section 1.6 we may perform a unitary transformation on the form

$$T = \exp\left(-\frac{iq}{\hbar c}\Lambda(\boldsymbol{x}, t)\right),$$

which is equivalent to the gauge transformation

$$\boldsymbol{A} \longrightarrow \boldsymbol{A}_\Lambda = \boldsymbol{A} - \nabla\Lambda, \quad \phi \longrightarrow \phi_\Lambda = \phi + \frac{1}{c}\frac{\partial \Lambda}{\partial t}.$$

The transformed quantum state $\Psi_\Lambda = T\Psi$ obeys the Schrödinger equation obtained by substituting the potentials with the transformed potentials. (Theorem 10, section 1.6, which also holds for the case in which spin is incorporated because T commutes with the spin operators $\boldsymbol{S}$.)

We can of course not eliminate $\boldsymbol{A}$ or ϕ in general and in this way obtain a simpler description of our particle. But in several circumstances, such as in the dipole approximation discussed below, some gauges have advantages over others.

In the below discussions of different magnetic fields it is important to bear in mind that for the potentials $\boldsymbol{A}$ and ϕ to have any meaning they *must represent fields consistent with Maxwell's equations* (1.23). For example, a magnetic field depending on time alone is not consistent.[4] We discuss different kinds of magnetic fields, such as a uni-directional field, but it may happen that they are inconsistent. In that case we must be somewhat careful in our manipulations of the fields and potentials. We must have some kind of justification for using such a field. One such justification is that our system is small compared to the spatial variations of the field so that we may neglect the spatial dependence of the fields.

3.1.1 Uni-Directional Magnetic Field

A uni-directional magnetic field is a field given by

$$\boldsymbol{B} = B(\boldsymbol{x}, t)\boldsymbol{n}(t),$$

[4] It is easily seen that $\boldsymbol{B} = \boldsymbol{B}(t)$ implies that $\boldsymbol{B} = \text{const}$ is a contradiction through Maxwell's equations. $\boldsymbol{B} = \boldsymbol{B}(t)$ implies $\boldsymbol{E}_t = c\nabla \times \boldsymbol{B} = 0$, which again implies $\boldsymbol{E} = \text{const}$. This in turn implies $\boldsymbol{B}_t = -c\nabla \times \boldsymbol{E} = 0$, and finally $\boldsymbol{B} = \text{const}$.

where $\boldsymbol{n}(t)$ is a unit vector dependent on time only. The field strength B may vary in space as well. Hence, the *direction* of $\boldsymbol{B}$ only varies with time.

Assume that the unit normal is independent of time. If the magnetic field $\boldsymbol{B}$ is directed along the z-axis (if not, we just rotate the frame of reference) we note a considerable simplification in the Schrödinger equation (3.2). Let $\boldsymbol{B} = B(\boldsymbol{x}, t)\hat{k}$, where $\hat{k}$ is the unit vector in the z-direction. Then

$$H_{\text{spin}} = -\Gamma B(\boldsymbol{X}, t) S_3,$$

and Eqn. (3.2) becomes

$$i\hbar \frac{\partial \Psi^{(\sigma)}}{\partial t} = H_{\text{spatial}} \Psi^{(\sigma)} - \Gamma B(\boldsymbol{X}, t) S_3^{(\sigma,\sigma)} \Psi^{(\sigma)}. \tag{3.3}$$

Hence, the different equations are no longer coupled due to S_3 being diagonal.

We may not do this if $\boldsymbol{n}$ varies with time because the spin-diagonal Schrödinger equation (3.3) is not valid at all times. One could imagine an attempt at diagonalizing H_{spin} with an operator T defining a picture transformation. The columns of T would be the eigenvectors of $\boldsymbol{n} \cdot \boldsymbol{S}$, and

$$H_T = H_{\text{spatial}} - \Gamma B S_3 + i\hbar \frac{\partial T}{\partial t} T^\dagger,$$

but the last term will not in general be diagonal. There is no easy way to decouple the PDEs for $\Psi^{(\sigma)}$ whenever $\boldsymbol{n}$ varies with time. If $\boldsymbol{B}$ is allowed to vary even more arbitrarly the diagonalization of H_{spin} of course also breaks down.

Since the diagonal elements of S_3 are just real numbers, each PDE differs only by a scaling of the magnetic field. Thus, each component $\Psi^{(\sigma)}$ evolves like the other components except for a magnetic field of differing strength. For example, if $s = 1/2$ we have

$$S_3^{(1,1)} = -\frac{\hbar}{2} \quad \text{and} \quad S_3^{(2,2)} = +\frac{\hbar}{2},$$

and thereby

$$i\hbar \frac{\partial \Psi^{(\sigma)}}{\partial t} = H_{\text{spatial}} \Psi^{(\sigma)} \pm \frac{\hbar}{2} \Gamma B(\boldsymbol{x}, t) \Psi^{(\sigma)}.$$

If B in addition is independent of $\boldsymbol{x}$ (i.e., a dipole-approximation; see below), then the PDEs become even simpler. In fact, we may integrate each PDE analytically, if we know the eigenvectors of H_{spatial}.

It is easy to see that if $H = H_0 + f(t)$, where a basis Ψ_n of eigenvectors of H_0 are known and where $f(t)$ is a scalar function of time alone, then Ψ_n are also eigenvectors of H. The time development of each eigenvector is given by

$$\Psi_n(t) = \exp\left(-i(E_n t + \int_0^t f(t')\, \mathrm{d}t') \right) \Psi_n(0),$$

where E_n are the eigenvalues of H_0, and hence we can calculate the time development of any linear combination of the Ψ_ns. In fact, if $\Psi = \sum_n c_n \Psi_n$, then by linearity of the evolution operator we get

$$\Psi(t) = e^{-i\int_0^t f(t')\, \mathrm{d}t'} \sum_n e^{-iE_n t} \Psi_n.$$

In other words; the wave function differs from the one found by solving the time dependent Schrödinger equations with $H = H_0$ only by a phase factor. This means that it is only necessary to solve one PDE in the uni-directional case with a time dependent homogenous magnetic field. Note that we must require the direction of the field to be constant in time in order to de-couple the PDEs.

3.1.2 A Particle Confined to a Small Volume

Imagine a particle confined in a very small volume, i.e., approximately to a point in space. The particle is then approximately in an eigenstate of both the position operator and the momentum operator. The position is $\boldsymbol{x}_0$ and the momentum is zero.[5] Then we may take

$$H = H_{\text{spin}} = -\Gamma \boldsymbol{B}(\boldsymbol{x}_0, t) \cdot \boldsymbol{S}$$

as our Hamiltonian and ignore the spatial dependence altogether, leaving us with an element of $\mathbb{C}^n$ whose components are $\Psi^{(\sigma)}(\boldsymbol{x}_0)$ as the full quantum description of the particle.

In this case it is not necessary to solve a PDE; indeed the Schrödinger equation becomes an ODE. Note that we may choose the spin s to be arbitrary, making arbitrary large systems of ODEs. This system may be a good test case for ODE integrators that should preserve qualitative features of the full Schrödinger equation.

There are also physically interesting systems of this type which are analytically solvable, in which a time dependent magnetic field represents an explicit time dependence in the Schrödinger equation. For example, if

$$\boldsymbol{B}(t) = B_0(a\hat{k} + b(\cos(\omega t)\hat{i} + \sin(\omega t)\hat{j})) = B_0 \boldsymbol{n}(t),$$

the spin state performs so-called Rabi oscillations. This magnetic field rotates with constant angular frequency ω around the z-axis. The Hamiltonian reads

$$H = -\Gamma \boldsymbol{B}(t) \cdot \boldsymbol{S}.$$

This system may be solved analytically for spin-1/2 particles, i.e., for a two-dimensional ODE; see Ref. [27]. This is one of very few solvable time-dependent quantum mechanical problems. For different angular frequencies ω the Rabi oscillations display resonances, and such systems are utilized in both the theory of quantum computing and in laser cooling techniques.

3.1.3 The Dipole-Approximation

Imagine that our particle is under influence of a source of electromagnetic radiation from a long distance r. The power emitted from the source is then inversely proportional to the distance, and we say that the radiation is of a *dipole type*. If our system is small of extent compared to the distance from the source it is a good approximation to set the electromagnetic fields to be independent of position, i.e.,

$$\boldsymbol{E} = \boldsymbol{E}(t), \quad \boldsymbol{B} = \boldsymbol{B}(t).$$

Clearly, this is a very simple form for the electromagnetic fields as they are independent of position. We stress that these fields are *not* consistent with Maxwell's equations (1.23) as proven on page 54. Our model allows us to neglect the necessary spatial dependence of the fields. Care must be taken when considering potentials $\boldsymbol{A}$ and ϕ, since they must be derived according to the full model and not the simplified fields.

3.2 Physical Problems

We have described the features of various simplifications of the full problem represented by the Hamiltonian (3.1). In this section we will briefly describe some actual and interesting physical systems that is worth investigating numerically. Many applications are possible to find in atomic physics and laser physics as well as solid state physics.

[5] Of course it is not possible to perfectly confine a particle in this way due to Heisenberg's uncertainty principle, but it may be a very good approximation.

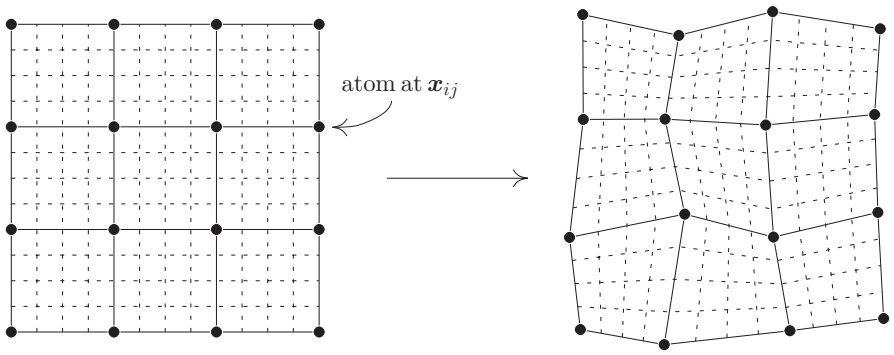

Figure 3.1: Finite element grid corresponding to a two-dimensional model of a solid with node perturbation and refinement

3.2.1 Two-Dimensional Models of Solids

We have not yet described the finite element method, but it is nevertheless interesting to hint at a stationary model that directly may exploit the features of the finite element discretization in a numerical solution.

As a model of a solid (e.g., a metal, crystal or similar) we may study a two-dimensional system in which the atoms of the solid are arranged in a simple grid. We imagine a quadratic slab of length L with N^2 atoms, ions or similar distributed evenly, i.e., at positions

$$\boldsymbol{x}_{ij} = \left(\frac{i-1}{N-1}, \frac{j-1}{N-1} \right), \quad i,j = 1, \ldots, N.$$

An electron inserted into the system experiences a potential $V_{ij}(\boldsymbol{x})$ from each lattice site, e.g., an effective repulsive Coulomb force. The one-particle Hamiltonian then reads

$$H = -\frac{\hbar^2}{2\mu}\nabla^2 + \sum_{i,j=1}^{N} V_{ij}(\boldsymbol{x}).$$

Such models are popular and some of them are in fact possible to solve analytically. One feature of the model is that, roughly speaking, the inserted electrons surprisingly enough acts as if the potentials were absent; they move unhinderedly at constant velocity through the grid, see for example Ref. [22] in which a one dimensional model with infinitely many atoms is considered. The eigenfunctions in an infinite periodic model are plane waves $\exp(ikx)$ modulated with periodic functions. (Not all wave numbers k are allowed, however.) In practice, such behavior is not found in real systems, and this is due to irregularities in the grid-like structure.

To solve the system numerically we may use a rectangular finite element grid, placing the nodes at $\boldsymbol{x}_{ij}$ and then refine the grid to provide sufficient accuracy to the interpolating functions between the nodes. This is illustrated in Fig. 3.1. The finite element method finds a piecewise polynomial approximation over this grid; finer grid means a better piecewise approximation. We will return to the finite element discretization in section 4.4; at this point we will just consider the method as one with very flexible possibilities with respect to the geometry of the problem.

The discrete Hamiltonian implies a discrete time independent Schrödinger equation. Diagonalizing the discrete Hamiltonian arising from this process will then intuitively give an approximation to the exact wave function and energy levels of such a system. This is what we will do in chapter 6 for other systems. If we perturb the positions $\boldsymbol{x}_{ij}$ randomly the discrete Hamiltonian will similarly be perturbed, yielding new eigenvalues

and eigenfunctions. The effect is however to perturb the locations of the atoms or ions, and we have in addition obtained a numerical approximation.

Doing simulations on an ensemble of such perturbed systems may yield statistical information on the distribution of for example energies.

Systems such as those described here are very hot topics in solid state physics. For an introduction, see for example Ref. [28].

3.2.2 Two-Dimensional Hydrogenic Systems

An interesting problem is a two-dimensional hydrogenic system with an applied magnetic field. Such a system has a Hamiltonian given by

$$H = \frac{1}{2\mu}\left(-i\hbar\nabla + \frac{e}{c}\boldsymbol{A}\right)^2 - \frac{ke^2}{r} - \Gamma\boldsymbol{B}\cdot\boldsymbol{S},$$

where $-e$ is the electron charge. The external potential $-ke^2/r$ is an attractive Coulomb force.[6] The full three-dimensional system has proven to be interesting in several major research fields, such as plasma physics, astrophysics and solid state physics. The solid state interest centers on the effect of a magnetic field on shallow impurity levels in a bulk semiconductor, see Ref. [29] and references therein. Ref. [30] is also a comprehensive treatment of the system.

We will concentrate on the two-dimensional model of a hydrogenic system. This system arises as a limit in the case of a bulk semiconductor with impurities, but it also constitutes an interesting system in itself as a fairly complicated system that may display many features of quantum mechanics. Furthermore, such electronic systems in two dimensions constitute a hot topic when viewed as so-called quantum dots (one or more electrons confined in small two-dimensional areas).

In Ref. [29] one concentrates on a vertical time-dependent magnetic field arising from for example the dipole approximation, viz.,

$$\boldsymbol{B} = \gamma(t)\hat{k},$$

and a candidate for the vector potential is then the so-called symmetric gauge, viz.,

$$\boldsymbol{A} = \frac{\gamma(t)}{2}(-y, x, 0).$$

Note that $\nabla\cdot\boldsymbol{A} = 0$, hence we have the Coulomb gauge and the vector potential commutes with the momentum, viz.,

$$[\boldsymbol{A}, \boldsymbol{P}] = \boldsymbol{A}\cdot\boldsymbol{P} - \boldsymbol{P}\cdot\boldsymbol{A} = 0.$$

The spin-dependent term in the Hamiltonian becomes

$$H_{\text{spin}} = -\Gamma\gamma(t)S_3,$$

and the PDEs for different spin components gets decoupled. In fact, H_{spin} commutes with H_{spatial} (i.e., with the rest of the Hamiltonian), and hence we may ignore the spin degrees of freedom altogether. If E_n are the energies of H_{spatial} and ϵ_σ are the energies of H_{spin}, the energies of $H_{\text{spatial}} + H_{\text{spin}}$ are $E'_{n\sigma} = E_n + \epsilon_\sigma$.

In Ref. [29] the eigenvalues of the (spinless) Hamiltonian are discussed as a function of the applied magnetic field strength. In the limits of a vanishing field and of a strong field the eigenvalues are found analytically. Perturbative methods combined with Padé interpolation for analytic continuation of the perturbative results are used to find expressions for the approximate eigenvalues in the intermediate regime.

[6] Note that in Gaussian units $k = 1$. We keep such constants however for completeness.

In Ref. [31] one mentions the possibility of studying the moderate magnetic field region by use of finite element methods to achieve eigenvalues and eigenvectors with a high degree of accuracy. The focus here is to perform a finite element approximation to this system and compare the results with those of the Padé approximants of Ref. [29].

To study the two-dimensional hydrogen atom numerically we need to write the Schrödinger equation on dimensionless form. The numerical values of for example $\hbar$ in cgs or SI units is extremely small, and using such quantities in numerical computations easily lead to round-off errors. Furthermore they are less intuitive to work with. It is easier to study quantities of order unity instead of order 10^{-34}.

The Hamiltonian is given by

$$H = H_{\text{spatial}} + H_{\text{spin}}$$

where $H_{\text{spin}} = -\Gamma \boldsymbol{B} \cdot \boldsymbol{S}$ and

$$H_{\text{spatial}} = \frac{1}{2\mu}\left(-i\hbar\nabla + \frac{e}{c}\boldsymbol{A}\right)^2 - ke^2\frac{1}{r}.$$

Let us first focus on H_{spatial}, dropping the subscript for ease of notation. We introduce a new length scale to our problem, writing $\boldsymbol{x} = \alpha\boldsymbol{x}'$, where α is a numerical constant with the dimension of length, hence $\boldsymbol{x}'$ is a dimensionless vector. Then,

$$\nabla = \frac{1}{\alpha}\nabla',$$

where ∇' denotes differentiation with respect to the components of $\boldsymbol{x}'$. If we insert this into our Hamiltonian, we get

$$H = \frac{\hbar^2}{2\mu\alpha^2}\left(-i\nabla' + \frac{\alpha e}{\hbar c}\boldsymbol{A}\right)^2 - \frac{ke^2}{\alpha}\frac{1}{r'}.$$

Here, $r = \alpha r'$. Note that $\alpha e/\hbar c \cdot \boldsymbol{A}$ must be dimensionless, so we introduce $\boldsymbol{A}' = \alpha e/\hbar c \cdot \boldsymbol{A}$. Next we introduce a scaled dimensionless Hamiltonian H' given by

$$H' = \frac{2\mu\alpha^2}{\hbar^2}H = (-i\nabla' + \boldsymbol{A}')^2 - \frac{\mu\alpha k e^2}{\hbar^2}\frac{2}{r'}.$$

If we require the factor in front of $2/r'$ to be unity, we obtain the length scale

$$\alpha = \frac{\hbar^2}{\mu k e^2} \approx \frac{(197.3 \text{ eV}\cdot\text{nm})^2}{0.5107\cdot 10^6 \text{ eV}\cdot 1.440 \text{ eV}\cdot\text{nm}} = 0.0529 \text{ nm},$$

and accordingly

$$H' = \frac{2\hbar^2}{m(ke^2)^2}H = \beta^{-1}H,$$

where β is the energy scale. The numerical value of β is

$$\beta \approx 0.5107\cdot 10^6 \text{ eV}\frac{(1.440 \text{ eV}\cdot\text{nm})^2}{2(197.3 \text{ eV}\cdot\text{nm})^2} = 13.603 \text{ eV}.$$

When studying the time dependent Schrödinger equation we also need a time scale, i.e., $t = \tau t'$. Upon insertion into the Schrödinger equation we have

$$i\frac{\hbar}{\beta\tau}\frac{\partial\Psi}{\partial t'} = H'\Psi,$$

where

$$\tau = \frac{\hbar}{\beta} = \frac{2\hbar^3}{\mu(ke^2)^2} = \frac{6.582\cdot 10^{-16} \text{ eV}\cdot\text{s}}{13.603 \text{ eV}} = 4.839\cdot 10^{-17} \text{ s}$$

quantity	*definition*	*numerical value*
length	$\alpha = \frac{\hbar^2}{\mu k e^2}$	0.0529 nm
energy	$\beta = \frac{\mu (k e^2)^2}{2\hbar^2}$	13.603 eV
time	$\tau = \frac{2\hbar^3}{\mu (k e^2)^2}$	$4.839 \cdot 10^{-17}$ s
mag. field	$\frac{\beta}{g\mu_B}$	$4.282 \cdot 10^{-8}$ gauss

Table 3.1: Units for the two dimensional hydrogen atom

is the natural time scale.[7] Thus,

$$i\frac{\partial \Psi}{\partial t'} = H'\Psi(\boldsymbol{x}', t'),$$

with

$$H' = (-i\nabla' + \boldsymbol{A}')^2 - \frac{2}{r'}, \tag{3.4}$$

is the dimensionless form of the time dependent Schrödinger equation. Still we need to find a suitable scale for the magnetic field and the spin operators in H_{spin} to complete our discussion.

The natural unit for spin is $\hbar$, as the spin matrices are given in terms of this constant, i.e., $\boldsymbol{S} = \hbar \boldsymbol{S}'$. Hence,

$$\beta^{-1} H_{\text{spin}} = g\beta^{-1}\mu_B \delta \boldsymbol{B}' \cdot \boldsymbol{S}',$$

where we have introduced δ as the natural scale of the magnetic field. The constant μ_B is called the Bohr magneton and has the value

$$\frac{ge\hbar}{2\mu c} = 5.789 \cdot 10^{-9} \text{ eV/gauss}.$$

Requiring $g\mu_B\delta/\beta = 1$ yields

$$\delta = \frac{\beta}{g\mu_B} = 4.282 \cdot 10^{-8} \text{ gauss}$$

where we have used $g = 2.00232$ for the electron, quoted from Ref. [12].

From now on, we will drop the primes as the conversion in units is unambiguously given by the tabulated quantities in Table 3.1. We use the symmetric gauge $\boldsymbol{A} = \gamma(t)/2 \cdot (-y, x, 0)$ for the vector potential. Note that by the chain rule,

$$\frac{\partial}{\partial \phi} = \frac{\partial x}{\partial \phi}\frac{\partial}{\partial x} + \frac{\partial y}{\partial \phi}\frac{\partial}{\partial y} = -y\frac{\partial}{\partial x} + x\frac{\partial}{\partial y},$$

and that $\boldsymbol{A}^2 = \gamma(t)^2(x^2 + y^2)/4 = \gamma(t)^2 r^2/4$. We obtain for the kinetic energy term in Eqn. (3.4)

$$(-i\nabla + \boldsymbol{A})^2 = -\nabla^2 - 2i\boldsymbol{A}\cdot\nabla + \boldsymbol{A}^2 = -\nabla^2 - i\gamma(t)\frac{\partial}{\partial\phi} + \frac{\gamma(t)^2}{4}r^2.$$

The spinless Hamiltonian now reads

$$H = -\nabla^2 - i\gamma(t)\frac{\partial}{\partial\phi} + \frac{\gamma(t)^2}{4}r^2 - \frac{2}{r}. \tag{3.5}$$

In section 6.1.3 we deal with the time independent version of this problem numerically.

[7] Compare the time scale with the life time of the classical hydrogen atom losing its energy due to radiation, $\tau \approx 1.5 \cdot 10^{-11}$ s.

Chapter 4

Numerical Methods for Partial Differential Equations

This chapter is devoted to numerical solutions of partial differential equations (PDEs). Most such are very hard to solve analytically, implying the need for approximate solution methods. The first section describes the main features of partial differential equations in general and ordinary differential equations (ODEs) in particular. Then we turn to different numerical methods for approximating these.

The main difference between PDEs and ODEs is that in the ODE the unknown function we search for depends only on one parameter, i.e., the equation contains only derivatives with respect to one variable.

Let us outline the general strategy used when attacking a PDE numerically. Consider as an example a simplified version of the Schrödinger equation, viz.,[1]

$$iu_t = -u_{xx} + V(x)u.$$

This differential equation contains partial derivatives of first order with respect to time and of second order with respect to spatial coordinates. We use $u(t = 0) = f$ as initial condition and there are boundary conditions that must be fulfilled in the spatial directions.

How do we solve such a problem on a computer? Clearly, the problem must be finite in extent in order to be calculable. We must in some way transform differential operators and functions into *discrete and finite* representations.

The unknown function $u(x)$ at time t has infinitely many degrees of freedom; one for each x. Imagine that we instead assume that x only can take a finite number of values x_j, $j = 1, 2, \ldots, N$, which is the main idea of finite difference approximations among others. Clearly, $u(t)$ becomes an element of $\mathbb{C}^N$. As a consequence the differential operator $\partial^2/\partial x^2$ must be replaced by an *algebraic* relation between the different function values $u(x_j, t)$. The multiplicative operator $V(x)$ must be represented by some appropriate variant; clearly also an algebraic relation.

The PDE is by these means converted into an ordinary differential equation of dimension N, viz.,

$$iu_t = D(u) + V(u), \quad u(t) \in \mathbb{C}^N,$$

where D and V are mappings from $\mathbb{C}^N$ into $\mathbb{C}^N$. Since the Hamiltonian $-\partial^2/\partial x^2 + V(x)$ is an Hermitian operator, we also wish that our numerical methods yield Hermitian discrete versions, i.e., that D and V can be represented by linear Hermitian matrices. This ensures that unitarity of the Schrödinger equation is transferred to the ODE as well.

[1] Note the subscript notation for partial derivatives.

Our attention may now be focused on solving the ODE. The following sections outline some widely used methods for performing the conversion of a PDE to an ODE. After this we outline methods for solving ODEs, focusing on methods applicable to the Schrödinger equation. This is typically done with a finite difference method of some kind of consideration of the exact flow of the ODE such as operator splitting methods. In other words, the temporal degree of freedom is also converted into a discrete approximation. Ultimately, we have a complete discrete formulation of our differential equation.

There is nothing wrong by going the other way round, i.e., first discretize the time dependence and then apply the spatial approximation. Indeed, in some cases it is clearer to take this alternate approach.[2] Ultimately, the PDE is reduced to an algebraic problem, or if we wish to view it as such, a sequence of such; one problem at each time level.

4.1 Differential Equations

In this section we review some basic concepts regarding differential equations; both ordinary and partial differential equations. We will not employ mathematical rigor, because it is not necessary for our applications. There are numerous standard texts to consult, see for example Refs. [17, 32, 33].

A differential equation is an equation in which the unknown is a function. The equation relates the unknown's partial derivatives. We distinguish between *ordinary differential equations* (ODEs) and *partial differential equations* (PDEs). Partial differential equations are an extremely rich class of problems. They are divided into different types, each displaying different behavior. The different kinds of equations may behave differently with different numerical methods. Examples of PDE types are parabolic equations, elliptic equations (such as the Schrödinger equation) and hyperbolic equations.

4.1.1 Ordinary Differential Equations

The unknown in an ODE is a function of only one variable t. In general we search for a function

$$y : I \subset \mathbb{R} \longrightarrow V,$$

where V is some vector space called *the phase space* and $I = [t_0, t_1]$ is some interval, and the ODE reads

$$\dot{y} = f(y, t).$$

This is a *first order* equation because only derivatives of first order enter the equation. This is not a limitation, since all higher-order ODEs may be rewritten as first-order ODEs by extending V. The function $y(\cdot, t)$ is a function from V into V, i.e., a vector field. We say that an ODE is *autonomous* if f has no explicit dependence on time. Any non-autonomous ODE may be turned into an autonomous ODE by extending the phase space with one dimension, viz.,

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} y \\ \tau \end{pmatrix} = \begin{pmatrix} f(y, \tau) \\ 1 \end{pmatrix}.$$

If f is sufficiently nice it is easy to see that the solution (if it exists) is unique, once given the *initial condition* $y(t_0)$. We will not prove existence or uniqueness here, see Ref. [15].

[2] If discretizing the time domain first, we must be aware of some existence problems in the formulation. If we ignore this, our derivations become purely formal, and the formality of the calculations vanish when we introduce a spatial discretization. We will see this in section 4.5.

In the case of a unique solution we have a mapping $U(t,s)$ that takes an initial condition $y(s)$ into the solution of the differential equation at time t, i.e.,

$$U(t,s)y(s) = y(t).$$

By first propagating to t and then to t' we get the composition property

$$U(t',s) = U(t',t)U(t,s).$$

For t sufficiently close to s we may invert the time propagation, yielding

$$U(t,s) = U(s,t)^{-1}.$$

Of course,

$$U(t,t) = 1.$$

This mapping is called *the flow*, *the propagator* or *the evolution operator*. It corresponds to the propagator of the Schrödinger equation.

This means that ordinary differential equations describe the motion of a point in phase space. The image $y(I)$ is called *the trajectory*. One might consider more general ODEs in which y is a point on a differentiable manifold instead of a vector space, and in which f is a tangent vector field. The differential equation for the propagator is such a generalized ODE. By the following identities,

$$\dot{y}(t) = \frac{\mathrm{d}}{\mathrm{d}t}U(t,s)y(s) = \dot{U}(t,s)y(s) = f(U(t,s)y(s)) = f \circ U(t,s)y(s),$$

which must hold for all $y(s)$, we get

$$\dot{U}(t,s) = f \circ U(t,s), \quad U(s,s) = 1.$$

Hamilton's equations of motion (1.2) is a very important example of an ordinary differential equation.

4.1.2 Partial Differential Equations

Where ODEs searched for a (multidimensional) function of a single variable, partial differential equations (PDEs) search for a (possibly multidimensional) function of several variables. Hence, the partial derivatives with respect to all the arguments of the unknown appear in the equation.

Assume that the function u we are seeking is defined on some subset of $\mathbb{R}^n$, viz.,

$$u : \Omega \subset \mathbb{R}^n \longrightarrow V,$$

where V again is some vector space. It is useful to consider u as an element in some linear space, such as a Banach space or a Hilbert space.

In general we write our PDE as

$$\mathcal{L}(u) = f,$$

where f is a function in the mentioned space, on equal footing with u. The operator $\mathcal{L}$ may be complicated, but in case of a linear operator we say that the PDE is linear. (Note that linear equations may be harder to solve than non-linear ones! Linear PDEs form a huge class of problems.) Furthermore, the equation is said to be *homogenous* if $f = 0$.

Here are some simple yet important examples of PDEs.

– $u_t + u_x = 0$: A transport equation.

- $u_t - u_{xx} = 0$: The one-dimensional diffusion equation.
- $iu_t - u_{xx} = 0$: The one-dimensional time dependent Schrödinger equation for a free particle.
- $u_{tt} - u_{xx} = 0$: The one-dimensional wave equation.
- $u_{xx} + u_{yy} = 0$: Laplace' equation.

Common to all these examples is that they are homogenous. The time dependent Schrödinger equation is in general also homogenous. There is some missing information in the examples for them to represent well-defined physical problems with a unique solution. What is the initial distribution of temperature u in the heat equation? Assuming that we model a vibrating string with the wave equation, what are the reflection coefficients at the ends of the string?

Typically we separate between spatial dependence and time dependence. This is connected to our physical way of thinking. Thus, often one silently assumes that an initial condition problem is at hand when the PDE contains t as a parameter, and some sort of boundary condition problem whenever $\boldsymbol{x}$ occurs. This is particularly clear in the last two examples above. The wave equation and Laplace' equation are almost identical, but the presence of y instead of t in the last implies a problem of completely different character. Laplace' equation is a stationary problem while the wave equation is a time-dependent problem.

For equations of order n in time (i.e., when n is the highest order of the time derivative that occurs) we need n initial conditions to "set off" the solutions, i.e., we must supply $\partial^k u/\partial t^k$ for $k = 1, 2, \ldots, n$ at $t = 0$.

Boundary conditions specify the asymptotic behavior of u. When solving PDEs numerically we consider only compact domains (i.e., closed and bounded domains), and in that case the boundary conditions specify the behavior of the solution at the boundary $\partial\Omega$. Most important for our purposes are *Dirichlet boundary conditions* in which we specify the value of u at the boundary, i.e. we specify $u(\partial\Omega)$. Other kinds of conditions include *Neumann conditions*, in which the normal component of the gradient of u is specified, i.e., $\boldsymbol{n} \cdot \nabla u$ is supplied, where $\boldsymbol{n}$ is a unit normal field of $\partial\Omega$.

Usually we formulate the Schrödinger equation with $\Omega = \mathbb{R}^n$, but for computational purposes this is not suitable. Physically, limiting Ω to some compact region in effect sets the potential to infinity there since the particle is not allowed to penetrate into the region outside Ω. Then the wave function vanishes at the boundary for the equation to be fulfilled. In other words, when solving the time dependent Schrödinger equation we use homogenous Dirichlet boundary conditions, viz.,

$$u(\partial\Omega) \equiv 0.$$

PDEs and ODEs are closely related, especially for computational purposes. On one hand, an ODE is a PDE in which the unknown depends only on t. On the other hand, if we can specify u as being an element in some (infinite-dimensional) Hilbert space and if we can find a countable basis, then a time-dependent PDE is equivalent to an infinite-dimensional ODE. An example of this is seen in the discussion on the time dependent Schrödinger equation in section 1.4. The typical computational approach is then to in some way limit the dimensionality of the Hilbert space by ignoring all but a finite subspace, as indicated in the introduction.

4.2 Finite Difference Methods

Perhaps the most obvious and intuitive way of discretely approximating differential operators is by means of *finite differences*. This method replaces the continuous function

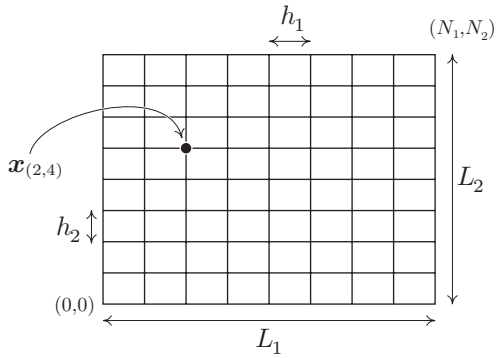

Figure 4.1: A uniform grid

by a discrete function defined at evenly spaced grid points in space. Partial derivatives are approximated with difference expressions derived from considerations of Taylor expansions of the original function.

4.2.1 The Grid and the Discrete Functions

Assume that we are given a PDE with unknown u, viz.,

$$u : \Omega \subset \mathbb{R}^n \longrightarrow V.$$

Let us introduce some notation. A *multi-index* α is an ordered sequence of n integers, viz.,

$$\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_n),$$

where n is the dimension of Ω in our applications. We may add and subtract multi-indices in a component-wise fashion, i.e.,

$$(\alpha + \beta)_i = \alpha_i + \beta_i,$$

and without danger of confusion, if we specify a multi-index $\alpha + \beta_i$, then β_j is supposed to have zero components β_i for $i \neq j$.

Suppose that Ω is a rectangular domain, i.e., it is the cartesian product of intervals, viz.,

$$\Omega = I_1 \times I_2 \times \cdots \times I_n.$$

If we subdivide each interval I_i into N_i subintervals of uniform length h_i, we get $N_i + 1$ well-defined "joints" x_j^i, $j = 0, \cdots, N_i$. See Fig. 4.1 for an illustration. The grid G now consists of the points $\boldsymbol{x}_{i_1 i_2 \ldots i_n}$ given by

$$\boldsymbol{x}_{i_1 i_2 \ldots i_n} = (x_{i_1}^1, x_{i_2}^2, \ldots, x_{i_n}^n), \quad i_j = 0, \ldots, N_j, j = 1, \ldots, n.$$

Thus, the grid G is given as

$$G = \{\boldsymbol{x}_\alpha \ : \ \alpha_j = 0, 1, \ldots, N_j\}.$$

Such a G is referred to as a *uniform grid*. The *mesh width* is the largest of the spacings h_i and intuitively measures the "fineness" of the approximation.

Assuming that Ω is rectangular is not really hampering, because one may specify boundary conditions in the discrete formulation mimicking the real shape of Ω. This is illustrated in Fig. 4.2. Our computational algorithm gets much more complicated when employing such complicated geometries, one of the major drawbacks with finite difference methods. Furthermore, if the discretized boundary has for example "staircase shape" artificial physical effects may be introduced to the numerical solution.

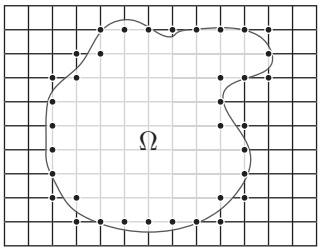

Figure 4.2: Discretizing a non-rectangular domain

As for the function u and other functions in the problem definition such as initial conditions, we define a discrete approximation u_h defined in the grid points of G, viz.,

$$u(\boldsymbol{x}) \longrightarrow u_h(\boldsymbol{x}_\alpha).$$

Thus, u_h may be viewed as an $N_1 N_2 \ldots N_n$-dimensional vector whose components are elements in V, or as a tensor of rank n with N_i components of type V in each direction. Indeed, we write

$$u_\alpha := u_h(\boldsymbol{x}_\alpha)$$

for the components of this tensor or discrete function. When doing practical computations in for example two dimensions, we simplify the notation and write x_{ij} for the grid points and u_{ij} for the components of u_h, where i and j are indices in the x and y directions, respectively.

4.2.2 Finite Differences

The differential operators occurring in the PDE problem are approximated by difference expressions called *finite differences*. At a given grid point $\boldsymbol{x}_\alpha$ the partial derivatives are approximated by a (linear) function of u_α and the neighboring grid points. There is no unique way to define finite difference expressions for the different partial derivatives. The choice of the approximation is highly problem dependent. In addition, more complicated compositions of operators such as $(f(x)u_x)_x$ may be approximated by choosing a perhaps non-obvious but "smart" finite difference.

There are however some finite differences that are more widely used than others, and we shall introduce a special notation for these. This notation makes it easy to develop difference schemes for various PDEs and the notation follows closely that of the actual partial derivatives.

We introduce the notation in the setting of a one-dimensional problem. The generalization to several dimensions is obvious. First we define a *centered difference* for the partial derivative $\partial/\partial x$:

$$[\delta_{nx} u]_j := \frac{1}{nh}(u_{j+n/2} - u_{j-n/2}). \tag{4.1}$$

Here, h is the grid spacing and $n > 0$ scales the step size in the finite difference. Note that this is clearly inspired by the definition of the partial derivative. The same goes for the *one-sided forward* and *backward* differences, viz.,

$$[\delta^+_{nx} u] := \frac{1}{nh}(u_{j+n} - u_j), \tag{4.2}$$

and

$$[\delta^-_{nx} u] := \frac{1}{nh}(u_j - u_{j-n}). \tag{4.3}$$

In all three cases, the distance along the axis of the sampling points are nh.

These three finite difference expressions all approximate the first derivative. A widely used expression for the second derivative is given by combining δ_x twice, viz.,

$$[\delta_x \delta_x u]_j = \frac{1}{h^2}(u_{j+1} - 2u_j + u_{j-1}). \tag{4.4}$$

It is fundamental to be able to analyze the error in the numerical discretizations. One of the concepts to consider is *the truncation error*, defined as the error in the numerical discretization when we insert an exact solution. Thus, if $\mathcal{L}$ is a differential operator and if $\mathcal{L}_h$ is a numerical approximation, then the truncation error is defined as

$$\tau := \mathcal{L}_h(u) - \mathcal{L}(u).$$

For example, the approximation (4.4) of the second derivative has truncation error

$$\tau = \frac{1}{h^2}(u(x+h) - 2u(x) + u(x-h)) - u''(x).$$

When we expand $u(x \pm h)$ in Taylor series we easily obtain

$$\tau = \frac{1}{h^2}(2\frac{h^2}{2}u''(x) + \mathcal{O}(h^4) - u''(x) = \mathcal{O}(h^2),$$

and so the truncation error is of second order in the step size. When τ vanishes with vanishing mesh width we say that we have a *consistent approximation*. The name of τ is related to the fact that it is the error arising from *truncating the Taylor series* after a few terms. The truncation errors for the other finite differences presented above is easily derived, viz.,

$$[\delta_{nh} u]_j = u'(x_j) + \frac{1}{24}(nh)^2 u'''(x) + \mathcal{O}(h^4), \tag{4.5}$$

$$[\delta^+_{nh} u]_j = u'(x_j) + \frac{1}{2}nhu''(x) + \mathcal{O}(h^2), \tag{4.6}$$

$$\text{and} \quad [\delta^-_{nh} u]_j = u'(x_j) - \frac{1}{2}nhu''(x) + \mathcal{O}(h^2). \tag{4.7}$$

The centered difference provides a better approximation to $u'(x_j)$ than the one-sided differences. Intuitively this is so because it utilizes information from both sides of x_j.

4.2.3 Simple Examples

Let us apply the finite differences to some simple model problems. We will see that the notation introduced makes the similarity between the PDE formulation and finite difference formulation obvious. The examples will also illustrate some important concepts, such as incorporating boundary conditions and initial conditions.

The Heat Equation. Consider the two-dimensional heat equation, viz.,

$$u_t = u_{xx} + u_{yy}.$$

We assume that the spatial grid is given and has points which we denote as $\boldsymbol{x}_{i,j} = (x_i, y_j)$. Time is discretized as $t_\ell = \ell \Delta t$ and although we according to the above description of the grid should include it in $\boldsymbol{x}$ it is somewhat clearer to separate the time-dependent part of the grid. We also assume that $h_x = h_y = h$. First, let us consider a finite difference scheme applied to only the spatial degrees of freedom, viz.,

$$[\dot{u}(t) = \delta_x \delta_x u + \delta_y \delta_y u]_j.$$

This clearly turns the spatial derivatives into algebraic operators on the discrete function. Since we still have a time derivative in the equation we now have an ordinary differential equation. We will employ two different difference schemes for resolving the derivative; namely a forward difference and an backward difference approximation. A forward difference yields

$$[\delta_t^+ u]_{i,j}^\ell = [(\delta_x \delta_x + \delta_y \delta_y)u]_{i,j}^\ell.$$

Note that we have placed the time level index ℓ as a superscript. When performing an implementation we seldom keep the solution at all time levels in memory.

Writing out the difference scheme and reorganizing yields

$$u_{i,j}^{\ell+1} = u_{i,j}^\ell + \frac{\Delta t}{h^2}\left(u_{i+1,j}^\ell + u_{i-1,j}^\ell + u_{i,j+1}^\ell + u_{i,j-1}^\ell - 4u_{i,j}^\ell\right).$$

This scheme is called *explicit*, because updating the solution at the next time level is explicitly given as a function of the solution of the current time level. Time stepping with δ_t^+ is called *forward Euler* or *explicit Euler*.

Replacing δ_t^+ with δ_t^- yields a different scheme. This way of time stepping is called *backward Euler* or *implicit Euler*. The scheme reads

$$u_{i,j}^\ell - \frac{\Delta t}{h^2}\left(u_{i+1,j}^\ell + u_{i-1,j}^\ell + u_{i,j+1}^\ell + u_{i,j-1}^\ell - 4u_{i,j}^\ell\right) = u_{i,j}^{\ell-1}.$$

This scheme is called *implicit* because the solution at time level ℓ is given implicitly; we have to solve a non-trivial set of algebraic equations to find u^ℓ. It is easy to see that the system of equations is a *linear* algebraic system, i.e., we may rewrite it as a matrix equation.

Boundary conditions were neglected in the above discussion, but we include them in the next example.

The Wave Equation. Let the domain be given as $\Omega = [0, 1]$. Let us describe a finite difference method for the one-dimensional wave equation, i.e.,

$$u_{tt} = u_{xx},$$

with boundary conditions

$$u(0, t) = u(1, t) = 0.$$

The initial condition is

$$u(x, 0) = f(x), \quad u_t(x, 0) = g(x).$$

We use $N + 1$ grid points, i.e.,

$$G = \{x_j = jh \,:\, j = 0, \ldots N\}.$$

The time levels are given as $t_\ell = \ell\Delta t$ as in the previous example. We devise the scheme

$$[\delta_t \delta_t u]_j^\ell = [\delta_x \delta_x u]_j^\ell, \quad j = 1, \ldots, N-1.$$

For $j = 0$ and $j = N$ we impose the boundary conditions, i.e., that $u_0^\ell = u_N^\ell = 0$. For $\ell = 0$ we use the initial condition, viz.,

$$u_j^0 = f(x_j).$$

For $\ell = 1$ we use the second part of the initial condition and a forward difference approximation, viz.,

$$u_j^1 = u_j^0 + \Delta t g(x_j).$$

For the subsequent time levels we use the devised scheme, which when written out reads

$$u_j^{\ell+1} = 2u_j^\ell - u_j^{\ell-1} + \frac{\Delta t^2}{h^2}\left(u_{j+1}^\ell - 2u_j^\ell + u_{j-1}^\ell\right).$$

This is an explicit scheme.

Note the way in which the initial conditions were incorporated by a special rule for the two first time levels. Similarly, the boundary conditions were incorporated by an (obvious) special rule.

Let us mention some stability properties of this scheme. It turns out that the scheme is stable if and only if *the Courant number* $C := \Delta t^2/h^2 \leq 1$. In fact we obtain the exact solution at the grid points if $C = 1$. (This is not true for the two-dimensional generalization of the scheme.) Thus, taking too large time steps makes the process unstable, a feature that we also will see when studying the Schrödinger equation. See Ref. [34] for a thorough discussion.

A Non-Linear PDE. Now for a more delicate treat. Consider the PDE

$$u_t = (f(u)u_x)_x,$$

where $f(u)$ may be any positive and smooth function. This equation is a non-linear Heat equation. We will ignore boundary conditions for now to focus on the non-linearity, and we use the same grid in space and time as for the wave equation.

We devise the following scheme:

$$[\delta_t^- u]_j^\ell = [\delta_x(f(u)\delta_x u]_j^\ell.$$

On the right hand side we have a small problem. When written out it reads

$$\frac{1}{h}\left(f(u_{j+1/2})[\delta_x u]_{j+1/2} - f(u_{j-1/2})[\delta_x u]_{j-1/2}\right)^\ell,$$

and the trouble is we do not know $u_{j\pm 1/2}$. Therefore, we choose an approximation given by

$$\bar{f}_j := f(\frac{u_{j+1} + u_j}{2}).$$

It is easy to see that

$$\bar{f}_j = f(u_{j+1/2}) + \mathcal{O}(h^2),$$

so this approximation is of second order.

When we write out the equation and gather the unknowns u_j^ℓ on one side we get

$$u^\ell - \frac{\Delta t}{h^2}\left(\bar{f}_j^\ell(u_{j+1}^\ell - u_j^\ell) - \bar{f}_{j-1}^\ell(u_j - u_{j-1})\right) = u_j^{\ell-1},$$

which is non-linear in u_j^ℓ. Therefore, to generate the solution on the next time step we must be able to solve a non-linear set of algebraic equations. There are several ways to do this, but perhaps the most popular one is Newton-Rhapson iteration. See Ref. [34] for a discussion.

By choosing for example forward Euler instead of backward Euler one may get an explicit scheme instead, with no non-linear equations to solve. However, forward Euler tends to be unstable.

In addition to being more complicated to solve, the non-linear schemes are more difficult to analyze with respect to stability and convergence. Ref. [34] contains some material on the subject.

The One-Dimensional Schrödinger Equation. As an appetizer to solving the time dependent Schrödinger equation in chapter 7 we will solve the simplified one-dimensional version for a spinless particle, i.e.,

$$iu_t = -u_{xx} + V(x)u(x),$$

where the potential $V(x)$ is some scalar function that is independent of time. The Hamiltonian is then $H = \partial^2/\partial x^2 + V$. We choose for simplicity $\Omega = [0, 1]$ as for the wave equation and the corresponding grid. The standard finite difference approximation reads

$$i\dot{u}_j = [-\delta_x\delta_x u]_j + V_j u_j,$$

where $V_j = V(x_j)$. Notice that $\delta_x\delta_x$ can be written as a tridiagonal matrix when acting on u_h.

The big question is what discretization to use for time integration. In section 4.5 we will study the *Crank-Nicholson* scheme, a scheme first introduced by Goldberg *et al.* in Ref. [35]. Let us write out this scheme for our example. It is defined by

$$i[\delta_t^+ u]^\ell = \frac{1}{2}Hu^{\ell+1} + \frac{1}{2}Hu^\ell.$$

This yields

$$\left(1 + \frac{1}{2}i\Delta t H\right) u^{\ell+1} = \left(1 - \frac{1}{2}i\Delta t H\right) u^\ell,$$

and we see that we have an implicit scheme. This scheme preserves the norm of $u_h \in \mathbb{C}^N$ exactly. In other words, the scheme is *stable*.

The implicit nature of the scheme means that we have to solve a set of linear equations of dimension N at each time step. This system is easily seen to be tridiagonal, and hence it can be solved in $\mathcal{O}(N)$ operations, see chapter 5. When going to higher dimensions the system no longer becomes tridiagonal ant it takes much longer to solve.

4.2.4 Incorporating Boundary and Initial Conditions

As seen in the last example, incorporating boundary conditions was no big trouble. Let us however review the process in a more general setting.

When devising a grid G corresponding to the domain Ω, we should choose a subset $\Gamma \subset G$ corresponding to the boundary $\partial\Omega \subset \Omega$. On this set we impose our discrete boundary conditions. Choosing Γ is not always a simple task, but on a rectangular Ω it is of course in a natural way given by

$$\Gamma = \{\boldsymbol{x}_\alpha \in G \,:\, \alpha_j = 0 \text{ or } \alpha_j = N_j \text{ for at least one } j\}.$$

Imposing Dirichlet boundary condition is the simplest task, because one can simply set

$$u_\alpha^\ell = \psi(\boldsymbol{x}_\alpha, t_\ell) = \psi_\alpha^\ell, \quad \boldsymbol{x}_\alpha \in \Gamma, \ell = 0, 1, \ldots,$$

where $\psi : \partial\Omega \to V$ is the boundary condition.

Neumann conditions may impose difficulties. How do we interpret $\boldsymbol{n}$, i.e., the unit normal, when considering the discrete boundary Γ? The answer is problem dependent.

For now, let us only consider rectangular domains. In this case, the boundary of Ω is clearly an $n-1$ dimensional box in $\mathbb{R}^n$ consisting of $2n$ sides (that is hyperplanes). If such a side is described by the condition $x^k = c$ with c a constant, then the unit normal is $\pm e^k$; the kth standard basis vector in $\mathbb{R}^n$, the sign depending on what side we are considering. Thus,

$$\boldsymbol{n} \cdot \nabla u = \pm \frac{\partial u}{\partial x^k}$$

along this side.

Let us consider a concrete example for illustration. Given a two dimensional PDE (with perhaps an additional time dependence) and assume that we want to apply the Neumann condition

$$\boldsymbol{n} \cdot \nabla u = a$$

along the boundary of our rectangular domain Ω given by

$$\Omega = [x_0, x_1] \times [y_0, y_1].$$

Along the boundary $x = x_0$ the unit normal is $-\boldsymbol{e}_x$ and thus

$$\boldsymbol{n} \cdot \nabla u = -\frac{\partial u}{\partial x} = a.$$

Similarly,

$$\boldsymbol{n} \cdot \nabla u = \frac{\partial u}{\partial x} = a$$

along $x = x_1$. For the two remaining parts $y = y_0$ and $y = y_1$ we obtain

$$-\frac{\partial u}{\partial y} = a \quad \text{and} \quad \frac{\partial u}{\partial y} = a,$$

respectively.

Solving a transient problem of course means solving the PDE for $t \in [t_0, t_1]$, and so the time grid is always rectangular. The initial conditions are imposed by devising special rules for $\ell = 0, \ldots, p-1$, where p is the order of the highest time derivative occurring in the PDE. In each case we must consider exactly how, but u^0 is of course the initial function and u^1 is usually calculated with either a forward Euler step and similarly for u^1 up to u^{p-1}.

4.3 The Spectral Method

The finite difference approximations for spatial derivatives that are employed are typically of rather low order in the step size h, i.e., typically of order two. Based on the *discrete Fourier transform* (DFT) we may create a method that is of order N, where N is the number of grid points in one spatial direction. While this method is applicable up to three dimensions, and while it provides great accuracy, there are some catches.

First of all the computational effort is much higher. A finite difference approximation requires typically not much more than $\mathcal{O}(n)$ operations (where n is the total number of grid points), while the spectral method requires $\mathcal{O}(n \log n)$ operations when we utilize the so-called fast Fourier transform algorithm (FFT) for performing the DFT.[3] In addition, the FFT algorithm requires 2^k grid points in each spatial direction with an equal spacing each, and the method requires periodic boundary conditions in a natural way. Thus the freedom of the geometry in question gets somewhat restricted.

The idea behind the spectral method is that differentiation is a diagonal operator in the frequency domain. Recall that the Fourier transform of a function $f(x)$ is given by

$$F[f](k) := g(k) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(x) e^{-ikx} \, \mathrm{d}x,$$

and the inverse Fourier transform is given by

$$F^{-1}[g](x) := f(x) = \int_{-\infty}^{\infty} g(k) e^{ikx} \, \mathrm{d}x.$$

[3] Se Ref. [...] for a discussion of the fast Fourier transform.

Differentiating we get

$$f'(x) = \int_{-\infty}^{\infty} ikg(k)e^{ikx}\,\mathrm{d}x.$$

Thus,

$$F[f'(x)](k) = ikF[f](k) = ikg(k),$$

and differentiating becomes multiplication with ik in the frequency domain, i.e., it is a diagonal operation. It is easy to see that if A is an operator combining different differentiations, i.e. $A = A(\frac{\partial}{\partial x})$,[4] then

$$F[A(\frac{\partial}{\partial x})f(x)](k) = A(ik)F[f](k).$$

For example, if $A = \frac{\partial^2}{\partial x^2} + \beta\frac{\partial}{\partial x}$, then

$$F[Af(x)](k) = (-k^2 + \beta ik)F[f](k).$$

It is not difficult to imagine that if we can make a good approximation to the Fourier transform of our discrete function, then we get a good approximation to any differentiation process with a simple diagonal (i.e., multiplicative) operator on the Fourier transformed function. Operating with A on $f(x)$ is then equivalent to calculating

$$A(\frac{\partial}{\partial x})f = F^{-1}\left[A(ik)F[f]\right].$$

4.3.1 The Discrete Fourier Transform

Assume that we have a discretely sampled function $f(x)$ on $[0, a]$, i.e., that we have a one-dimensional grid G of $N + 1$ points (with N even), viz.,

$$G = \{x_n = hn \,:\, n = 0, 1, \ldots, N\}$$

with $f_n = f(x_n)$ and $h = a/N$. We shall assume $f_0 = f_N$, viz., that f is periodic. If we imagine the discrete f as being a superposition of plane waves, i.e. of different e^{ikx}, then clearly we cannot have wave numbers greater than π/h, that is, waves with wavelength smaller than $2h$. If the wavelength were smaller, then the discrete e^{ikx} would be equivalent to a wave of bigger wavelength. This phenomenon is called *aliasing* or *folding*. The critical wave number $k_c = \pi/h = N\pi/a$ is called the *Nyquist wave number*.[5]

As motivation for the discrete Fourier transform we need the so-called *Nyquist theorem* which we state without proof.

Theorem 16
Assume that $g(k) = F[f](k)$ is identically zero for $|k| \geq k_c = \pi/h$, i.e., that $f(x)$ is band-limited. *Then $f(x)$ is uniquely determined by the discrete version f_n, $n = 0, 1, \ldots, N-1$.*

This remarkable theorem states that the information content in $f(x)$ is very much less than the content in a function that is not band-limited. In fact, the complete information is given by a finite number of values, namely f_n. On the other hand, this means that there is much redundant information in the Fourier transformed function $g(k)$; it would be sufficient to devise N numbers $g_m = g(k_m)$, $m = 0, 1, \ldots N-1$ that can be put in a one-to-one correspondence with f_n. In other words, we may

[4] Of course, some care should be taken when claiming this, but at least for simple polynomial expressions it holds trivially.

[5] Some texts work with time t and angular frequency ω as conjugate variables. In that case one speaks of the Nyquist frequency.

devise a discrete Fourier transform (that of course in some sense should be a good approximation to the continuous transform) containing all the information in f_n.

The DFT is defined by defining a second grid G' with $N+1$ points, viz.,

$$G' = \{k_m = \frac{2\pi m}{Nh} = 2k_c\frac{m}{N} \ : \ m = -\frac{N}{2}, \ldots, \frac{N}{2}\},$$

and by using the trapezoidal rule for approximating the Fourier transform at the grid points k_m, viz.,

$$\begin{aligned} g_m = \frac{1}{2\pi}\int_0^a e^{-ik_m x} f(x)\,\mathrm{d}x &\approx \frac{h}{2\pi}\sum_{n=0}^{N-1} e^{-ik_m x_n} f_n \\ &= \frac{h}{2\pi}\sum_{n=0}^{N-1} e^{-2\pi i n m/N} f_n, \end{aligned} \tag{4.8}$$

where $a = Nh$ is the length of the interval, and where we have used the periodicity of f. If we define the matrix Z by

$$Z_{mn} = e^{-2\pi i n m/N}, \quad n, m = 0, 1, \ldots, N-1,$$

then

$$g_m = \sum_{n=0}^{N-1} Z_{mn} f_n.$$

Clearly, f has only N independent components due to $f_0 = f_N$ and we also have $g_{-N/2} = g^*_{N/2}$. If we can prove that Z is invertible, then we have a well-defined one-to-one mapping between f_n and g_m. It turns out that Z is in fact unitary (up to a multiplicative constant), so in the same way as in the original Fourier transform we may interpret the transformation as an orthogonal change of basis.[6] To be more specific,

$$ZZ^\dagger = NI \Rightarrow Z^{-1} = \frac{1}{N}Z^\dagger,$$

where I is the identity matrix. Let us prove this.

Theorem 17
The $N \times N$ matrix Z given by

$$Z_{mn} = e^{-2\pi i n m/N}, \quad n, m = 0, 1, \ldots, N-1,$$

obeys

$$ZZ^\dagger = NI, \quad Z^{-1} = \frac{1}{N}Z^\dagger.$$

In other words, the column vectors form an orthogonal basis for $\mathbb{C}^N$, their length being $\sqrt{N}$.

Proof: To prove this we must compute

$$\begin{aligned} [ZZ^\dagger]_{mn} = \sum_{j=0}^{N-1} Z_{mj} Z^\dagger_{jn} &= \sum_{j=0}^{N-1} e^{-2\pi i m j/N} e^{2\pi i n j/N} \\ &= \sum_{j=0}^{N-1} e^{2\pi i j(n-m)/N} \equiv \sum_{j=0}^{N-1} \phi_j. \end{aligned}$$

[6] See Appendix A.

Note that for $n = m$ we have $\phi_j = 1$, so that the diagonal elements are equal to N. For the off-diagonal elements, note that $|n - m| < N$ and that if we define $k = N - 1 - j$, then we may write

$$\begin{aligned} S &= \sum_{k=N-1}^{0} e^{2\pi i(N-1-k)(n-m)/N} \\ &= e^{2\pi i(n-m)} e^{-2\pi i(n-m)/N} \sum_{k=0}^{N-1} e^{-2\pi ik(n-m)/N} = e^{-2\pi i(n-m)/N} S. \end{aligned}$$

Since $|n - m| < N$ then $(n - m)/N$ is not an integer and so $e^{2\pi i(n-m)/N} \neq 1$, and consequently we must have $S = 0$.

Hence, $ZZ^\dagger = NI$, and if ψ_n is the n'th column vector, then (ψ_m, ψ_n) is the inner product with the m'th row vector of $Z^\dagger$, and hence

$$(\phi_m, \phi_n) = \delta_{nm} N \Rightarrow \|\phi_n\| = \sqrt{N}.$$

■

By virtue of the unitarity property of Z the inverse transformation of Eqn. (4.8) is then given by

$$f_n = \frac{2\pi}{Nh} \sum_{m=0}^{N-1} Z^\dagger_{nm} g_m = \frac{2\pi}{Nh} \sum_{m=0}^{N-1} e^{2\pi inm/N} g_m. \tag{4.9}$$

Note that considered as functions defined on the whole real line, both f_n and g_m are periodic functions: $f_n \equiv f_{N+n}$ and $g_m \equiv g_{N+m}$. Hence, the spectral range $|k| \leq k_c$ is not unique. For an interpretation of the inverse DFT we turn to the inverse continuous Fourier transform and again use the trapezoidal rule for approximation, viz.,

$$f_n = \int_{-k_c}^{k_c} e^{ikx_n} g(k)\, \mathrm{d}k \approx \frac{2\pi}{Nh} \sum_{m'=-N/2}^{m'=N/2-1} e^{ik_{m'}x_n} g_{m'}. \tag{4.10}$$

We have used the natural frequency range given by the Nyquist theorem.

It is convenient however for both implementation purposes and for the intuitive appeal to the final formulas of the DTFs to have the same summation range in both cases. By using Eqn. (4.10) we get the wave number range $|k| \leq k_c$, and by using Eqn. (4.9) we get the range $0 \leq k \leq 2k_c$.

If we shift the negative indices in Eqn. (4.10) upward by N, we get Eqn. (4.9). But then we must bear in mind that the actual frequencies k_m are changed, according to Fig. 4.3. Taking $-N/2 < m < 0$ corresponds to $k > 0$, and taking $0 \leq m < N/2 - 1$ corresponds to $k < 0$. We may of course just use the range $0 \leq k \leq 2k_c$; the numerical results are equivalent due to the periodicity of e^{ikx}. Note that taking $-k_c$ or k_c at $m = N/2 - 1$ is equivalent.

In a similar fashion, one sees that the restriction to the interval $0 \leq x \leq a$ is not necessary. Shifting away only leads to multiplying g_m by a constant of magnitude 1.

The number h is in computational settings a quite small number. If we define $hG_m := g_m$, we may rid ourselves of this factor in the transformations. We now have the pair

$$f_n = \frac{2\pi}{N} \sum_{m=0}^{N-1} Z^\dagger_{nm} G_m. \quad \text{(DFT)}$$

and

$$G_m = \frac{1}{2\pi} \sum_{n=0}^{N-1} Z_{mn} f_n \quad \text{(inverse DFT)}.$$

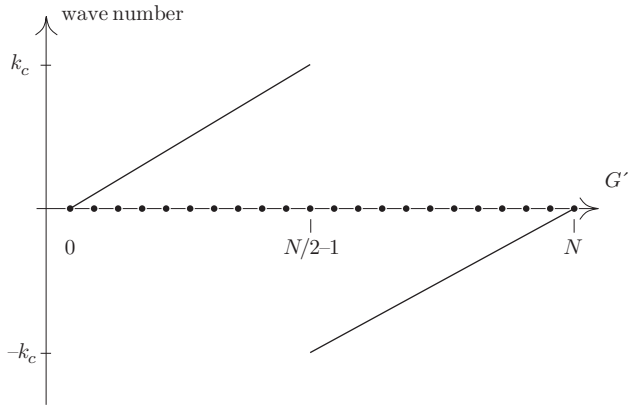

Figure 4.3: Adjusting the wave numbers

Note that h does not appear anywhere in the formulas, implying that the coefficients G_n is *independent* of h.

Because of the trapezoidal approximation nature of the definition, it is natural to take $\frac{\partial}{\partial x}$ to be a diagonal operator when applied to the transformed function (which is now a vector in $\mathbb{C}^N$), viz.,

$$[f'(x)]_n := \frac{1}{N}[Z^\dagger D(Z\tilde{f})]_n,$$

where $\tilde{f}$ is the vector whose components are f_n and D is a diagonal matrix whose first $N/2$ diagonal elements D_{mm}, $m = 0, \ldots, N/2-1$ are given by $2\pi/Nh$. The last $N/2-1$ diagonal elements are then given as $D_{m+N,m+N} = 2\pi/Nh$. It can be shown that the DFT provides an approximation to $f'(x)$ of order N, i.e., we have exponential convergence of the differential operator. Intuitively this is so because the spectral method uses *all values* f_n to compute the derivative, not just the ones in the immediate vicinity of x_n.

4.3.2 A Simple Implementation in Matlab

Here we present a very simple implementation of the spectral method applied to a time-independent model problem. More specifically, we wish to study the scattering of a wave packet onto a square barrier potential. We will also make a brief comparison of the DFT results with finite difference results. Further error analysis will not be performed in this example.

The Hilbert space for this problem is given by

$$\mathcal{H} = L^2([-c, c]),$$

where c is chosen large enough so that we may neglect interference of the wave packet with itself due to the boundary conditions. The Hamiltonian is

$$H = T + V,$$

with $T = -\partial^2/\partial x^2$ and

$$V(x) = \begin{cases} a & x \in [-b, b], \\ 0 & \text{otherwise.} \end{cases}$$

Here, a is a positive constant defining the strength of the barrier, and $2b$ is the width.

As initial condition we use a Gaussian wave packet centered at x_0 with initial momentum k_0 and width σ, i.e.,

$$\psi_0(x) = \frac{1}{(2\pi\sigma^2)^{1/4}} e^{-(x-x_0)^2/2\sigma^2} e^{-ik_0 x}.$$

We will choose x_0 and k_0 so that the wave is neglible outside the barrier and so that it is travelling towards it from the left.

Since we are going to use the DFT for implementing the differential operator, we employ periodic boundary conditions.

For integration in time of the Schrödinger equation we use a split-operator method (see Ref. [36]) which approximates the propagator to third order in the time step size τ, viz.,

$$U_\tau = e^{-i\tau T/2} e^{-i\tau V} e^{-i\tau T/2} = U(t_0 + \tau, t_0) + \mathcal{O}(\tau^3).$$

Note that V is diagonal in position representation and that T is diagonal when applied to the Fourier transformed wave function. Therefore, the algorithm for obtaining the (approximate) wave function at time $t + \tau$ reads:

1. Use DFT on the numerical solution.
2. Apply $e^{-i\tau T/2}$ (a diagonal operator.)
3. Use inverse DFT.
4. Apply $e^{-i\tau V/2}$ (again a diagonal operator.)
5. Use DFT on the numerical solution.
6. Apply $e^{-i\tau T/2}$ (a diagonal operator.)
7. Use inverse DFT.

If we perform this scheme several times in succession, we are doing quite a few redundant Fourier transforms. Usually one combines the last and the first application of $e^{-i\tau T/2}$ if no output of the wave function is desired in between time steps. For this simple problem however, it is no problem to waste a few computer cycles.

The integration scheme is also applicable to finite difference methods. If we use the $-\delta_x\delta_x$ difference approximation to T, then we have by the definition of the exponential operator

$$e^{i\tau\delta_x\delta_x/2} = 1 + i\frac{\tau}{2}\delta_x\delta_x - \frac{\tau^2}{4}(\delta_x\delta_x)^2 + \mathcal{O}(\tau^3),$$

and the error of this approximation is of the same order as the scheme itself. The $\delta_x\delta_x$ operator with periodic boundary conditions is easily implemented with a sparse matrix in Matlab, making the implementation quick and easy to read. See appendix B for the code.

In our program we implement both the spectral method and the finite difference method for comparison.

Fig. 4.4 shows a series of numerical wave functions for a simulation with $a = 28$, $b = 2.5$, $x_0 = -10$, $k_0 = 5$ and $\sigma^2 = 4$. Parameters for the numerical methods were $N = 1024$ and $\tau = 0.00025$. Note that the kinetic energy of the wave packet is $k_0^2 = 25 < a$, so that classically the particle is not allowed to pass the barrier. Nevertheless we see that some of the packet passes the obstacle, i.e., there is some probability that the particle upon measurement of its position will be found in the classically forbidden regime.

Performing numerical experiments of this kind can give very rich insight into the behavior of the system. This particular case can display resonance phenomena such as

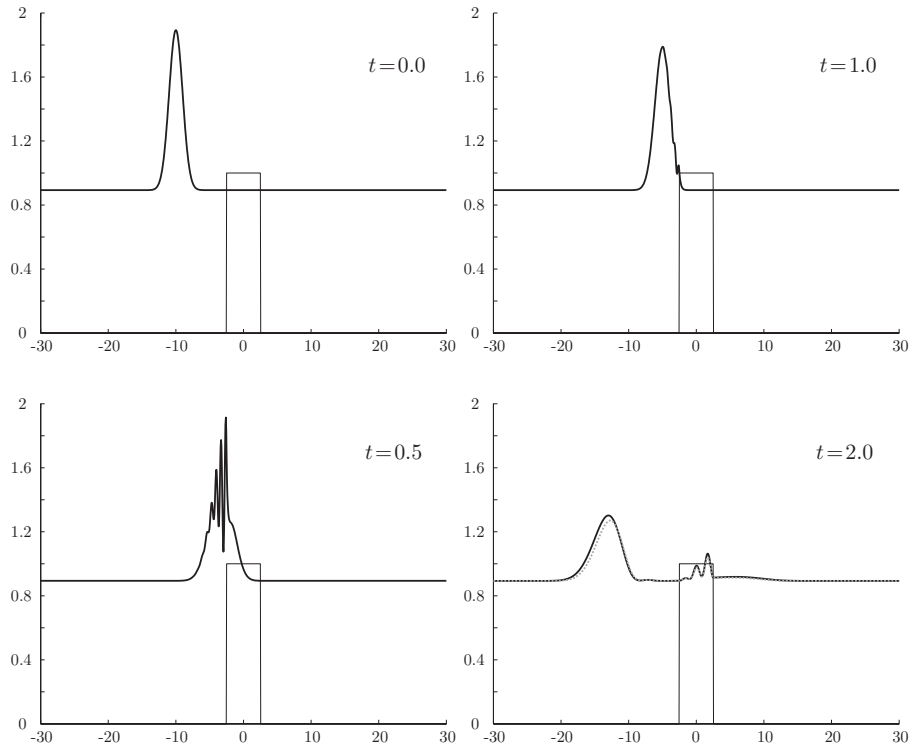

Figure 4.4: Results from running a simple simulator using the spectral method and split-operator time stepping.

some of the probability getting caught *inside the barrier*. (This is actually hinted at in the picture at $t = 2.0$.) Furthermore, we see an interference pattern at the moment of collision. What is the width of the fringes? What parameters in the simulation does it vary with? Another interesting experiment is to measure the particle's mean position at each side of the wall as function of t and compare this with the classical results. Animations of systems of this kind can be reached from Ref. [5]. These are produced with finite difference methods and the leap-frog scheme for time integration. (See section 4.5 for details on the leap-frog scheme.) These simulations were used in lectures with the aim that the students should gain some insight into the behavior of quantum mechanical systems.

In the last picture, the finite difference solution is also shown for comparison. Qualitatively we see that they match very well, indicating that both the finite difference method and the spectral method yield reasonable results.

4.4 Finite Element Methods

In this section we introduce the finite element method; a powerful class of numerical methods for solving partial differential equations. The perhaps most intuitively appealing aspect of the method is the handling of complicated geometries. Furthermore, the method is formulated in a highly modularized way, giving object-oriented languages such as C++ a great advantage when one wishes to implement the methods.

We start out by introducing the finite element method in a rather informal way, emphasizing the algorithm over the numerical properties of the method. We begin by introducing the weighted residual method of which the finite element method is a special case. Next, we compute an example in one dimension and briefly describe

instances of the method in higher dimensions.

Finally, implementations of the finite element method with the aid of the programming library Diffpack for C++, which is the combination of tools used in this thesis, are discussed.

Throughout the discussion we will use the typical Hamiltonian

$$H = -\nabla^2 + V(\boldsymbol{x})$$

as a prototype for differential operators that we need to discretize. (We omit constants such as $\hbar$ and μ.)

Finite elements are typically used in the spatial domain. For time dependent problems we use some difference scheme in time, leading to a sequence of spatial problems that we may solve with the finite element method. Hence, we will consider stationary problems in the introduction to keep it simple.

Consider for simplicity the backward Euler method in time, i.e.,

$$\frac{1}{\Delta t}\left(u^{\ell+1} - u^{\ell}\right) = -iHu^{\ell},$$

where u^{ℓ} denotes the (complex) wave function at time $t_\ell = \ell\Delta t$. This equation may be written

$$(1 + \Delta t i H)u^{\ell+1} = u^{\ell},$$

or

$$Au = f, \tag{4.11}$$

where $A = 1 + \Delta t i H$, $u = u^{\ell+1}$ is the unknown and $f = u^{\ell}$. Eqn. (4.11) is the prototype of the kind of equations we solve in both finite difference and finite element methods. We see that the operator A is a simple function of the Hamiltonian, and this is the case in all our applications. Hence, it is important to understand how operators such as ∇^2, $V(\boldsymbol{x})$ and so on are discretized with the finite element method.

The PDE is defined on some domain $\Omega \subset \mathbb{R}^d$, where d is the dimension of our system. Finite element methods require Ω to be a compact domain, i.e., closed, bounded and path-connected. For unbounded quantum mechanical systems we must in some way choose an appropriate bounded representative of whole space.

We also have some kind of boundary conditions in our PDE. Typically, we may have Dirichlet boundary conditions or Neumann conditions. Divide the boundary $\Gamma = \partial\Omega$ into two parts: Γ_D and Γ_N, on which we impose Dirichlet and Neumann conditions, respectively:

$$\begin{aligned} u(\boldsymbol{x}) &= g(\boldsymbol{x}), \quad \boldsymbol{x} \in \Gamma_D, \\ \text{and} \quad \frac{\partial u}{\partial n} = \boldsymbol{n} \cdot \nabla u(\boldsymbol{x}) &= h(\boldsymbol{x}), \quad \boldsymbol{x} \in \Gamma_N, \end{aligned}$$

where $\boldsymbol{n}$ is the unit normal on Γ_N.

4.4.1 The Weighted Residual Method

In the weighted residual method we define a subspace $V_h \subset V$ as

$$\begin{aligned} V_h &= \operatorname{sp}\left\{N_1(\boldsymbol{x}), N_2(\boldsymbol{x}), \ldots, N_m(\boldsymbol{x})\right\}, \\ \dim V_h &= m. \end{aligned}$$

Hence, N_i form a basis for V_h.[7] In V_h we seek our discrete approximate solution u_h.

[7] The subscript h usually denotes some kind of discretization. Recall that h is usually used for the mesh width in discretization formulations.

Any element $u_h \in V_h$ can be written

$$u_h = \sum_{i=1}^{m} u_j N_j,$$

and we define a vector $U \in \mathbb{C}^m$ by letting $U_j = u_j$. Note the fundamental difference from finite difference methods in which we ignore the behavior of u in between grid points. In the weighted residual method the discrete function is exactly that; a *function* defined in the whole domain.

Next, for any $u \in V$ define *the residual* R through

$$R := Au - f.$$

The residual vanishes if and only if u is a solution of our prototype linear PDE (4.11).

The idea of the weighted residual method is to choose m functions $W_i \in V$ with which we weight the residual R, i.e., we take the inner product. Requiring this inner product to vanish for each i then forces R to be small (in some sense) and leads to m equations for the m unknown coefficients u_i. In other words, we require that

$$(W_i, R) = 0, \quad i = 1, \ldots, m.$$

By linearity of A we obtain

$$\sum_{j=0}^{m} u_j (W_i, AN_j) = (W_i, f).$$

Clearly, this is a matrix equation of finite dimension. Define the matrix A_h as

$$(A_h)_{ij} := (W_i, AN_j) = \int_\Omega W_i^*(\boldsymbol{x}) A N_j(\boldsymbol{x}) \, \mathrm{d}\Omega$$

and the vector b as

$$b_i := (W_i, f) = \int_\Omega W_i^*(\boldsymbol{x}) f(\boldsymbol{x}) \, \mathrm{d}\Omega,$$

and the weighted residual method becomes equivalent to the matrix equation

$$A_h U = b.$$

If this equation has a unique solution, we have found the unique discrete function $u_h \in V_h$ such that the residual is orthogonal to all weighting functions W_i. If we define

$$W = \operatorname{span}\{W_1, W_2, \ldots W_m\} \subset V,$$

then R is orthogonal to all vectors in W, i.e., it belongs to the orthogonal complement $W^\perp$ of W. If W spans a large portion of V then $W^\perp$ must be only a small portion of V (in some intuitive sense; $\dim W^\perp$ may easily be infinite!)

If our PDE is non-linear our system of algebraic equations also becomes non-linear. This is a more complicated problem to solve, and methods such as the Newton-Rhapson iteration or successive substitution are popular solution methods, see Ref. [34].

There are of course many different ways of choosing the weighting functions W_i. We mention two popular choices here.

– *The collocation method,* in which we require the residual to vanish at m specified points, i.e., the weighting functions are given by

$$W_i(\boldsymbol{x}) = \delta(\boldsymbol{x} - \boldsymbol{x}^{[i]})$$

for each of the m points $\boldsymbol{x}^{[i]}$.

– *Galerkin's method,* in which we require R to be orthogonal to V_h which is equivalent to R being orthogonal to the basis functions N_i. Hence,

$$W_i = N_i$$

are the weighting functions.

We will exclusively use Galerkin's method in this text. When considering the numerical properties of the weighted residual method and the finite element method, Galerkin's method shows some remarkable and fortunate properties, such as best approximation properties in different norms and equivalence with minimation of a functional over V and V_h, respectively, in the continuous and discrete problems.

Galerkin's method written out in terms of our prototype differential equation then reads

$$\sum_{j=1}^{m} u_j(N_i, AN_j) = (N_i, f), \quad i = 1, \ldots, m.$$

4.4.2 A One-Dimensional Example

We have introduced the weighted residual method, of which the finite element method is a special case. The finite element method defines the basis functions N_i and in a quick introduction such as this it is best explained through an example.

When solving the discrete equations numerically the particular form of A_h matters. First, if N_i are far from orthogonal we will in general obtain a dense matrix A_h which requires a lot of computer time and storage to process. Second, the matrix may become ill-conditioned, rendering the solution sensitive to round-off errors. Hence, we should choose orthogonal (or nearly orthogonal) basis functions. The finite element method in a natural way ensures this.

The finite element basis functions N_i are defined in conjunction with the grid. The grid in finite element methods is somewhat more complicated than in finite difference methods. In addition to defining grid points called *nodes*, the domain is also divided into disjoint *elements* with which the nodes are associated. The elements play a fundamental role when defining the basis functions.

For the one-dimensional example, consider $\Omega = [0, 1]$ and choose m points $x^{[k]}$ such that

$$0 = x^{[1]} < x^{[2]} < \cdots < x^{[m]} = 1.$$

The grid points, or nodes, naturally divide Ω into sub-intervals $\Omega_e = [x^{[e]}, x^{[e+1]}]$. These sub-intervals are our *elements*, and we see that we have $n_e = m - 1$ elements.

Now we define our m basis functions N_i:

– Each N_i should be a simple polynomial over each element Ω_e. Thus, u_h becomes a piecewise polynomial function. In our case we choose linear functions.

– Fundamental in the finite element method is the requirement

$$N_i(x^{[j]}) = \delta_{ij}.$$

This imposes as many conditions on each polynomial as there are nodes in an element. In our case we have two nodes per element, which implies that N_i must be linear over each element.

Note that

$$u_h(x^{[i]}) = \sum_j u_j N_j(x^{[i]}) = \sum_j u_j \delta_{ij} = u_i,$$

i.e., u_i is u_h evaluated at a node point.

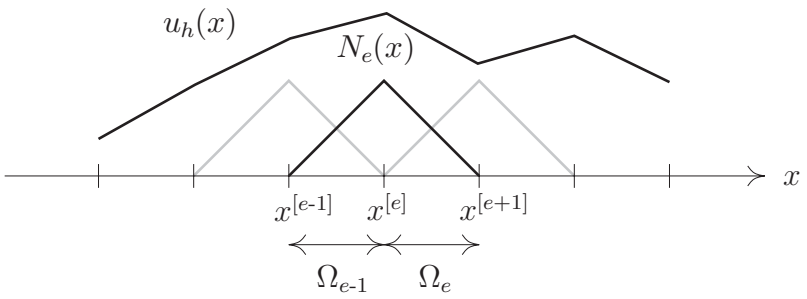

Figure 4.5: Linear elements in one dimension. A sample basis function N_k is showed together with a piecewise linear function u_h.

Fig. 4.5 illustrates our one-dimensional grid. If we let $h_e = |\Omega_e|$, then it is easy to see that for $i = 2, \ldots, m-1$, $N_i(x)$ is given by

$$N_i(x) = \begin{cases} 0 & x \notin \Omega_{i-1} \cup \Omega_i \\ \frac{1}{h_{i-1}}(x - x^{[i-1]}) & x \in \Omega_{i-1} \\ 1 - \frac{1}{h_i}(x - x^{[i]}) & x \in \Omega_i \end{cases}.$$

For $i = 1$ or $i = m$ the definition is similar, i.e.,

$$N_1(x) = \begin{cases} 0 & x \notin \Omega_1 \\ 1 - \frac{1}{h_1}(x - x^{[1]}) & x \in \Omega_1 \end{cases}$$

and

$$N_m(x) = \begin{cases} 0 & x \notin \Omega_{m-1} \\ \frac{1}{h_{m-1}}(x - x^{[m-1]}) & x \in \Omega_{m-1}. \end{cases}$$

Hence, the element functions vanish identically over most of the grid. The "tent functions" are also almost orthogonal, as $(N_i, N_j) = 0$ whenever $x^{[i]}$ and $x^{[j]}$ does not belong to the same element. It is also easy to see that they can be combined to create any piecewise linear function with $x^{[k]}$ defining the joints of the linear functions.

It is easy to see that the nodal point condition implies that

$$(N_i, N_j) = \int_\Omega N_i N_j \, \mathrm{d}\Omega$$

vanishes when i and j are nodal indices belonging to different elements. Looking forward, we see that matrices whose elements are on the form (AN_i, BN_j) with A and B arbitrary linear operators obtain the same sparse structure.

Let us sketch a weighted residual statement with these basis functions. We wish to solve

$$Au = f, \quad A = 1 - \nabla^2.$$

We choose as boundary conditions

$$u(0) = C_1, \quad u'(1) = C_2,$$

a Dirichlet and a Neumann condition, respectively. The Dirichlet condition automatically implies $u_1 = C_1$. As we will see the Neumann condition enters when we rephrase our equations into *weak form*; see also section 4.6 for more on the weak form.

Note that N_i'' is not defined, since N_i' is piecewise linear and hence cannot be differentiated.[8] The weighted residual statement reads

$$\sum_{j=1}^{m} u_j(N_i, N_j - N_j'') = (N_i, f), \quad i = 1, \ldots, m.$$

In other words,

$$\sum_{j=1}^{m} u_j \left(\int_\Omega N_i(x)N_j(x) - \int_\Omega N_i(x)N_j''(x) \right) = \int_\Omega N_i(x)f(x),$$

where we have used that N_i are real functions. The second order derivative is eliminated by integrating by parts, viz.,

$$\int_\Omega N_i(x)N_j''(x) = -\int_\Omega N_i'(x)N_j'(x) + \left[N_i(x)N_j'(x)\right]_0^1.$$

This gives

$$\sum_{j=1}^{m} u_j \int_\Omega N_i(x)N_j(x) + \int_\Omega N_i'(x)N_j'(x) + N_i(0)u_h'(0) - N_i(1)u_h'(1) = \int_\Omega N_i(x)f(x).$$

The boundary term at $x = 0$ seems troublesome as we do not know $u_h'(0)$. However, equation $i = 1$ is eliminated due to the left boundary condition, a Dirichlet condition. As $N_i(0) = 0$ for the remaining equations, the term drops out. The boundary term at $x = 1$ involves the Neumann condition, and we have

$$N_i(1)u_h'(1) = \delta_{im}C_2.$$

Not only did we get rid of the second derivatives by integration by parts, but we introduced the Neumann condition into the discrete system in a convenient way.

Finally we obtain the equations

$$u_1 = C_1, \tag{4.12}$$

$$\sum_{j=1}^{m} u_j \int_\Omega N_i(x)N_j(x) + \int_\Omega N_i'(x)N_j'(x) = \int_\Omega N_i(x)f(x) - \delta_{im}C_2, \quad i = 2, \ldots, m. \tag{4.13}$$

This is a set of m linear equations in the unknowns u_i.

There are two matrices that appear naturally in the equations; the stiffness matrix K and the mass matrix M. They are defined by

$$K_{ij} := \int_\Omega \nabla N_i \cdot \nabla N_j \, \mathrm{d}\Omega \tag{4.14}$$

and

$$M_{ij} := \int_\Omega N_i N_j \, \mathrm{d}\Omega. \tag{4.15}$$

They have a tendency to appear in finite element formulations and it is wise to get to know them for the most used element types, such as the linear elements in one dimension. They are very sparse in large systems, and this indicates that iterative methods could be used to solve them with great efficiency, see chapter 5.

Let us sum up the general results from these calculations:

[8] Strictly speaking, neither can N_i be differentiated as N_i' is undefined as the nodes. Whatever value we choose for $N_i'(x^{[j]})$ does however not contribute to the integrals. This is connected with the concept of *weak derivatives*, see Ref. [17].

– Galerkin's method leads to an m-dimensional system of linear equations,

$$A_h U = b,$$

where $A = M + K$ and $b_j = (N_j, f)$ except for modifications due to boundary conditions.

– A Dirichlet boundary condition at node $\boldsymbol{x}^{[k]}$ is enforced by replacing equation k in the system with the boundary condition, i.e., $A_{kk} = 1$, $A_{kj} = 0$ whenever $j \neq k$, and $b_k = g(\boldsymbol{x}^{[k]})$, where g is the prescription of u at the boundary.

Dirichlet conditions are called *essential boundary conditions* in finite element contexts because their incorporation is done immediately in the linear system.

– Neumann boundary conditions are imposed by integration by parts of second order derivatives. This leads to extra terms in the right hand side vector b.

Neumann conditions and other boundary conditions involving derivatives are called *natural boundary conditions* in finite element contexts. They "appear naturally" in the process of integrating by parts.

4.4.3 More on Elements and the Element-By-Element Formulation

In this section we will briefly describe the generalization of the ideas presented in the one-dimensional example above.

In the one-dimensional case the shape of the elements Ω_e becomes rather limited. They are simply intervals of varying length. In higher dimensions we have greater freedom of choosing element shapes. For example, if linear elements (i.e., linear N_i) are employed, the shape of Ω_e may be any quadrangle. Furthermore, triangular shapes may be used, and indeed this is a very popular choice because very good algorithms exists for dividing a region Ω into triangles. In three dimensions we would use deformed parallelepipeds and tetrahedrons.

A simple example is depicted in Fig. 4.6. A simple rectangular grid is subdivided into equal square slabs. The nodes are located at the corners of each slab, and an element function is depicted at a particular node. Note the obvious tensor-generalization of the one-dimensional case, see Ref. [34] for details on tensor-product generalizations of one-dimensional element types.

Clearly, the process of calculating the element basis becomes rather complicated with increasing complexity of the geometries, location and shapes of elements and so on. All geometric variants of an element are reflected in the expressions for the corresponding basis functions over that element. Hence, it is natural to use a reference element and map the results from this element back to the real geometry. This is done in the *element-by-element formulation*, which we briefly describe here. For more details, see Ref. [34].

The idea is to first rewrite the integral over Ω as a sum of the integrals over each element Ω_e, viz.,

$$A = \sum_e A_e, \quad \text{and} \quad b = \sum_e b_e,$$

where each term in the sums are by definition identical to A and b but with integration only over Ω_e. The mass matrix for example becomes

$$M_{ij} = \sum_e M_{ij}^e = \sum_e \int_{\Omega_e} N_i N_j \, \mathrm{d}\Omega_e. \tag{4.16}$$

Next, one observes that due to the localized character of each basis function N_i, almost all of the components of A_e and b_e are zero. In fact, if i and j are indices that correspond to nodes outside of element e, then the basis functions N_i are zero and hence $(A_e)_{ij}$ and

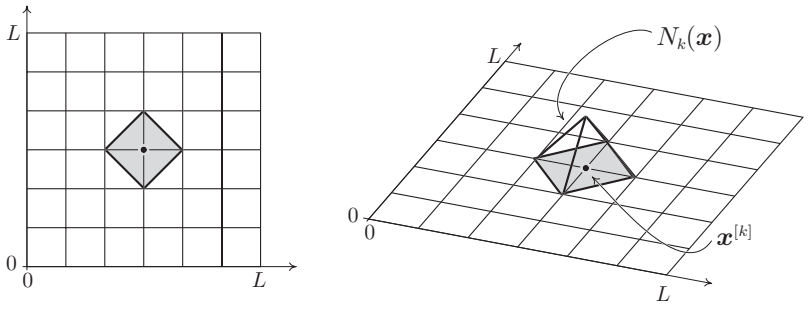

Figure 4.6: Linear elements in two dimensions. The nodes are located at line intersections.

$(b_e)_i$ are also automatically zero. Each element in our one-dimensional example has two nodes, hence A_e can be represented by a 2×2-matrix $\tilde{A}_e$ and b_e can be represented by a two-dimensional vector $\tilde{b}_e$.

In our example, we choose a reference element $\xi \in [-1, 1] = \tilde{\Omega}$. We use a linear change of coordinates to map Ω_e into $\tilde{\Omega}$. Then only two basis functions do not vanish over $\tilde{\Omega}$. The two nodes $\xi = \pm 1$ correspond to $x^{[e]}$ and $x^{[e+1]}$, respectively. Clearly, there is a mapping $q(e, r)$, where $r = 1, 2$ is the local node number, that maps the element number e and node number r into the global node number k. This mapping exists in more general cases as well, but usually it is only known from a table due to complex geometries.

The mapping from local coordinates to global coordinates is

$$x^{(e)}(\xi) = x^{[e]} + \frac{1}{2}(\xi + 1)(x^{[e+1]} - x^{[e]}).$$

Hence,

$$N_i(x^{(e)}(\xi)) = N_{q(e,r)}(x) = \tilde{N}_r(\xi)$$

is the global basis function in terms of the local basis function $\tilde{N}_r(\xi)$. In Fig. 4.7 this is illustrated. Integration over the reference element introduces the Jacobian of the coordinate change into the integrand.

The linear one-dimensional elements are examples of so-called *isoparametric elements*. Such elements are characterized by the fact that the *same* mapping is used both for interpolating u_h and for mapping from local to global coordinates. I.e., $\tilde{N}_r(\boldsymbol{\xi})$ defines both the basis functions and the mapping from local to global coordinates. The

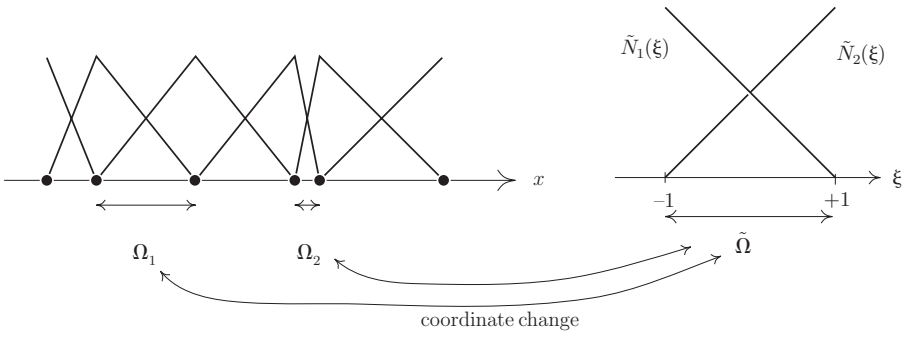

Figure 4.7: Illustration of local coordinates for one-dimensional linear elements.

(inverse) coordinate change in isoparametric elements are in general given by

$$\boldsymbol{x}^{(e)}(\boldsymbol{\xi}) = \sum_{r=1}^{n_{no}} \tilde{N}_r(\boldsymbol{\xi}) \boldsymbol{x}^{[q(e,r)]},$$

where n_{no} is the number of nodes in each element.

The introduction of local coordinates eases the implementation of a finite element solver and we also realize the modularized nature of the finite element method and hence also the appropriateness of object-oriented programming techniques in this case. Everything from the linear equations, via the grid and the elements to the assembly routines are actually defined in an object-oriented manner, so to speak.

4.5 Time Integration Methods

In its simplest form, we again state the time dependent Schrödinger equation:

$$\frac{\partial}{\partial t}\Psi = -iH(t)\Psi.$$

Again, Ψ is a complete quantum state, encapsulating space- and spin-degrees of freedom that the particle might have. The Hamiltonian $H(t)$ contains some spatial dependencies through the kinetic energy term (i.e., spatial second-order differentiation) and the potential energy. For example,

$$H(t) = -\nabla^2 + V(\boldsymbol{x}, t),$$

in the case of a spinless particle under the influence of the time dependent potential $V(\boldsymbol{x}, t)$. The Schrödinger equation imposes stronger constraints on the state than the states allowed in quantum theory. It must be at least twice differentiable with respect to spatial coordinates, for example.

The time dependence is our concern in this section. Naturally, we will use some kind of finite difference approximation to the time derivative, which yields a sequence of discrete problems, regardless of how we choose to handle the space dependencies and their approximations.

Our weapons of choice are *the leap-frog scheme*, an explicit integration scheme whose name points to the staggered nature of the time stepping, and *the theta-rule*, an implicit scheme covering both the forwards and backwards Euler schemes in addition to the popular Crank-Nicholson scheme.

What characterizes the quality of a numerical scheme? How do we decide whether it is a good scheme or not? What do we mean by "a good approximation?" Several factors contribute, among them the accuracy, the stability and the computational cost.

Clearly, when our time step becomes small enough our scheme should become more and more like the continuous differential equation. This is measured through the so-called *truncation error* τ.

When we write our PDE in homogenous form, i.e.,

$$\mathcal{L}(\Psi) = 0,$$

and formulate a discretization, viz.,

$$\mathcal{L}_\Delta(\Psi_h) = 0,$$

we define the truncation error τ as the residual we are left with when we insert a solution of the continuous PDE into the discrete version, viz.,

$$\tau = \mathcal{L}_\Delta(\Psi), \qquad \text{where} \quad \mathcal{L}(\Psi) = 0.$$

We use Ψ_h for the time-discretized wave function. It should not be confused with the spatially discretized wave function from the previous sections. The expression for τ and also for the integration scheme are purely formal. The iteration process implies that we must apply operators like the differential operator more then once. We have not assumed that Ψ is more than two times differentiable. However, the formality vanishes when we apply a spatial discretization. For instance, the operator $\delta_x\delta_x$ may always be applied to a discrete function.

Clearly, if τ does not vanish when the temporal mesh width becomes infinitely small, $\mathcal{L}_\Delta$ cannot represent a good approximation to the equation. In the opposite case of a vanishing τ, we say that the numerical scheme is *consistent*.

We also need *stability*. Stability may be defined as the property that the numerical solution Ψ_h reflects the qualitative properties, i.e., the physical properties, of the continuous solution Ψ. If this is not fulfilled, then our approximation is useless! We want physical solutions.

This demand is somewhat informal, but it is almost always easy to devise the proper requirement for a given PDE. If both stability and consistency is present, then a famous theorem by Lax states that our numerical solution is convergent, that is the solution to the numerical equations converges to the continuous ones in the limit of vanishing grid spacings. The converse is also true. See Ref. [34]. We study this further in section 4.6.

The stability criterion we are searching for in the case of the time dependent Schrödinger equation is conservation of probability, i.e., the norm of the wave function. Basic quantum mechanics teaches us that the norm of the wave function Ψ is conserved at all times, and this property we also demand from the numerical solution in order to ensure that it converges to the true solution by Lax' theorem.

4.5.1 The Theta-Rule

The theta-rule is a means for discretizing a first order derivative on the form

$$\frac{\partial\psi}{\partial t} = G. \tag{4.17}$$

Here, G is some arbitrary expression that we assume to be differentiable so we may take the Taylor expansion of G around some arbitrary time to obtain the expression at some other time.

The θ-rule is an interpolation between the forwards and backwards Euler discretizations and is defined through

$$\frac{1}{\Delta t}\left(\Psi_h^{\ell+1} - \Psi_h^{\ell}\right) = \theta G_h^{\ell+1} + (1-\theta)G_h^{\ell}. \tag{4.18}$$

We comment that the derivations in this section are *formal*, in the sense that we do not actually know whether the quantities are defined in all circumstances. For example, we do not know whether $G^2\Psi_h$ is a function. If $G = -\nabla^2$ then Ψ_h must be four times differentiable and so on. On the other hand, when introducing spatial discretization the problem vanishes. For example, the difference operator is *always* defined on any discrete function.

Reorganizing Eqn. (4.18) yields the updating scheme for the numerical solution:

$$\left[\mathbf{1} + i\theta\Delta t H^{\ell+1}\right]\Psi_h^{\ell+1} = \left[\mathbf{1} - i(1-\theta)\Delta t H^{\ell}\right]\Psi_h^{\ell}. \tag{4.19}$$

When discretizing a differential equation on the form of Eqn. (4.17) it is common to consider the forwards and backwards Euler schemes. For the Schrödinger equation though, neither one is particularly suitable. Forwards Euler is notoriously unstable and the backwards variant contains damping, making the scheme non-unitary. We

recover forwards and backwards Euler with $\theta = 0$ and $\theta = 1$, respectively. (The claimed stability and damping will be clear after we have analyzed the θ-rule.) The case $\theta = 1/2$ gives us the Crank-Nicholson scheme and is of special interest which will be clear later.

Let us derive the truncation error of the θ-rule.

We start by considering the left hand side of Eqn. (4.18) and insert an exact solution. We formally expand the Taylor series of $\Psi^{\ell+1}$ around t_ℓ, viz.,

$$\frac{1}{\Delta t}\left(\Psi^{\ell+1} - \Psi^{\ell}\right) = \frac{1}{\Delta t}\left(e^{\Delta t \partial_t} - 1\right)\Psi^{\ell}. \tag{4.20}$$

Note the notation for the Taylor series. To second order we obtain

$$\frac{1}{\Delta t}\left(\Psi^{\ell+1} - \Psi^{\ell}\right) = \frac{\partial \Psi^{\ell}}{\partial t} + \frac{\Delta t}{2}\frac{\partial^2 \Psi^{\ell}}{\partial t^2} + \frac{\Delta t^2}{6}\frac{\partial^3 \Psi^{\ell}}{\partial t^3} + \mathcal{O}(\Delta t^3). \tag{4.21}$$

This is nothing but the familiar truncation error for a forward difference, see section 4.2. Consider the right hand side in a similar fashion, viz.,

$$\theta G^{\ell+1} + (1-\theta)G^{\ell} = \theta e^{\Delta t \partial t} G^{\ell} + (1-\theta)G^{\ell}. \tag{4.22}$$

Expanding the series we obtain

$$\frac{\partial \Psi^{\ell}}{\partial t} + \frac{\Delta t}{2}\frac{\partial^2 \Psi^{\ell}}{\partial t^2} + \frac{\Delta t^2}{6}\frac{\partial^3 \Psi^{\ell}}{\partial t^3} = G^{\ell} + \theta \Delta t \frac{\partial G^{\ell}}{\partial t} + \theta \frac{\Delta t^2}{2}\frac{\partial^2 G^{\ell}}{\partial t^2} + \mathcal{O}(\Delta t^3). \tag{4.23}$$

The PDE (4.17) written in homogenous form is

$$\mathcal{L}(\Psi) = \frac{\partial \Psi}{\partial t} - G(\Psi, t) = 0,$$

and our discretization is correspondingly

$$\mathcal{L}_\Delta(\Psi_h) = \frac{1}{\Delta t}\left(\Psi_h^{\ell+1} - \Psi_h^{\ell}\right) - \theta G^{\ell+1} - (1-\theta)G^{\ell} = 0.$$

The truncation error τ is defined as

$$\tau = \mathcal{L}_\Delta(\Psi). \tag{4.24}$$

We insert the results obtained in Eqns. (4.21) and (4.23) and when noting that

$$\frac{\partial^n G^{\ell}}{\partial t^n} = \frac{\partial^{n+1} \Psi^{\ell}}{\partial t^{n+1}},$$

we obtain

$$\tau = \mathcal{L}_\Delta(\Psi^{\ell}) = \Delta t(\frac{1}{2} - \theta)\frac{\partial^2 \Psi^{\ell}}{\partial t^2} + \Delta t^2(\frac{1}{6} - \frac{\theta}{2})\frac{\partial^3 \Psi^{\ell}}{\partial t^3} + \mathcal{O}(\Delta t^3). \tag{4.25}$$

Hence, the theta-rule has truncation error of order $\mathcal{O}(\Delta t)$ for $\theta \neq 1/2$, but in the case of $\theta = 1/2$ we obtain a truncation error of order $\mathcal{O}(\Delta t^2)$. This particular case is called the *Crank-Nicholson-scheme* and is very well suited for solving the time dependent Schrödinger equation; not only because of its higher accuracy, but we shall also see that we obtain unitarity of the scheme. Thinking about it, since the forward Euler scheme makes the solution unstable in the sense that the norm of the solution is growing, and the backwards Euler scheme behaves in the opposite way, it is not unnatural to expect that for some θ these effects are exactly balanced.

That our scheme is of second order is fine, but another important question is: What is the error in Ψ_h^{ℓ} after ℓ time integration steps in time? How much does it differ from Ψ^{ℓ}, which is the exact solution found from solving the continuous differential equation (4.17), when we in both cases start with the same initial condition? In the case of the θ-rule the answer is somewhat delicate, but in the leap-frog case it is rather trivial. The trouble is that $\Psi_h^{\ell+1}$ is implicitly defined in Eqn. (4.19).

4.5.2 The Leap-Frog Scheme

We now turn to the leap-frog scheme. In the context of the time dependent Schrödinger equation this scheme was first proposed by Askar and Cakmak in 1978 (see Ref. [37]) as an alternative to the implicit Crank-Nicholson scheme. Given a first order differential equation on the form of Eqn. (4.17) we simply use a centered difference on the left hand side and evaluate the right hand side in-between, viz.,

$$\frac{1}{2\Delta t}\left(\Psi_h^{\ell+1} - \Psi_h^{\ell-1}\right) = G(\Psi_h^{\ell}, t_\ell). \tag{4.26}$$

Reorganizing this equation yields for the new wave function $\Psi^{\ell+1}$

$$\Psi_h^{\ell+1} = \Psi_h^{\ell-1} - 2i\Delta t H^{\ell}\Psi_h^{\ell}. \tag{4.27}$$

Notice that we need two earlier wave functions Ψ_h^{ℓ} and $\Psi_h^{\ell-1}$ to find the new $\Psi_h^{\ell+1}$, i.e., twice the information needed in the theta-rule. In reality we will not need this double information. If we separate the real and imaginary parts of the wave function, i.e., write

$$\Psi_h^{\ell} = R_h^{\ell} + I_h^{\ell},$$

we obtain

$$\begin{aligned} R^{\ell+1} &= R^{\ell-1} + 2\Delta t H^{\ell} I^{\ell} \\ I^{\ell+2} &= I^{\ell} - 2\Delta t H^{\ell+1} R^{\ell+1}. \end{aligned}$$

If we assume that H^{ℓ} applied to a real vector again is a real vector, we have created an algorithm to update R and I alternately. In this case we say that we are using a staggered grid.[9]

On homogenous form we have

$$\mathcal{L}_\Delta(\Psi_h^{\ell}) = \frac{1}{2\Delta t}\left(\Psi_h^{\ell+1} - \Psi_h^{\ell-1}\right) - G(\Psi_h^{\ell}, t_\ell).$$

To find the truncation error we insert a solution Ψ^{ℓ} of the continuous problem in $\mathcal{L}_\Delta$ and expand $\Psi^{\ell\pm1}$ in Taylor series around t_ℓ. This yields for the time difference

$$\begin{aligned} \frac{1}{2\Delta t}\left(\Psi^{\ell+1} - \Psi^{\ell-1}\right) &= \frac{1}{\Delta t}\left[\sum_{n=0}^{\infty} \frac{\Delta t^{2n+1}}{(2n+1)!}\frac{\partial^{2n+1}\Psi^{\ell}}{\partial t^{2n+1}}\right] \\ &= \frac{\partial \Psi^{\ell}}{\partial t} + \frac{\Delta t^2}{6}\frac{\partial^3 \Psi^{\ell}}{\partial t^3} + \frac{\Delta t^4}{120}\frac{\partial^5 \Psi^{\ell}}{\partial t^5} + \mathcal{O}(\Delta t^6). \end{aligned} \tag{4.28}$$

Hence, the truncation error is of second order also for the leap-frog scheme, viz.,

$$\tau = \frac{\Delta t^2}{6}\frac{\partial^3 \Psi^{\ell}}{\partial t^3} + \mathcal{O}(\Delta t^4). \tag{4.29}$$

Note that in both the Crank-Nicholson case and the current leap-frog scheme is not only τ of second order, but the dominating term in the truncation error is also proportional to the third time derivative of Ψ, i.e.,

$$\tau \sim \frac{\partial^3 \Psi}{\partial t^3} = \frac{\partial^2 G}{\partial t^2}.$$

[9] The assumption does not hold in general. If ψ is a real function and if $H = -i\partial/\partial x$, $H\psi$ is purely imaginary.

4.5.3 Stability Analysis of the Theta-Rule

Recall from section 1.3 that the norm of the wave function Ψ is conserved at all times due to the unitarity of the evolution operator. In light of the probabilistic interpretation of the wave function this means that the probability of finding the particle at some place with some spin orientation at any time is unity. The unitarity of the evolution operator should of course be reflected in the numerical solution of our discretized equations, i.e., in the numerical evolution operator. The updating rules for the theta-rule and the leap-frog scheme in reality are approximations to the time evolution operator $U(t_{\ell+1}, t_\ell)$. In particular, for the theta-rule we have

$$U_\Delta^\ell = \left(\mathbf{1} + i\theta\Delta t H^{\ell+1}\right)^{-1} \left(\mathbf{1} - i(1-\theta)\Delta t H^\ell\right). \tag{4.30}$$

A simple special case is the case where H is independent of time. If ϵ_n is its discrete set of eigenvalues,[10] then the eigenvalues of U_Δ clearly are

$$\lambda_n = \frac{1 + i\theta\Delta t\epsilon_n}{1 - i(1-\theta)\Delta t\epsilon_n},$$

and the eigenvectors coincide with those of H. Furthermore, the norm of the eigenvalues are

$$|\lambda_n| = \left[\frac{1 + \theta^2\Delta t^2\epsilon_n^2}{1 + (1-\theta)^2\Delta t^2\epsilon_n^2}\right]^{1/2}.$$

It is easy to see that for $\theta = 1/2$, $|\lambda_n| \equiv 1$. Furthermore, $\theta < 1/2$ yields $|\lambda_n| > 1$ and $\theta > 1/2$ yields $|\lambda_n| < 1$, in other words unconditionally unstable and stable schemes, respectively, in the sense of amplification and damping of the solution.

For time dependent Hamiltonians however, the matter is more complicated.

Let us define two operators A and B so that $U_\Delta = A^{-1}B$, viz.,

$$U_\Delta^\ell = \underbrace{\left(\mathbf{1} + i\theta\Delta t H^{\ell+1}\right)}_{A}{}^{-1} \underbrace{\left(\mathbf{1} - i(1-\theta)\Delta t H^\ell\right)}_{B}.$$

Given some wave function Ψ_h^ℓ, the norm of the new wave function after evolving it with U_Δ^ℓ is

$$\left\|\Psi_h^{\ell+1}\right\|^2 = (\Psi_h^\ell, (U_\Delta^\ell)^\dagger U_\Delta \Psi_h^\ell) = (\Psi_h^\ell, B^\dagger (AA^\dagger)^{-1} B\Psi_h^\ell).$$

If the scheme is to be perfectly unitary, then $(U_\Delta^\ell)^\dagger U_\Delta$ must be the identity operator $\mathbf{1}$. This happens if the Hamiltonian is independent of time and $\theta = 1/2$. For the non-autonomous cases we cannot expect this to happen. We can only hope for unitarity up to some order of Δt.

First we note that

$$AA^\dagger = \mathbf{1} + \Delta t^2\theta^2 (H^{\ell+1})^2.$$

We need the inverse of this operator. In general, in a similar fashion as with ordinary scalar numbers, we have the power series expansion:

$$(\mathbf{1} - X)^{-1} = \sum_{k=0}^{\infty} X^k,$$

provided that all the eigenvalues of X is less than 1 in magnitude. This puts a restriction on Δt for the series to converge, i.e.,

$$\Delta t^2\theta^2(\epsilon_n^{\ell+1})^2 < 1, \quad \forall n,$$

[10] Even though H may have a continuous spectrum, any spatially discretized version will have a discrete spectrum with a finite number of eigenvalues.

where ϵ_n^ℓ are the (real) eigenvalues of H^ℓ.

We cannot invert A with a series in a general infinite dimensional Hilbert space, since the eigenvalues are not bounded in general.[11] However, we are going to do some kind of discretization in space, leaving the Hilbert space with a finite dimension and thus a bounded spectrum. We may then choose Δt small enough and perform the expansion of A^{-1}.

Increasing the dimension of the discrete Hilbert space will typically lower the typical mesh width h of the spatial grid used. This leaves us with a better approximation of the complete, infinite dimensional space and also more eigenvalues, which of course will grow in magnitude as we get more of them. Hence, the Δt-criterion must contain h in such a way that lowering h will automatically require a lower Δt.

Assume that the series is convergent, i.e.,

$$(AA^\dagger)^{-1} = \sum_{m=0}^{\infty} (-\theta \Delta t H^{\ell+1})^{2m} = \mathbf{1} - \Delta t^2 \theta^2 (H^{\ell+1})^2 + \Delta t^4 \theta^4 (H^{\ell+1})^4 + \mathcal{O}(\Delta t^6).$$

Multiplying with B from the right yields

$$\begin{aligned}(AA^\dagger)^{-1} B = \mathbf{1} - i(1-\theta)\Delta t H^\ell - \theta^2 \Delta t^2 (H^{\ell+1})^2 + i(1-\theta)\theta^2 \Delta t^3 (H^{\ell+1})^2 H^\ell \\ + \theta^4 \Delta t^4 (H^{\ell+1})^4 - i(1-\theta)\theta^4 \Delta t^5 (H^{\ell+1})^4 H^\ell + \mathcal{O}(\Delta t^6).\end{aligned}$$

Then, multiply with $B^\dagger$ from the left and note that the first order terms cancel, viz.,

$$\begin{aligned}B^\dagger (AA^\dagger)^{-1} B = \mathbf{1} + (1-\theta)^2 \Delta t^2 (H^\ell)^2 - \theta^2 \Delta t^2 (H^{\ell+1})^2 \\ - i(1-\theta)\theta^2 \Delta t^3 H^\ell (H^{\ell+1})^2 + i(1-\theta)\theta^2 \Delta t^3 (H^{\ell+1})^2 H^\ell \\ - (1-\theta)^2 \theta^2 \Delta t^4 H^\ell (H^{\ell+1})^2 H^\ell \\ + \theta^4 \Delta t^4 + \mathcal{O}(\Delta t^5)\end{aligned} \tag{4.31}$$

This final expression shows that the operator is of order $\mathcal{O}(\Delta t^2)$. However, an examination of the terms of this power reveals that if the Hamiltonian varies slowly enough, then they are of order $\mathcal{O}(\Delta t^3)$ in the $\theta = 1/2$ case. This is because

$$H^{\ell+1} = e^{\Delta t \partial_t} H^\ell,$$

so we get

$$(H^{\ell+1})^2 = (H^\ell)^2 + \Delta t (H^\ell \frac{\partial H^\ell}{\partial t} + \frac{\partial H^\ell}{\partial t} H^\ell) + \mathcal{O}(\Delta t^2).$$

Inserting this into Eqn. (4.31) yields

$$\begin{aligned}U_\Delta^\dagger U_\Delta &= \mathbf{1} - \frac{\Delta t^2}{4} \left[(H^{\ell+1})^2 - (H^\ell)^2 \right] + \mathcal{O}(\Delta t^3) \\ &= \mathbf{1} - \frac{\Delta t^3}{4} \left[H^\ell \frac{\partial H^\ell}{\partial t} + \frac{\partial H^\ell}{\partial t} H^\ell \right] + \mathcal{O}(\Delta t^4).\end{aligned} \tag{4.32}$$

If we insert Eqn. (4.32) into the norm of $\Psi_h^{\ell+1}$, we obtain

$$\left\| \Psi_h^{\ell+1} \right\|^2 = \left(1 + \mathcal{O}(\Delta t^3) \right) \left\| \Psi_h^\ell \right\|^2$$

in the Crank-Nicholson case. In the other cases like forward Euler, we cannot erase the $\mathcal{O}(\Delta t^2)$ term, and we know that this integration is unstable and explodes after just a few integration steps. Hence, the crucial point in the stability is this term, which we managed to get rid of.

[11] For example the hydrogen atom has a spectrum divided into two parts: A discrete bounded spectrum part of energies below zero, and an unbounded continuous spectrum part above zero.

4.5.4 Stability Analysis of the Leap-Frog Scheme

We will now analyze the stability of the leap-frog scheme, i.e., of

$$\Psi_h^{\ell+1} = \Psi^{\ell-1} - 2i\Delta t H^\ell \Psi_h^\ell. \tag{4.33}$$

Where we have reorganized Eqn. (4.26) to obtain the rule that updates Ψ_h. Using the truncation error $\tau = \mathcal{O}(\Delta t^2)$ yields

$$\Psi_h^{\ell+1} = \Psi^{\ell+1} + \mathcal{O}(\Delta t^3).$$

Hence, the numerical development of the wave function in time has an error of third order in Δt. We may use this in Eqn. (4.33) to find a good approximation to U_Δ^ℓ, viz.,

$$\begin{aligned}
\Psi_h^{\ell+1} &= \Psi_h^{\ell-1} - 2i\Delta t H^\ell \left(\Psi^\ell + \mathcal{O}(\Delta t^3)\right) \\
&= \Psi_h^{\ell-1} - 2i\Delta t H^\ell \left(e^{\Delta t \partial_t^{\ell-1}} \Psi^{\ell-1}\right) + \mathcal{O}(\Delta t^4) \\
&= \Psi_h^{\ell-1} - 2i\Delta t H^\ell \left(e^{\Delta t \partial_t^{\ell-1}} (\Psi_h^{\ell-1} + \mathcal{O}(\Delta t^3))\right) + \mathcal{O}(\Delta t^4) \\
&= \left[\mathbf{1} - 2i\Delta t H^\ell e^{\Delta t \partial_t^{\ell-1}}\right] \Psi_h^{\ell-1} + \mathcal{O}(\Delta t^4)
\end{aligned} \tag{4.34}$$

We expand the exponential operator to second order, making our operator correct to third order. To easily see the result, we operate on an arbitrary function Ψ, viz.,

$$\begin{aligned}
e^{\Delta t \partial_t^{\ell-1}} \Psi &= \Psi + \Delta t \left.\frac{\partial \Psi}{\partial t}\right|_{t_{\ell-1}} + \frac{\Delta t^2}{2} \left.\frac{\partial \Psi^2}{\partial t^2}\right|_{t_{\ell-1}} + \mathcal{O}(\Delta t^3) \\
&= \Psi - i\Delta t H^{\ell-1}\Psi + \frac{\Delta t^2}{2} \left.\frac{\partial}{\partial t}\right|_{t_{\ell-1}} (-iH\Psi) + \mathcal{O}(\Delta t^3) \\
&= \Psi - i\Delta t H^{\ell-1}\Psi + \frac{\Delta t^2}{2} \left(-i\frac{\partial H^{\ell-1}}{\partial t}\Psi - (H^{\ell-1})^2\Psi\right) + \mathcal{O}(\Delta t^3) \\
&= \left[\mathbf{1} - i\Delta t H^{\ell-1} - \frac{\Delta t^2}{2}\left(i\frac{\partial H^{\ell-1}}{\partial t} + (H^{\ell-1})^2\right) + \mathcal{O}(\Delta t^3)\right] \Psi.
\end{aligned} \tag{4.35}$$

With the definition

$$A = i\frac{\partial H^{\ell-1}}{\partial t} + (H^{\ell-1})^2,$$

our discretized time evolution operation becomes

$$\Psi_h^{\ell+1} = \underbrace{\left[\mathbf{1} - 2i\Delta t H^\ell - 2\Delta t^2 H^\ell H^{\ell-1} + i\Delta t^3 H^\ell A\right]}_{U_\Delta^\ell} \Psi_h^{\ell-1} + \mathcal{O}(\Delta t^4). \tag{4.36}$$

In order to find the norm of $\Psi_h^{\ell+1}$ in terms of $\Psi_h^{\ell-1}$ we must calculate the product $(U_\Delta^\ell)^\dagger U_\Delta^\ell$, where the operator U_Δ^ℓ is defined in Eqn. (4.36). It is clear that the order of $\mathbf{1} - (U_\Delta^\ell)\dagger U_\Delta^\ell$ is also the order of the unitarity of the leap-frog scheme.

The first order terms cancel immediately, viz.,

$$\begin{aligned}
(U_\Delta^\ell)^\dagger U_\Delta^\ell = \mathbf{1} &- i\Delta t^3 A^\dagger H^\ell + 4\Delta t^2 (H^\ell)^2 + 4i\Delta t^3 H^{\ell-1}(H^\ell)^2 \\
&- 2\Delta t^2 H^\ell H^{\ell-1} - 4i\Delta t^3 (H^\ell)^2 H^{\ell-1} + i\Delta t^3 H^\ell A + \mathcal{O}(\Delta t)^4
\end{aligned}$$

Introducing a commutator relation yields

$$\begin{aligned}
(U_\Delta^\ell)^\dagger U_\Delta^\ell = &\mathbf{1} - 2\Delta t^2 (H^{\ell-1}H^\ell + H^\ell H^{\ell-1}) + 4\Delta t^2 (H^\ell)^2 \\
&+ i\Delta t^3 (H^\ell A - A^\dagger H^\ell) + 4i\Delta t^3 [H^{\ell-1}, (H^\ell)^2] + \mathcal{O}(\Delta t^4).
\end{aligned}$$

The second order terms should "almost" cancel because $H^{\ell-1}$ is equal to H^ℓ to first order, viz.,

$$H^{\ell-1} = H^\ell - \Delta t \frac{\partial H^\ell}{\partial t} + \mathcal{O}(\Delta t^2).$$

Introducing this relation transforms the second order terms into third order terms, and this

$$2\Delta t^2 (H^{\ell-1}H^\ell + H^\ell H^{\ell-1}) = 4\Delta t^2 (H^\ell)^2 - 2\Delta t^3 \left(\frac{\partial H^\ell}{\partial t} H^\ell + H^\ell \frac{\partial H^\ell}{\partial t} \right).$$

The other third order term may be simplified, viz.,

$$H^\ell A - A^\dagger H^\ell = i \left(H^\ell \frac{\partial H^{\ell-1}}{\partial t} + \frac{\partial H^{\ell-1}}{\partial t} H^\ell \right) + [H^\ell, (H^{\ell-1})^2]$$

The commutator $[H^\ell, (H^{\ell-1})^2]$ is of first order in Δt. To see this we Taylor expand $H^{\ell-1}$ and use

$$(H^{\ell-1})^2 = (H^\ell)^2 + \mathcal{O}(\Delta t),$$

and thereby,

$$[H^\ell, (H^{\ell-1})^2] = [H^\ell, (H^\ell)^2] + \mathcal{O}(\Delta t) = \mathcal{O}(\Delta t).$$

Thus, we may ignore the commutator altogether as the third order term becomes a fourth order term. By the same reasoning we also ignore the other commutator $[H^{\ell-1}, (H^\ell)^2]$. Furthermore,

$$\frac{\partial H^{\ell-1}}{\partial t} = \frac{\partial H^\ell}{\partial t} + \mathcal{O}(\Delta t).$$

Combining these considerations yields

$$(U_\Delta^\ell)^\dagger U_\Delta^\ell = \mathbf{1} + \Delta t^3 \left(H^\ell \frac{\partial H^\ell}{\partial t} + \frac{\partial H^\ell}{\partial t} H^\ell \right) + \mathcal{O}(\Delta t^4).$$

Hence we have secured unitarity to third order. For time independent problems, we see that we have at least fourth order unitarity.

Simplified Analysis in One Dimension. We will use a variant of the von Neumann stability analysis, see Ref. [34]. Let us assume that the Hamiltonian is independent of time. Assume an initial condition on the form of an eigenfunction for the discrete Hamiltonian, viz.,

$$u_j^0 = \sin(k\pi x_j).$$

The eigenvalues for the Hamiltonian of a particle-in-box (in the unit interval) are

$$\epsilon_k^{\mathrm{FDM}} = \frac{4}{h^2} \sin^2(k\pi h/2),$$

and

$$\epsilon_k^{\mathrm{FEM}} = \frac{12}{h^2} \frac{\sin^2(k\pi h/2)}{2 + \cos(k\pi h)},$$

for the finite difference method and finite element method with linear elements, respectively. See section 6.4. The number h is the mesh width.

The leap-frog scheme reads

$$u^{\ell+1} = u^{\ell-1} - 2i\Delta t H u^\ell.$$

We postulate that the wave function at time t_ℓ is given by

$$u^\ell = \xi^\ell \sin(k\pi x),$$

where ξ is a complex number. It is clear that the scheme is then stable if and only if $|\xi| \leq 1$ and unitary if and only if $|\xi| \equiv 1$. Inserting u_j^ℓ into the time stepping scheme yields

$$\xi u_j^0 = \xi^{-1} u_j^0 - 2i\Delta t \epsilon_k u_j^0.$$

As the eigenvectors u_j^0 for the Hamiltonian form a basis for the set of discrete functions this means that

$$\xi^2 + 2i\Delta t \epsilon_k \xi - 1 = 0.$$

This yields

$$\xi = -i\Delta t \epsilon_k \pm \sqrt{1 - (\Delta t \epsilon_k)^2}.$$

Assume that the radicand is positive, i.e., that

$$1 - \Delta t^2 \epsilon_k^2 \geq 0, \quad \forall k.$$

Then we have

$$|\xi|^2 = \Delta t^2 \epsilon_k^2 + 1 - \Delta t^2 \epsilon_k^2 = 1,$$

and the scheme is unitary. This imposes the constraint

$$\Delta t \leq \frac{1}{\max_k \{\epsilon_k\}}$$

on the time step. On the other hand, assume that there exists a k such that the radicand is negative, i.e., that ξ is purely imaginary, viz.,

$$\xi = -i\Delta t \epsilon_k \pm i\sqrt{1 - \Delta t^2 \epsilon_k^2}.$$

Unless $\Delta t^2 \epsilon_k^2 = 0$ there is no way to make $|\xi| = 1$. But this is impossible since both the square energies ϵ_k^2 and the time step must be positive quantities. Hence, the leap-frog scheme is stable if and only if the time step is restricted by the inverse of the largest eigenvalue, i.e.,

$$\Delta t \leq \frac{1}{\epsilon^{\max}}.$$

For the finite difference method we obtain for the particle-in-box

$$\Delta t \leq \frac{h^2}{4\sin^2(N\pi h/2)} = \frac{h^2}{4}.$$

For the finite element method we obtain another estimate, viz.,

$$\Delta t \leq \frac{h^2}{12} \frac{2 + \cos(N\pi h)}{\sin^2(N\pi h/2)} = \frac{h^2}{12}.$$

If we have an additional potential, i.e., that $H = H_{\text{pib}} + V(x)$, and if we assume that $V(x)$ varies slowly in space, the stability criterion should not be severely modified. If on the other hand $V(x)$ varies rapidly, we must lower Δt.

We see that the finite element method actually requires a *smaller* step size than the finite difference method. If we use a two-dimensional uniform grid, the largest eigenvalue has twice the magnitude as in the one-dimensional grid. Hence, Δt must be halved in the two-dimensional case.

Actually, when looking at Fig. 6.2 we see that the fact that the finite element method produces qualitatively *better* eigenvalues and that the eigenvalues are *overestimated* leads to a more strict stability criterion.

Notice that the number of time steps required is not linear in the number of grid points $N = 1/h$, but quadratic. For the finite element method we must solve a linear system at each time step, each requiring perhaps $\mathcal{O}(N^2)$ operations. This totals $\mathcal{O}(N^4)$ operations for the complete simulation! For the theta-rule the time step may be chosen more arbitrarily. On the other hand, it is clear that to improve the quality of the numerical solution we must also lower Δt if we lower h in the theta-rule as well, so it is not clear at this point what method is best.

4.5.5 Properties of the ODE Arising From Space Discretizations

We have seen that in general we have represented the wave function Ψ with a finite-dimensional vector which we call $y \in \mathbb{C}^N$. Accordingly, the spatial part of the PDE is represented by N algebraic equations in N unknowns. Spatially linear PDEs become linear systems, i.e., the spatial operators become $N \times N$ matrices.

For the time dependent Schrödinger equation, the spatial operator is just the Hamiltonian; an Hermitian operator. When solving the resulting ODE it would be fortunate if this Hermiticity is preserved, i.e., that the discretized operator is still Hermitian. This implies that some qualitative properties such as unitarity also is inherited by the ODE.

Let us be more specific. The time dependent Schrödinger equation (1.5) has been converted into a linear ODE, viz.,

$$i\dot{y} = H(t)y, \quad y \in \mathbb{C}^N, \tag{4.37}$$

where N is the total number of degrees of freedom in our discretization. Let $\|y\|$ be the Euclidean norm on $\mathbb{C}^N$. Differentiation yields

$$\frac{\mathrm{d}}{\mathrm{d}t}\|y\|^2 = \dot{y}^\dagger y + y^\dagger \dot{y} = iy^\dagger H^\dagger y - iy^\dagger H y = iy^\dagger (H^\dagger - H)y,$$

i.e., the ODE conserves the Euclidean norm of the solution if and only if $H^\dagger = H$.

Can we be sure of that all the numerical methods described in this chapter yield Hermitian H? Let us for simplicity consider the case where the PDEs for the different spin orientations $\Psi^{(\sigma)}$ are decoupled. The coupled case yields the same results but with a little bit more notation. Let us also consider only the operators $i\partial/\partial x_k$ and x_k, that is the momentum component k and the kth position operator. If the discretizations yield Hermitian matrices P_k and X_k for these, then any Hermitian combination $A(i\partial/\partial x_k, x_k)$ also yields Hermitian discretizations if this is defined by $A(P_k, X_k)$.

The spectral method is the simplest to analyze. For simplicity we confine the discussion to one dimension. Of course x is Hermitian, since it is just multiplication by a diagonal matrix with real elements. The momentum operator $i\partial/\partial x$ is given by

$$P = i\frac{1}{N}Z^\dagger iKZ = -\frac{1}{N}Z^\dagger KZ,$$

where K is the diagonal matrix of real wave numbers. Then P is clearly Hermitian, and so is any power of P.

For finite difference discretizations we must be a little more careful, since we are free to choose any consistent difference for $i\nabla$ and $-\nabla^2$ as our approximation. The standard second order differences are however easily seen to be Hermitian. The matrix for δ_{2x} is skew-symmetric; upon multiplication with i it becomes Hermitian. The matrix for $\delta_x\delta_x$ is real and symmetric and hence Hermitian.

The finite element method involves the stiffness matrix K representing $-\nabla^2$. This matrix is by definition Hermitian, see Eqn. (4.14). For the momentum operator $i\nabla$, we have by integration by parts the matrix elements

$$P_{ij} := i\int_\Omega N_i(\nabla N_j) = -i\int_\Omega (\nabla N_i)N_j + \text{boundary terms} = P_{ji}^*,$$

because the boundary terms vanish due to homogenous Dirichlet boundary conditions. Hence, P is Hermitian.

For an operator A that is a function of $\boldsymbol{x}$ we have

$$A_{ij} := \int_\Omega N_i A(\boldsymbol{x}) N_j$$

which clearly is Hermitian.

4.5.6 Equivalence With Hamilton's Equations of Motion

We may develop a useful analogy with Hamilton's equations of motion of the discretized time dependent Schrödinger equation (4.37). This analogy will also hold for quantum systems of finite dimension such as pure spin systems. The analogy may be very useful when solving the Schrödinger equation, because many good integrators for classical Hamiltonian systems have been discovered.

The vector $y \in \mathbb{C}^N$ is complex. If we by q and p denote the real and imaginary parts, respectively, we have

$$\frac{\mathrm{d}}{\mathrm{d}t}(q+ip) = -iH(t)(q+ip) = -iH(t)q + H(t)p.$$

If we assume that the action of the Hamiltonian on a real vector is again a real vector, we have

$$\frac{\mathrm{d}q}{\mathrm{d}t} = H(t)p, \quad \text{and} \quad \frac{\mathrm{d}p}{\mathrm{d}t} = -H(t)q,$$

where q and p are vectors in $\mathbb{R}^N$ for all t. If we define the function $\mathcal{H}$ as

$$\mathcal{H} = \frac{1}{2}q^T H(t)q + \frac{1}{2}p^T H(t)p,$$

then it is easy to see that q and p satisfy Hamilton's equations of motion with $\mathcal{H}$ as the Hamiltonian, viz.,

$$\dot{q}_i = \frac{\partial \mathcal{H}}{\partial p_i}, \quad \text{and} \quad \dot{p}_i = -\frac{\partial \mathcal{H}}{\partial q_i}, \quad i = 1, 2, \ldots, N.$$

Solving the Schrödinger equation with $y(0)$ as initial condition is then equivalent to solving Hamilton's equations of motion with $p(0) = \text{Re } y(0)$ and $q(0) = \text{Im } y(0)$ as initial conditions.

The unitarity of Eqn. (4.37) is equivalent to

$$\|y\|^2 = q^T q + p^T p = \text{constant}.$$

This is not a general feature of Hamiltonian mechanics. But Hamilton's equations are known to be *symplectic*, that is area preserving. If we take a ball $B \subset \mathbb{R}^{2N}$ of initial conditions, then the sum of the areas of the projections of B onto the $q_i p_i$ planes is conserved, i.e.,

$$A = \sum_{i=1}^{N} \text{Area } P_i(B) = \text{constant},$$

where P_i is the projection onto the $q_i p_i$-plane.

Furthermore, the volume of B is conserved. This is referred to as Liouville's theorem, viz.,

$$V = \iint_B \ldots \int \mathrm{d}q_1 \mathrm{d}p_1 \ldots \mathrm{d}q_N \mathrm{d}p_N = \text{constant}.$$

This property defines a *conservative* ODE.

There are many useful theorems and interesting views on classical Hamiltonian mechanics. See Refs. [6, 23, 38] for a full treatment.

4.6 Basic Stability Analysis

This short section is an introduction to the methods and concepts used when analyzing the numerical properties of time dependent and stationary finite element discretizations. The mathematical theory of finite element methods introduces advanced

function spaces such as Sobolev spaces and also concepts such as weak derivatives. This is beyond the scope of this thesis, but we nevertheless aim at giving a sketch of how the analysis is performed. We focus on the basic ideas, leaving out many details and calculations.

4.6.1 Stationary Problems

A PDE is an equation in which the unknown is a function. Mathematical analysis defines different *function spaces*. These are linear spaces of functions, and the Hilbert spaces such as L_2 of quantum mechanics are examples of such.[12] The spaces L_p is defined as the p-integrable functions (where $p > 0$ is an integer), i.e., the set of functions $u : \Omega \subset \mathbb{R}^d \to \mathbb{R}$ for which the norm is finite, viz.,

$$\|u\| := \left(\int_\Omega |u|^p \right)^{1/p} < \infty.$$

The integral in question is the *Lebesgue integral*. These spaces are Banach spaces, i.e., complete normed spaces. The space L_2 is a special case, and this space is also a Hilbert space with respect to the inner product

$$(u, v)_{L_2} := \int_\Omega uv,$$

and hence the norm is $\|u\| := \sqrt{(u, u)}$. The Sobolev spaces W_p^k are generalizations of L_p in which we assume that the partial derivatives of u up to order k are elements of L_p. The norm is defined by

$$\|u\|_{W_p^k} := \sum_{|\alpha| \leq k} \|D^\alpha u\|_{L_p},$$

where we have used the multi-index notation from section 4.2.1. See also See Refs. [16, 17] for details on Sobolev spaces and other Hilbert spaces, that are the function spaces most common in finite element analysis.[13] The Sobolev spaces $W_2^k =: H^k$ also become Hilbert spaces with the inner product

$$(u, v)_{H^k} := \sum_{|\alpha| \leq k} (D^\alpha u, D^\alpha v)_{L_2}.$$

The most common class of Hilbert spaces used in finite element analysis are the Sobolev spaces H^k, of which $L_2 = H^0$ is an example.

Whatever space we are working with, let us call it V. A stationary PDE is then an equation on the form

$$Au = f, \quad u \in V, v \in W.$$

Here, A is assumed to be a linear operator from V to W. The space W may be larger than V. Introducing a discretization, i.e., a subspace $V_h \subset V$, gives a discrete formulation of the problem, viz.,

$$A_h u_h = f, \quad u_h \in V_h,$$

where A_h is a linear operator on V_h into W. The subscript h indicates a discretization of some kind with parameter h. For finite difference methods it may be is the largest grid spacing and for finite element methods it may be the diameter of the largest

[12] To comply with the notation of Sobolev spaces we will use L_2 instead of L^2 in this section.

[13] The derivatives appearing in the definition of Sobolev spaces are actually so-called weak derivatives, i.e., derivatives of L^2-functions that do not necessarily have a classical derivative. The weak derivative coincides with the classical if the latter exists.

element. If $h \to 0$ then $V_h \to V$ (in some sense) and we expect that $u_h \to u$. (Unless otherwise stated, $u \to v$ means $\|u - v\| \to 0$.) We assume for simplicity that A_h is defined on all of V and that it has an inverse. Let us estimate the error of the discrete solution u_h:

$$\|u - u_h\|_V = \|A_h^{-1}A_h(u - u_h)\|_V = \|A_h^{-1}(A_h u - Au)\|_V \leq \|A_h^{-1}\|_{\mathcal{L}(W,V)} \cdot \|A_h u - Au\|_W.$$

Hence, for the discrete solution u_h to approach u it is enough to require a uniform boundedness on A_h^{-1}, i.e., that it exists a constant $C > 0$ such that $\|A_h^{-1}\| \leq C$ for all h. This property is called *stability*. Furthermore, we must require that

$$(A_h - A)u = A_h u - f \to 0$$

as $h \to 0$. This property is *consistency*. It measures the error in the equation, as we easily see.

This formulation is the one usually used in the analysis of finite difference methods, and we easily recognize stability and consistency as they were introduced earlier. Both properties are easy to deduce in this case. For finite element methods, however, the usual approach is different, as the truncation error $A_h u - f$ is not easy to estimate.

In finite element methods, our discrete problem is a discretization of a *weak formulation* of the PDE. For simplicity, we will confine the discussion to real PDEs.

The above formulation of the PDE was: Find u such that $Au = f$. This pointwise fulfillment (u is then called a *classical solution*) of the PDE is replaced with a fulfillment on average, i.e., find u such that

$$a(u, v) = (f, v), \quad \forall v \in H.$$

Here, $a(\cdot, \cdot) : H \times H \to \mathbb{R}$ is a bilinear form on H, i.e., linear in both arguments. We have replaced V with another space $H \supset V$, as the weak formulation has lesser constraints on the solution than the classical, pointwise PDE. To see this, consider the PDE

$$-\nabla^2 u = f,$$

i.e., Laplace' equation. Note that we must require u to be at least twice continuously differentiable if f is continuous. Multiplying with a test function v and integrating yields

$$-\int_\Omega v\nabla^2 u = \int_\Omega \nabla v \cdot \nabla u - \int_{\partial\Omega} v\frac{\partial u}{\partial n} = \int_\Omega fv.$$

Assuming that the boundary terms vanish as in the case of homogenous Neumann boundary conditions, we may identify the bilinear form

$$a(u, v) = \int_\Omega \nabla u \cdot \nabla v.$$

Notice that we only need u and v to be differentiable. Hence, a solution of $a(u, v) = (f, v)$ is weaker than the classical solution. Before integrating by parts, we had to require $u \in H_2^2$. Integrating by parts then reduced the constraints, i.e., $u \in H_2^1$.

We must assume some fundamental properties of bilinear forms to carry out our discussion. First, we need[14] $a(\cdot, \cdot)$ to be *symmetric*, i.e., $a(u, v) = a(v, u)$. Second, we need *boundedness*, i.e., there exists a constant $C_1 \geq 0$ such that

$$a(u, v) \leq C_1\|u\|\|v\|, \quad \forall u, v \in H.$$

Boundedness of the bilinear form is easily seen to be equivalent with continuity. Third, we need *coercivity*, i.e., there exists a constant $C_2 > 0$ such that

$$a(u, u) \geq C_2\|u\|^2, \quad \forall v \in H.$$

[14] Not really, but it makes life much simpler.

The bilinear form in the example is easily seen to be symmetric, bounded and coercive. If $a(\cdot,\cdot)$ is symmetric and coercive the weak PDE is equivalent to the minimization of the functional

$$J(u) = \frac{1}{2}a(u,u) - (f,u),$$

i.e., to find u such that $J(u) \leq J(v)$ for all $v \in H$; see Ref. [34]. In our example,

$$J(u) = \frac{1}{2}\int_\Omega |\nabla u|^2 - \int_\Omega fu.$$

For this reason, the weak formulation is also called *the variational formulation*. When formulating a discrete problem one chooses $V_h \subset H$ as for the classical PDE and seek u_h such that

$$a(u_h, v) = (f, v), \quad \forall v \in V_h.$$

This is exactly Galerkin's method in the weighted residual method. Let us analyze the error $u - u_h$ of the weak formulation.

Note that by coercivity, we have

$$C_2\|u_h\|^2 \leq a(u_h, u_h) = (f, u_h) \leq \|f\|_*\|u_h\|,$$

where $\|f\|_*$ is the *dual norm* of f, i.e., the norm of f as a linear functional on H. Hence,

$$\|u_h\| \leq \frac{1}{C_2}\|f\|_*,$$

and u_h is bounded automatically by f. This is the stability criterion, and we see that in the weak formulation we get this for free, so to speak, as long as coercivity of $a(\cdot,\cdot)$ holds.

Let us turn to consistency, i.e., if the discrete equation converges to the exact equation as $h \to 0$. Let $\delta_h(u)$ be the distance from u to V_h, i.e.,

$$\delta_h(u) := \inf_{v \in V_h} \|u - v\|.$$

This really defines that $V_h \to V$. If u is the solution to the exact problem, then $\delta_h(u)$ must vanish as $h \to 0$ in order to make the discrete formulation consistent. We see that consistence depends on the choice of V_h. In finite element methods, it turns out that the error $\|u - u_h\|$ is proportional to $\delta_h(u)$. Hence, both stability and consistence is fulfilled for every finite element method. See for example Refs. [16, 17].

Finally we show the *best approximation in norm property* of Galerkin's method. First note that for any $v \in V_h$,

$$a(u - u_h, v) = a(u, v) - a(u_h, v) = (f, v) - (f, v) = 0.$$

Hence, for any $v \in V_h$,

$$C_2\|u - u_h\|^2 \leq a(u - u_h, u - u_h) = a(u - u_h, u - v) \leq C_1\|u - u_h\| \cdot \|u - v\|.$$

Hence,

$$\|u - u_h\| \leq \frac{C_1}{C_2}\|u - v\|, \quad \forall v \in V_h. \tag{4.38}$$

We see that the discrete formulation actually finds $u_h \in V_h$ that is closest to u. If $\delta_h(u) \to 0$ as $h \to 0$ then clearly $u_h \to u$. See also Ref. [34].

Let us introduce a theoretical tool which will become useful when we study the stability of time dependent problems. We define an operator $R_h \in \mathcal{L}(V, V_h)$. This

operator projects a function u onto the discrete space V_h with respect to the energy inner product $a(\cdot,\cdot)$, i.e.,

$$a(R_h u, v) = a(u, v)m, \quad \forall v \in V_h.$$

Eqn. (4.38) then becomes

$$\|(\mathbf{1} - R_h)u\|_{H^1} \leq \frac{C_1}{C_2} \inf_{v \in V_h} \|u - v\|_{H^1}.$$

4.6.2 Time Dependent Problems

Although we argued in the introduction to this chapter that discretization in time is independent from discretization in space this is not entirely true as far as the numerical convergence concerned. For example, when $h \to 0$ in space, the resulting ODE becomes larger and larger. Hence, the convergence in time is affected by the mesh width in space. We need some kind of uniformity to circumvent this.

Let us perform an analysis similar to the one above for a time dependent problem. Will consider a PDE on the form

$$u_t + Au = f, \quad u \in H^2,$$

where we seek $u : [0, T] \longrightarrow H^1$, where $T \leq \infty$. We are given an initial condition $u(0) = g \in H^1$. A mathematical question we will not investigate here is whether $u(t)$ for a given t exists. We will assume that it does.

The weak formulation reads

$$(u_t, v) + a(u, v) = (f, v), \quad \forall v \in H^1 \tag{4.39}$$

where we assume $a(\cdot,\cdot)$ to be a symmetric, bounded and coercive bilinear form on H^1. In particular, the weak formulation must hold for $v = u$, i.e.,

$$(u_t, u) + a(u, u) = (f, u).$$

We have

$$(u_t, u) = \int_\Omega u_t u = \int_\Omega \frac{1}{2}\frac{\partial}{\partial t}(u^2) = \frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}t}\int_\Omega u^2 = \frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}t}\|u\|_{L^2}^2,$$

where we assume that integration and differentiation with respect to time is interchangeable. This gives

$$\frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}t}\|u\|_{L^2}^2 \leq (f, v), \tag{4.40}$$

as $a(u, u) \geq 0$ by coercivity. We note that by the triangle inequality,

$$|(f, v)| \leq \|f\|_{L^2}\|u\|_{L^2} \leq \frac{1}{2}\left(\|f\|_{L^2}^2 + \|u\|_{L^2}^2\right).$$

Integrating Eqn. (4.40) from $t = 0$ to $t = T$ and using $u(0) = g$ gives

$$\frac{1}{2}\|u(T)\|_{L^2}^2 \leq \frac{1}{2}\|g\|_{L^2} + \frac{1}{2}\int_0^T \|f\|_{L^2}\,\mathrm{d}t + \frac{1}{2}\int_0^T \|u(t)\|_{L^2}\,\mathrm{d}t.$$

This is a stability criterion. To see this, consider Grönwall's inequality, see Refs. [larsson2003,mcowen2003]. If $x(t) \geq 0$ obeys

$$x(T) \leq h(T) + b\int_0^T x(t)\,\mathrm{d}t,$$

with $h(t) \geq 0$ and $0 \leq T \leq t$, then

$$x(T) \leq e^T x(0) + \int_0^T e^{T-t} f(t) \, \mathrm{d}t.$$

Turning to a discrete version of the weak formulation, we see that like in the stationary case we need an approximation property similar to Eqn. (4.38). Consistency is built into the formulation in the case of finite element methods. The discrete version of Eqn. (4.39) reads

$$((u_h)_t, v) + a(u_h, v) = (f, v), \quad \forall v \in V_h,$$

with $u_h(0) = g_h$ as initial condition. From this we immediately obtain

$$\|u_h(t)\|_{L^2} \leq \|g\|_{L^2},$$

i.e., stability is built into the formulation.

We wish to say something about the time development of the error $u - u_h$. We introduce an operator $R_h : H^1 \to H^1$ that takes an element $u \in H^1$ to its projection on V_h using $a(\cdot, \cdot)$ as inner product.[15] I.e., $R_h u$ is defined by

$$a(R_h u, v) = a(u, v), \quad \forall v \in V_h.$$

Hence, $R_h u \in V_h$. This operator is only a theoretical tool; we will never actually compute it (or it's action on u).

The time development of the error is given by

$$((u - u_h)_t, v) + a(u - u_h, v) = 0, \quad \forall v \in V_h.$$

Writing $u - u_h = u - R_h u + R_h u - u_h$ where $R_h u - u_h \in V_h$ yields

$$((R_h u - u_h)_t, v) + a(R_h u - u_h, v) = ((R_h u - u)_t, v), \quad \forall v \in V_h,$$

where the right hand side may be interpreted as a source term. We have used that by definition $a(u - R_h u, v) = 0$ for all $v \in V_h$. We see that the time development of $u - u_h$ is rewritten in terms of functions in V_h. Letting $v = R_h u - u_h$ yields

$$\frac{1}{2} \frac{\mathrm{d}}{\mathrm{d}t} \|R_h u - u_h\|_{L^2}^2 \leq (R_h u_t - u_t, R_h u - u_h),$$

in the same way as when we analyzed stability of $u \in H^1$. Hence, if we define $e = R_h u - u_h \in V_h$ we have

$$\|e(T)\|_{L^2}^2 \leq \int_0^T \|R_h u_t - u_t\|_{L^2}^2 \, \mathrm{d}t + \int_0^T \|e(t)\|_{L^2}^2 \, \mathrm{d}t,$$

where we have used $u_h(0) = R_h u(0)$ by definition. By Grönwall's inequality we have

$$\begin{aligned} \|e(T)\|_{L_2}^2 &\leq e^T \int_0^T \|(\mathbf{1} - R_h) u_t\|_{L_2}^2 \, \mathrm{d}t + \|e(0)\|_{L_2}^2 \\ &\leq T e^T \max_{0 \leq t \leq T} \|(\mathbf{1} - R_h) u_t\|_{L_2}^2 + \|e(0)\|_{L_2}^2, \end{aligned}$$

where $e(0)$ is the initial error. The initial error is not zero because the discretized initial condition is not necessarily equal to the continuous initial condition. On the other hand, we see that if δ_h becomes smaller then so must $\|e(0)\|_{L_2}$. In that case the stability criterion becomes stronger. The function $(\mathbf{1} - R_h)w$ is the orthogonal complement to $R_h w$ whose norm also must become smaller when δ_h decreases. We see that decreasing the mesh width improves the stability criterion.

[15] An inner product is a symmetric, positive definite bilinear form.

Chapter 5

Numerical Methods for Linear Algebra

This chapter is a rather brief overview of the various methods that are in use for solving (square) systems of linear equations and (standard and generalized) eigenvalue problems. Linear algebra is perhaps the most fundamental tool in numerical computation, and hence it is important to master the powerful tools that have been developed.

The theoretical aspects of linear algebra may be found in Ref. [39]. The numerical methods are described well in Ref. [40] and Ref. [41].

In this section we are primarily concerned with square matrices. We shall consider two kinds of equations; square systems of linear equations and eigenvalue problems.

Let $A \in \mathbb{C}^{N\times N}$ be any square matrix of dimension n and let $b \in \mathbb{C}^N$ be any n-dimensional vector. We then seek $x \in \mathbb{C}^N$ such that

$$Ax = b. \tag{5.1}$$

This equation has a unique solution if and only if A has an inverse A^{-1} and if and only if determinant does not vanish, i.e., $\det A \neq 0$. The matrix is then called *non-singular*. If $\det A = 0$, i.e., if A is singular, the set of solutions to the homogenous equation $Ax = 0$ is a subspace of $\mathbb{C}^N$. A solution to Eqn. (5.1) may happen not to exist in the non-singular case, but if it does then it is in general given by $x + x_p$, where x_p is any particular (i.e., an arbitrary) solution of Eqn. (5.1). We will assume that A is non-singular in the following discussion.

The need for solving Eqn. (5.1) arise in a wide range of problems in computational physics. For our part, we need to solve large systems when using the finite element method and also when we are given implicit time integration schemes. In addition, solving eigenvalue problems also implies solving linear systems, as will be described later. In fact, most of the time spent in solving PDEs is invested in linear algebra problems, and so it is of major importance to choose fast and reliable methods.

The generalized eigenvalue problem reads: Given a pair of square matrices A and B, find the *eigenpairs* (λ, x), where λ is a scalar and x is a vector such that

$$Ax = \lambda Bx.. \tag{5.2}$$

The scalar λ is called an eigenvalue and x is called an eigenvector. In the case $B = I$, i.e., the identity matrix, we obtain the standard eigenvalue problem, viz.,

$$Ax = \lambda x.$$

We shall only concern ourselves with the Hermitian eigenvalue problems, i.e., both A and B are Hermitian matrices. In this case one may always find N eigenpairs with real eigenvalues and orthonormal eigenvectors, i.e., a basis for $\mathbb{C}^N$.

5.1 Introduction to Diffpack

Before we discuss the numerical methods we give a brief introduction to Diffpack, the C++ library used in the finite element implementations in this thesis. It is natural to introduce Diffpack at this point, after having introduced the finite element discretizations. A basic knowledge of Diffpack is also useful to get an overview of how the numerical methods are used in the program. We cannot even scratch the surface of Diffpack in this thesis. We will however mention some of the key points in the structure, class hierarchy and implementations of finite element solvers in order to understand pros and cons, limitations and possibilities.

Diffpack is a very complex library. In addition to defining hundreds of classes and functions it also includes many of auxiliary utilities, performing a wide range of tasks such as generation of geometrically complicated grids, visualization of data and conversion of data to and from various common formats. The programming library and the utilities together constitute a complete environment for solving PDE related problems; including planning and preparation, implementation, simulation and data organizing, visualization and presentation of results.

The Diffpack project was started in the mid 90's by Sintef and the University of Oslo. As the project grew in size, complexity and reputation the firm Numerical Objects was founded in 1997. Diffpack is now owned by the German company *inuTech* and jointly developed with Simula Research Laboratory. See Refs. [42–44] for details.

Diffpack is a library developed with extensive use of object-oriented techniques, as is a must when developing flexible finite element solvers. Many classes are templated[1] and a hierarchy of smart pointers (so-called *handles*) are implemented, easing the memory handling which otherwise have a tendency to require many debugging session in C++. There are class hierarchies for matrices, grids and grid generators, finite element abstractions, solvers for systems of linear equations and so on.

Ref. [34] is an excellent introduction to both the finite element method and *Diffpack* programming. It is an easy-to-read and informative account and a good introduction and reference for practitioners at all levels. In addition, the online Diffpack documentation in Ref. [45] is recommended.

5.1.1 Finite Elements in Diffpack

Diffpack implements many different finite element types, including (but not limited to) linear and quadratic elements in one dimension, and tensor product generalizations to more dimensions. The subclasses of `ElmTensorProd` exemplify this.

For example, the element class `ElmB2n1D` implements a linear element in one dimension, i.e., one-dimensional elements (intervals) with two nodes located at the endpoints, and the class `ElmB4n2D` is the two-dimensional generalization, by taking the tensor product of two one-dimensional elements. Hence, it is defined on quadliterals and has four nodes located at the corners. The (piecewise bilinear) element functions are depicted in Fig. 4.6.

The class `ElmB3n2D` is a one-dimensional quadratic element with three nodes in each element, two located at the endpoints of the interval and the third somewhere in-between. The two-dimensional generalization is `ElmB9n2D` with 9 nodes. In three dimensions we have the class `ElmB27n3D` with 27 nodes distributed in a parallelepiped.

Besides methods that evaluate the basis functions and their partial derivatives, the element classes contain Gaussian integration rules that may be used when assembling the element matrices and vectors. This is done in the `integrands()` method of `FEM`, the class from which our finite element simulators are derived. This method evaluates the integrands in the one-dimensional finite element system example in Eqns. (4.12) and

[1] Actually, templating is emulated with preprocessor macros and directives due to compiler-dependent behavior the template feature in C++.

(4.13). Assembling the element matrices and vectors is done when calling `FEM::makeSystem()`. Numerical integration takes part and `FEM::integrands()` evaluates the integrands at the points defined by the current integration rules. Hence, numerical integration is a very important part of finite element implementations with Diffpack.

A detailed description of Diffpack is out of the scope for this text. The code for the programs written in this thesis use many of Diffpack's features than described here, but the code is well commented in order to compensate for this.

5.1.2 Grid Generation

Diffpack is bundled with lots of utilities and scripts that typically call third-party grid generation software (called preprocessors) in order to produce compatible grids. Examples of such third-party software are Triangle and GeomPack. Diffpack comes with classes with interfaces to such utilities in addition to scripts and utilities. With these, both simple and complicated grids may easily be created. Ref. [46] is a comprehensive introduction to finite element grid preprocessors used in Diffpack. In this thesis, we employ square grids and disk-grids (i.e., approximations to circular regions).

A typical way to generate a grid is by the external utility (written with Diffpack) makegrid. This is a command line based interface to the various grid preprocessors. Here is a sequence of commands that generate a unit box in two dimensions with 9×9 linear elements:

```
set preprocessor method = PreproStdGeom
set output gridfile name = box
ok
set geometry = d=2 [0,1]x[0,1]
set partition = d=2 e=Elmb4n2D div=[10,10] grading=[1,1]
ok
```

For making a disk grid, one may create a file describing the outline of the circular region in terms of a polygon and then use Triangle to triangulate it. Here is a short snippet of Python code to create such a grid:

```
import os
from math import *

# Generate a disk with radius r and mesh width h.
# Uses the "Triangle" program.
def makeDiskTri(r, h):

    div = floor(r/h + 1) # number of segments in outline polygon
    print "generating disk with div=%d, h=%f" % (div, h)

    nel = floor((div-1)*(div-1)/(2*pi)) # desired number of elms
    el_area = pi*r*r/nel                # average element area


    # create the input file to Triangle.

    poly_file_text = "%d 2 0 1\n" % (div)
    for i in range(div):
        angle = i*2*pi/float(div)
        (x, y) = (cos(angle)*r, sin(angle)*r)
        poly_file_text = poly_file_text + "%d %g %g 20010\n" %
                                                    (i+1, x, y)
```

```
poly_file_text = poly_file_text + "%d 1\n" % (div)

for i in range(div):
    if (i==div-1):
        i2 = 1
    else:
        i2 = i+2

    poly_file_text = poly_file_text + "%d %d %d 20010\n" %
                                          (i+1, i+1, i2)


poly_file_text = poly_file_text + "0\n"

f= open(".disk.poly", "w")
f.write(poly_file_text)
f.close()

# Call Triangle. No angles should be less than 28 deg,
# all triangle areas should be less than el_area
# (yielding at least nel elements).

os.system("triangle -q28Ipa%f .disk.poly" % (el_area))
os.system("triangle2dp .disk")

fname = "disk_%d_%f" % (div, h)
os.system("mv .disk.grid %s.grid" % (fname))
```

A grid generated with this script can be seen in Fig. 5.1.

5.1.3 Linear Algebra in Diffpack

It is important to be aware of the different kinds of matrices that Diffpack offers with their advantages and disadvantages. Here we give a brief overview of the matrix types and their corresponding classes. Each matrix type may be complex or real. As the matrix classes in Diffpack are templated this is indicated as a template parameter, e.g., `Matrix(Complex)` or `Matrix(real)`. In general we write `NUMT` for the type. All matrix classes are derived from `Matrix(NUMT)`. See the online documentation in Ref. [45] for details.

The important differences between the matrix types lies in that we assume different structures of the matrices. This allows for optimizations when calculating for example matrix-vector products. There is no need to store a full 1000 by 1000 matrix if we know that only the diagonal is different from zero, and performing a matrix-vector product with nearly a million multiplications with zero is a waste of resources. As a matter of fact, the matrix-vector product is very important to optimize for this thesis, as it is the most fundamental operation when both solving linear systems of equations iteratively and when solving eigenvalue problems.

An important property of the finite element matrices is *sparsity*. The element matrices are sparse, meaning that most of the elements are known to be zero. Using a grid with 10^8 nodes yields a linear system of the same dimension, and storing a full matrix clearly cannot (and should not!) be done. The matrix elements A_{ij} where $\boldsymbol{x}^{[i]}$ and $\boldsymbol{x}^{[j]}$ are indices not belonging to the same element are known to be zero. Hence, the number of non-zeroes in the element matrix is of order N, yielding *very sparse* matrices as the fraction of non-zeroes is actually of order $1/N$. We must mention,

however, that this is a somewhat simplified picture. Higher order elements yields more couplings between the nodes (as more nodes belong to the same element) and hence increase the number of non-zeroes.

Fig. 5.1 shows a typical moderately-sized finite element grid with linear triangular elements and the corresponding sparsity pattern. The figure shows a square picture where each point corresponds to a matrix element. Black dots are non-zeroes and white dots are zeroes.

Dense Matrices. Dense $N \times M$ matrices are the most general matrices. Every entry is assumed to be *a priori* a possible non-zero. Hence, a full array of $N \cdot M$ (complex or real) numbers must be allocated in an implementation.

The ith component of the matrix-vector product $w = Av$ is given by

$$w_i = \sum_{j=1}^{n} A_{ij} v_j,$$

and as w has M components, this amounts to NM multiplications and additions, and hence $\mathcal{O}(NM)$ floating point operations are required for a matrix-vector product in the real case. For complex matrices, the multiplications and additions are of course complex, roughly quadrupling the number of multiplications.

Dense matrices are implemented in the class `Mat(NUMT)`.

Diagonal Matrices. On the other extreme of dense matrices we find the diagonal matrices, in which only the diagonal elements are non-zeroes. The lumped mass matrix is an example of a diagonal matrix. For a square N-dimensional matrix exactly N (real or complex) numbers must be stored. The matrix vector product reduces to

$$w_i = A_{ii} v_i,$$

and hence only $\mathcal{O}(N)$ operations are needed for a matrix-vector product. Again, complex matrices require about four times more work than the real matrices.

Diagonal matrices are supported through the `MatDiag(NUMT)` class.

Tridiagonal Matrices and Banded Matrices. Tridiagonal matrices are allowed to have non-zeroes at the first super- and subdiagonal as well as the main diagonal, i.e., only $A_{i,j}$, $j = i-1, i, i+1$ are allowed to be nonzero. A square N-dimensional tridiagonal matrix is stored as a $3 \times N$ dense matrix. It is easily seen that a matrix-vector product requires $\mathcal{O}(3N)$ operations. (Again, complex matrices require a bit more.)

Tridiagonal matrices are special cases of *banded matrices*. A banded matrix allows non-zeroes at the k first sub- and superdiagonals. The total number $2k+1$ of non-zero diagonals is called the bandwidth. It is stored as a $(2k+1) \times N$ dense matrix and clearly $\mathcal{O}((2k+1)N)$ operations are needed for a matrix-vector product.

General Sparse Matrices. A matrix is called sparse if in some sense the number of (possible) non-zeroes is low. A typical example is diagonal and tridiagonal matrices, in which the number of non-zero elements is of order N. A matrix-vector product then requires only $\mathcal{O}(N)$ operations as is easily seen.

Fig. 5.1 shows the banded structure of an element matrix and the fact that many of the zeroes inside the band is zero, resulting in very many unnecessarily stored elements if we use `MatBand(NUMT)`. Indeed, the fraction of non-zeroes in the band decreases rapidly for larger grids (i.e., larger element matrices), and hence we need support for matrices with an even more general structure.

In general sparse matrices we store only the non-zero elements. The class `MatSparse(NUMT)` implements the so-called *compressed sparse row* storage scheme (CSR

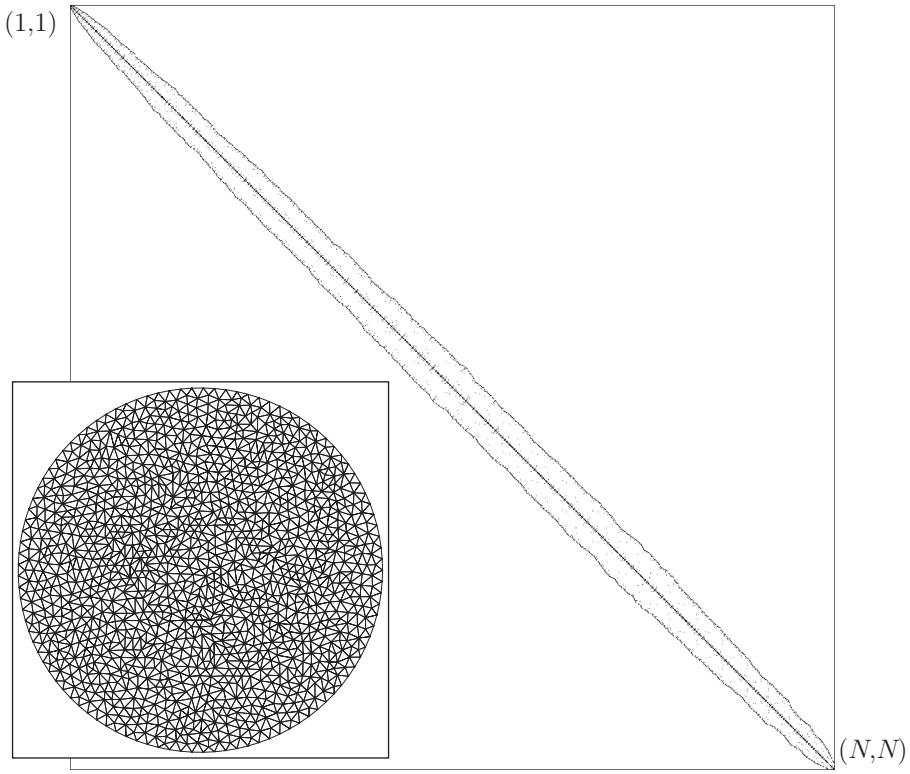

Figure 5.1: Sparsity pattern matrix for a finite element grid (inset). Non-zeroes are shown as black dots. The matrix dimension is 1148.

for short). It is the most compact way of storing the matrix and allows very fast computations of matrix-vector products.

The $N \times M$ sparse matrix is stored by means of three one-dimensional vectors $a \in \mathbb{C}^n$, $c \in \mathbb{N}^n$ and $r \in \mathbb{N}^{N+1}$, where n is the total number of non-zero entries and N is the number of rows. The elements of a are the non-zero matrix entries ordered row-wise from left to right, i.e., in reading order. The entries of c are the column indices for these, i.e., $a(s)$ resides in column $c(s)$. Element number i of r stores the index s of the *first entry* of row i. In addition, $r(N+1) = M+1$ by definition. Hence, the number of non-zero entries in row i is $r(i+1) - r(i)$. For example, consider the matrix

$$A = \begin{pmatrix} 1 & 0 & 2 \\ 0 & -3 & 4 \\ 5 & 0 & 0 \\ 0 & 0 & 6 \end{pmatrix}.$$

Here, $n = 6$ and

$$a = (1, 2, -3, 4, 5, 6)$$

are the entries. The column-indices are

$$c = (1, 3, 2, 3, 1, 3)$$

and the row pointers are

$$r = (1, 3, 5, 6, 7).$$

Solvers for Linear Systems of Equations. As we have seen, large sparse matrices arise from finite element discretizations and finite difference approximations. Hence, solving large systems of linear equations with a sparse coefficient matrix is fundamental. As we shall see, the corresponding eigenvalue problems, i.e., finding the eigenvalues and eigenvectors of the discrete Hamiltonian, also requires solutions to linear systems in order to be solved.

For linear systems Diffpack implements many solvers, from Gaussian elimination to Krylov subspace methods with preconditioning. In the next section we will review the most important methods that are implemented. Again, we refer to Ref. [34] for details. Here we merely mention that a separate hierarchy of classes are implemented in Diffpack (i.e., `LinEqSolver` and its descendants) that allows the user to solve linear systems with all available methods.

Eigenvalue Problems in Diffpack. Unfortunately there is no direct support for solving eigenvalue problems in Diffpack as of yet. To achieve this one must either implement the proper methods or in some way connect Diffpack matrix classes with external solvers. As eigenvalue computation is fairly complicated business we would prefer to use ready-made libraries. In an external report written for Simula Research Laboratory (see Refs. [5, 44]) I have investigated the possibility of using the popular ARPACK eigenvalue package with matrices from Diffpack. A simple implementation was made and the eigenvalue solvers in this thesis are an extension of the implementation found in this work.

5.2 Review of Methods for Linear Systems of Equations

In this section we consider square linear systems of equations, i.e., find x such that

$$Ax = b,$$

where A is an $N \times N$ matrix and x and b are N-dimensional (complex) vectors.

For large systems the time spent on solving the system might be much higher than constructing the system, for example via assembly of the element matrices. It is then vital to have access to a variety of numerical methods for solving linear systems. The optimal choice for solving the linear systems is highly problem dependent, and one should not rely solely on for example Gaussian elimination. It is then important that the programming environment makes is easy to switch between the methods.

There are three basic properties that we need to consider for each method: For what matrices they apply, the computational cost (both number of operations and storage requirements) of the method and how the method actually works. Se Refs. [34, 40] for details on the methods presented here.

5.2.1 Gaussian Elimination and Its Special Cases

Strictly speaking, Gaussian elimination is only one of the methods that usually is referred to by that name. The pure Gaussian elimination is rarely implemented; instead one uses the LU decomposition method, for example.

LU Decomposition. The LU decomposition method assumes that we can rewrite our matrix A as

$$A = LU,$$

where L is lower triangular and U is upper triangular. Solving a lower or upper triangular system is trivial and is done by forward substitution and backsubstitution,

respectively. Let us write

$$LUx = L(Ux) = b.$$

The equation $Ly = b$ is easily solved by *forward substitution*, i.e.,

$$y_1 = \frac{b_1}{\alpha_{11}},$$

and

$$y_i = \frac{1}{\alpha_{ii}} \left[b_i - \sum_{j=1}^{i-1} \alpha_{ij} y_j \right], \quad i = 2, 3, \ldots, N.$$

Here, α_{ij} are the lower triangular elements of L. Then we solve the equation $Ux = y$ with *forward substitution*, i.e.,

$$x_N = \frac{y_N}{\beta_{NN}},$$

and

$$x_i = \frac{1}{\beta_{ii}} \left[y_i - \sum_{j=i+1}^{N} \beta_{ij} x_j \right], \quad i = N-1, N-2, \ldots, 1.$$

Here, β_{ij} are the upper triangular elements of L. As we see, this produces the solution of $Ax = b$ in $\mathcal{O}(N^2)$ operations, given L and U.

The LU decomposition is usually performed with an algorithm called *Crout's algorithm*. (The algorithm also shows that is always is possible to do LU decomposition, by construction.) The algorithm needs $\mathcal{O}(N^3)$ operations for a dense matrix. Clearly, with $N \sim 10^7$ the LU decomposition is not feasible in general, but for some special cases the amount of work needed reduces drastically.

Notice that once the LU decomposition is performed, any right hand side b can be solved with forward and backsubstitution. If we need to solve several right hand sides this should be taken advantage of.

In Diffpack, Gaussian elimination with LU decomposition is implemented in the `GaussElim` class, a subclass of `LinEqSolver`.

Gaussian elimination with LU decomposition is mathematically fool-proof. If A is non-singular the solution x is given by the above formulas. Numerically we however cannot avoid round-off errors. One way to minimize such errors is to perform a process called *pivoting* in the LU decomposition. In fact, Gaussian elimination methods are numerically unstable if pivoting is not carried out.

Round-off errors are still a problem if the matrix A is *ill-conditioned*, meaning that its determinant is almost zero. The *condition number* κ is defined as the square root of the ratio of the magnitudes of the largest and the smallest eigenvalue, i.e.,

$$\kappa = \sqrt{\frac{|\lambda_{\max}|}{|\lambda_{\min}|}}.$$

If the condition number is infinite the determinant is zero as one of the eigenvalues is zero. Hence, the matrix is singular. If κ^{-1} approaches the machine precision, i.e., the smallest representable number in the computer, it is singular as far as the numerical methods concerned. Hence we can get unpredictable results when using such ill-conditioned coefficient matrices.

Tridiagonal Systems and Banded Systems. The situation is greatly simplified if A is a tridiagonal matrix. Recall that tridiagonal matrices have non-zero elements only along the diagonal and the first sub- and superdiagonal. Hence, it has approximately $3N$ nonzero elements. For such systems LU decomposition and forward and backsubstitution takes only $\mathcal{O}(N)$ operations, which is optimal. The algorithm can be coded in

merely a handful of lines, see for example Ref. [40]. The storage requirements are only a vector of length N for temporary storage, contrary to an array of length N^2 needed for the full LU decomposition.

Banded systems are more general, in that the matrix A has non-zero elements along k sub- and superdiagonals. Banded matrices are frequently output from element assembly routines, although general sparse matrices perform better. (One complication is however that Gaussian elimination is not implemented for complex sparse matrices, see the paragraph below on sparse systems.)

The computational cost of banded Gaussian elimination depends on the bandwidth, but if $n \ll N$ it is much faster than full Gaussian elimination.

It is noteworthy that the tridiagonal Gaussian elimination need almost no extra storage space. Nor does Crout's algorithm for dense matrices need extra space if we can live with the original dense matrix being destroyed. Banded Gaussian elimination however creates additional non-zeroes called *fill-ins* at locations outside the band in A. If the bandwidth is comparable to N the number of fill-ins are considerable.

Tridiagonal and banded Gauss elimination is also implemented in `GaussElim`.

Sparse Systems. Sparse matrices contain mostly zeroes. Often the number of non-zeroes is of order N. The structure of a sparse matrix may vary greatly, see for example Fig. 2.7.1 of Ref. [40]. The structure may for example lie somewhere between a dense matrix and a banded matrix.

The tridiagonal LU decomposition with forward and backsubstitution can be performed in only $\mathcal{O}(N)$ operations. This is due to clever application of the algorithm and bookkeeping of the zeroes in the system. The structure of the sparse matrix plays a fundamental role here, and for general sparse matrices one must in some way analyze the sparsity pattern in order to implement an effective algorithm. Keeping the number of fill-ins reasonable and at the same time minimizing the number of operations needed is no simple task.

[Should have more on Diffpack's approach to sparse systems here. Also mention that no Gaussian elimination is implemented for complex sparse systems.]

5.2.2 Classical Iterative Methods

Gaussian elimination is a direct method, i.e., an algorithm for finding x with an exactly known complexity. We can precisely count the number of operations needed for an exact solution to be found. Iterative methods take another approach. As the name suggests one creates a sequence of trial vectors x^k that hopefully converges to the real solution x. Each iteration should be in some sense cheap in order make this meaningful.

Classical iterative methods split A into $A = M - N$ and hence

$$Mx = Nx + b,$$

suggesting an iterative process given by

$$Mx^{k+1} = Nx^k + b.$$

This process is *the same at each iteration*, which is characteristic for classical methods. We see that the system $Mx' = b'$ must be in some sense cheap to solve if the iteration process should have any practical use. If we define the residual r^k as

$$r^k := b - Ax^k,$$

(note the similarity to the finite element formulation) we obtain

$$x^{k+1} = x^k + M^{-1}r^k.$$

Defining the matrix $G = M^{-1}N$ and the vector $c = M^{-1}b$ we obtain

$$x^{k+1} = Gx^k + c, \tag{5.3}$$

and this makes us able to analyze the stability and convergence properties of the iteration. Through the following chain of identities,

$$\begin{aligned} x^{k+1} - x &= Gx^k + c - x = Gx^k + M^{-1}(b - Mx) \\ &= Gx^k + M^{-1}(b - Ax - Nx) = Gx^k - Gx = G(x^k - x), \end{aligned}$$

we obtain

$$x^k - x = G^k(x^0 - x).$$

In other words the error $e^k = x - x^k$ converges to zero if and only if

$$\lim_{k\to\infty} \|G^k\| = 0.$$

If we can diagonalize G, then $\|G\| = (\max_{\lambda\in\sigma(G)} |\lambda|) =: \varrho(G)$, where $\sigma(G)$ is the set of eigenvalues and is called *the spectrum of* G. The quantity $\varrho(G)$ is called the *spectral radius*. Hence,

$$\lim_{k\to\infty} \|e^k\| = \|e^0\| \lim_{k\to\infty} \|G^k\| = \|e^0\| \lim_{k\to\infty} \varrho(G)^k,$$

and the iteration process converges if and only if the spectral radius $\varrho(G) < 1$.

The *asymptotic rate of convergence* is defined as

$$R_\infty(G) := -\ln \varrho(G),$$

and to reduce the error with a factor ϵ one needs $k_\epsilon = -\ln \epsilon / R_\infty(G)$ iterations. Unfortunately, the convergence rate of the different iteration methods are highly problem dependent. It is not obvious what method is the best. In Ref. [34] an analysis of a few cases and model problems are quoted and the iterative methods are interpreted in terms of a model problem. The text is highly recommended for further reading.

In the following, notice that when A is a sparse matrix the iterations become exceedingly simple to perform as each iteration only costs $\mathcal{O}(n)$ operations, where n is the number of non-zeroes in the matrix.

The classical iterative solvers are implemented in subclasses of `BasicItSolver` (which again is a subclass of `IterativeSolver` derived from `LinEqSolver`).

Simple Richardson Iteration and Jacobi Iteration. For the Richardson iteration (or just simple iteration) one chooses $M = I$. The system $Mx = b$ then becomes trivial. This yields the iteration

$$x^{k+1} = x^k + r^k = x^k - Ax^k + b.$$

For the Jacobi iteration we choose $M = D$, where D is the diagonal of A, i.e., $D_{ii} = A_{ii}$ and $D_{ij} = 0$ for $i \neq j$. Again, $Mx = b$ is very cheap. This yields for the iteration

$$x^{k+1} = D^{-1}((D - A)x^k + b) = x^k - D^{-1}Ax^k + D^{-1}b.$$

Writing out the equation for x_i^{k+1} yields

$$x_i^{k+1} = x_i^k + \frac{1}{A_{ii}} \left(b_i - \sum_{j=i+1}^{N} A_{ij} x_j^k \right), \quad i = 1, 2, \ldots, N.$$

Relaxed Jacobi Iteration. If we by x^* denote the Jacobi iteration, use instead

$$x^{k+1} = \omega x^* + (1-\omega)x^k,$$

i.e., a weighted mean between the old and new Jacobi iteration. It may turn out that using $0 < \omega < 1$ leads to faster convergence, but this is highly problem dependent.

Gauss-Seidel Iteration. As noted above in section 5.2.1, equations involving upper or lower triangular matrices are easy to solve by forward or backsubstitution. Hence, we could choose M to be the upper or lower triangular part of A, not to be confused with L or U in the LU decomposition. If we write $M = D + L + U$ where D is the diagonal from the Jacobi iteration and L and U is upper or lower-triangular, respectively, we may choose $M = D + L$ which defines the Gauss-Seidel iteration. Hence,

$$(D+L)x^{k+1} = -Ux^k + b.$$

Writing out the equations yields

$$x_i^{k+1} = \frac{1}{A_{ii}}\left(b_i - \sum_{j=1}^{i-1} A_{ij}x_j^{k+1} - \sum_{j=i+1}^{N} A_{ij}x_j^k\right), \quad i = 1, 2, \ldots, N.$$

Hence, it is similar to the Jacobi iteration, but we reuse the values of x_i^{k+1} in $x_{i'}^{k+1}$ for $i < i'$.

SOR and SSOR Iteration. In *successive over-relaxation* (SOR) we use Gauss-Seidel iteration with relaxation, viz.,

$$x^{k+1} = \omega x^* + (1-\omega)x^k,$$

where x^* is x^{k+1} from the Gauss-Seidel iteration. The relaxation parameter ω might be chosen to be greater than unity, and it is in fact a good choice for Poisson-type problems, see Ref. [34]. We obtain

$$M = \omega^{-1}D + L, \quad N = (\omega^{-1} - 1)D - U.$$

In *symmetric successice over-relaxation* (SSOR) one combines SOR with a backwards sweep, i.e., one first uses the lower triangular part in a "sweep" and then the upper triangular part (i.e., performing a Gauss-Seidel iteration with $M = D+U$). The matrix M is given by

$$M = \frac{1}{2-\omega}\left(\frac{1}{\omega}D + L\right)\left(\frac{1}{\omega}D\right)^{-1}\left(\frac{1}{\omega}D + U\right).$$

Reading from left to right (which is the order of appliance for M^{-1}) we see that first the lower triangular part is used and then the upper triangular part. Both are relaxed with ω as parameter.

5.2.3 Krylov Iteration Methods

An alternative approach to the presentation of the methods in this section can be found in Ref. [40]. Here we adopt the presentation in Ref. [34].

Krylov iteration methods are iterative methods originally proposed as direct methods for solving linear systems, but as such they are more expensive than Gaussian elimination. Nevertheless it turns out that the methods produce a good approximation to the exact solution in $k \ll N$ iterations. They do not fit into the framework of classical iterative methods as we cannot devise a "constant" matrix G as in Eqn. (5.3).

Again we seek a sequence of approximations x^0, x^1, x^2 et.c. that hopefully converge to the real solution x. The idea is to devise a subspace $V_k \subset \mathbb{C}^N$ with a basis q_j, $j = 1, \ldots, k$ such that

$$x^k := x^{k-1} + \delta x^k, \quad \delta x^k = \sum_{j=1}^{k} \alpha_j q_j,$$

i.e., δx^k is sought in V_k. We must determine the coefficients α_j and an algorithm for creating subspaces V_k at each iteration.

The subspaces employed are so-called *Krylov subspaces*. The iterative methods in this section is therefore referred to as *Krylov iteration methods* or *Krylov subspace methods*. The Krylov subspace is defined as

$$V_k(A, u) := \text{span}\,\{u, Au, A^2u, \ldots, A^{k-1}u\}.$$

Note that V_k need not be of dimension k. For more information on Krylov subspaces, see for example Ref. [41]. Our chosen Krylov subspace is $V_k = V_k(A, r^0)$, i.e., the span of iterates of the initial residual with A. Recall that

$$r^k := b - Ax^k = r^{k-1} - A\delta x^k.$$

To define the coefficients α_j at each iteration k, two choices are popular, and these may be viewed as a Galerkin method or least-squares method, respectively. For the Galerkin method we require that the residual r^k is orthogonal to V_k, i.e., that

$$(r^k, q_j) = 0, \quad j = 1, \ldots, k.$$

This leads to

$$\sum_{j=1}^{k} \alpha_j (Aq_j, q_i) = (r^{k-1}, q_i), \quad i = 1, \ldots k.$$

This is clearly a linear system of (at most) dimension k. This of course implies that at iteration N we need to solve a system at the size of A. On the other hand, if $\dim V_N = N$ we must have $r^N = 0$ so that the exact solution is found. In this way the method is a direct method, at least if r^0 and A are such that $\dim V_N = N$. The Galerkin method is referred to as *the conjugate gradient method.*

For the least-squares method, we minimize the norm of the residual with respect to α_i, i.e.,

$$\frac{\partial}{\partial \alpha_i}(r^k, r^k) = 0, \quad i = 1, \ldots k.$$

Assuming that A is a real matrix and that b is a real vector, we obtain

$$\frac{\partial r^k}{\partial \alpha_i} = -2\alpha_i (Aq_i, r^k) = 0.$$

Hence,

$$\sum_{j=1}^{k} \alpha_j (Aq_j, Aq_i) = (r^{k-1}, Aq_i), \quad i = 1, \ldots k.$$

We will not go into the details of finding the new basis vector q_k for V_k in the two methods, see instead Ref. [34]. Choosing some orthogonality condition and assuming some properties of A will however simplify the iteration process and the algorithm for q_k. We state the types of systems for which the Krylov iteration methods described here are effective. The Galerkin method needs a symmetric (Hermitian for complex A), positive or negative definite matrix, i.e., that all the eigenvalues of A are either

positive or negative. The least-squares can be used for every kind of real matrix but may require more temporary storage.

For sparse matrices each iteration needs $\mathcal{O}(n)$ operations, where n is the number of non-zero entries. If the method converges in $k \ll N$ operations, we have a method with speed comparable to the tridiagonal Gaussian elimination. A complication is the fact that we need a start vector x^0 to set off the iteration. For time dependent problems, choosing an initial vector x^0 (and hence an initial residual r^0) is easy; we simply use the solution at an earlier time step. The residual then starts out small and the method may converge quickly. As for the classical iterative methods, the convergence rate of the Krylov iteration methods are highly problem dependent.

In addition to the two mentioned above, there are also many other Krylov iteration methods implemented in Diffpack, such as SYMMLQ for non-definite symmetric (Hermitian) systems (such as our discrete Hamiltonian from both FEM and FDM) and BICGSTAB for non-symmetric systems. See the Diffpack documentation in Ref. [45], under the `KrylovIt` class which contains links to subclasses for the different solvers.

5.3 Review of Methods for Eigenvalue Problems

In contrast to solvers for linear systems of equations, eigenvalue solvers are exclusively of an iterative nature, at least for large systems. Iterative methods based on Krylov subspaces are also very popular for large systems. Ref. [41] is a thorough exposition on numerical methods for large eigenvalue problems and is highly recommended reading. For moderately-sized systems, Ref. [40] describes the widely-used methods of QR factorization, the Givens and Househölder methods et.c. for Hermitian standard problems. In addition, the documentation to ARPACK++ in Ref. [47] is very informative with respect to the Arnoldi method used in this thesis.

In this section we are concerned with the generalized eigenvalue problem, i.e.,

$$Ax = \lambda Bx,$$

where A and B are square matrices of dimension N, x is a vector and λ is a scalar. The classical methods used for moderately-sized matrices are for standard eigenvalue problems, i.e., for problems where $B = I$. We describe these first and then move on to the Krylov subspace methods such as the Arnoldi method of which the Lanczos method is a special case.

Is B is symmetric and positive definite it turns out that we may manipulate the eigenvalue problem to make it a standard eigenvalue problem. This will be addressed in section 6.3

Algorithms for eigenvalue and eigenvector computations are often very complicated, often due to complicated convergence properties, stability control and orthogonalization procedures. It is therefore usually not a good idea to implement the methods from scratch but instead use an existing library known to be stable, fast and that suits ones needs. On the other hand, a knowledge of the complexity and areas of application of the algorithms is necessary in order to choose the right method and also to invoke it properly. Hence, the below exposition is not meant to be complete. One should consult the references if more details are needed.

5.3.1 Methods For Standard Hermitian Eigenvalue Problems

The methods described in this section works for Hermitian matrices (and hence real symmetric matrices). They do not handle generalized problems.

The material in this section is taken from Ref. [40], and this reference is well worth spending some time on.

Jacobi Transformations. The method described here is fine for small matrices, but as the dimension increases, so do the number of iterations. One should then instead use the Househölder or Givens transformations described below.

Diagonalizing A is equivalent to finding a similarity transformation such that

$$D = V^{-1}AV$$

is diagonal. Then column vector i of V is an eigenvector of A with eigenvalue D_{ii}. If A is Hermitian, i.e., $A^\dagger = A$, the eigenvectors are all orthogonal which implies that $V^{-1} = V^\dagger$, i.e., V is unitary.

The idea in the Jacobi method is to use a sequence of orthogonal (i.e., unitary) transformations P^i to zero out off-diagonal elements in our matrix A, gradually reducing it to diagonal form. The diagonalizing operator is then given by

$$V = P^1P^2P^3\dots P^n.$$

The transformations P^i are referred to as *Jacobi rotations*. It is defined by $P^i_{pp} = P^i_{qq} = \cos\theta$, $P^i_{qp} = -P^i_{pq} = \sin\theta$ and $P^i_{ij} = \delta_{ij}$ otherwise. The angle θ is chosen such that A_{pq} and A_{qp} become zero under the similarity transform

$$A \longrightarrow A' = P^\dagger AP.$$

(The choice of p and q is of course dependent on i.) Unfortunately, performing a succession V of Jacobi rotations destroy the previous "annihilations" of non-zeroes, but when chosen appropriately the sequence yields convergence to D, albeit in an infinite number of steps.

Each rotation requires $\mathcal{O}(N)$ operations for a dense matrix. The number of rotations required to sweep through the whole matrix is about $N(N-1)/2$, i.e., the number of elements in the upper (or lower) triangular part. Typically one needs several sweeps to achieve convergence to machine precision; typical matrices require 6 to 10 sweeps. This totals $\mathcal{O}(20N^3)$ operations if 10 sweeps are required, according to Ref. [40]. This is of course prohibitive for dense matrices of large order.

If the matrix is sparse we may obtain better convergence properties, but the rotations introduces non-zeroes at new positions in the matrix.

Househölder and Givens Reductions. Using the Jacobi method for reducing A to diagonal form takes an infinite number of steps. Reducing it to tridiagonal form can however be done in a finite number of steps, and tridiagonal systems may efficiently be diagonalized by iterative methods such as the QR algorithm described below.

The Givens method chooses a sequence of (modified) Jacobi rotations that in a finite number of steps reduces any Hermitian matrix to tridiagonal form. The Househölder method is a little bit more efficient in achieving the same goal and generally preferred. See Ref. [40] for details on the Givens reduction and the Househölder reduction.

The QR *algorithm.* The QR method effectively diagonalizes a tridiagonal Hermitian matrix. In simple problems the coefficient matrix may be tridiagonal to begin with, but usually one obtains a tridiagonal problem by means of the Househölder reduction. The method is an iterative method that diagonalizes A through a series of unitary transforms. As the name suggests it is based on the well-known QR factorization, i.e., that any $N \times M$ matrix X can be rewritten as

$$X = QR,$$

where Q is an $N \times N$ unitary matrix and R is $N \times M$ and upper triangular. Indeed, it is simply a way of rewriting the standard Gram-Schmidt process for creating an

orthonormal basis (the columns of Q) for the column space of X. The matrix R provides the change of coordinates. Equivalently, we may decompose X into $X = QL$, where L is lower triangular. The corresponding method is called the QL decomposition. It can be shown that the QR (and QL) decomposition preserves both Hermiticity and tridiagonality of the matrix X.

Given our tridiagonal matrix $A = A_0$ we construct an iterative process as follows: Decompose $A_i = Q_i R_i$ and define

$$A_{i+1} := R_i Q_i = Q_i^\dagger A_i Q_i,$$

where the last equality follows from unitarity of Q_i. Hence, each iteration is a change of coordinates to an orthonormal basis for the columns of A_i. As is easily seen, one may choose either of the QL or QR decompositions to define the algorithm.

Under reasonable mild conditions on the eigenvalues of A this series can be shown to converge to a diagonal matrix and hence the diagonal contains the eigenvalues. To improve convergence (and to lessen the assumptions on A) one may introduce shifts at each iteration by instead factorize $A_i - \sigma_i I$ where σ_i is a scalar.

If A is tridiagonal each iteration can be implemented in only $\mathcal{O}(N)$ operations, yielding a very efficient iteration process, also for large systems. However, large systems are not that often tridiagonal and hence other methods is needed for such matrices, as the Givens reduction may be too complicated.

5.3.2 Iterative Methods for Large Sparse Systems

When the dimension of the matrices in our eigenvalue problem becomes very large the Househölder method described above fails badly as it requires too many operations to complete. Instead we must turn to pure iterative methods that work with A directly.

Common to the methods are the fact that each iteration uses (mostly) matrix-vector products with A (or some simple modification of A) as coefficients. If each product can be calculated fast the iteration process is in principle fast as well.

Ref. [41] is a comprehensive exposition into the methods discussed in this section. Ref. [39] also contains some details concerning the power method and the simple subspace iteration. The ARPACK user guide in Ref. [47] also describes the implicitly restarted Arnoldi method in detail.

The Power Method and Inverse Power Method. The simplest method for finding an eigenvalue of a standard eigenvalue problem is *the power method*. Given a start vector x^0 one simply defines the iteration

$$x^k = \frac{1}{\alpha^{(k)}} A x^{k-1},$$

where $\alpha^{(k)} = \max_j |x_j^k|$, i.e., the magnitude of the largest component of x^k. It is easy to see that if there is only one eigenvalue of largest magnitude and if this eigenvalue is not degenerated, the sequence will converge to the corresponding eigenvector. See Ref. [41] section 1.1 for a proof. The method converges faster the larger the ratio of the largest and next-to-largest eigenvalue magnitude is.

The power iteration method is the basis for the *inverse power iteration method*. Notice that the matrix $A - \sigma I$ has the same eigenvectors as A but that the spectrum is shifted by $-\sigma$, i.e., if λ is an eigenvalue of A then $\lambda - \sigma$ is an eigenvalue of $A - \sigma I$. Furthermore, if $A - \sigma I$ is non-singular, the matrix $B = (A - \sigma I)^{-1}$ has the eigenvalue $(\lambda - \sigma)^{-1}$. Hence, if we want an eigenvalue of A around σ we may iterate B instead in the power method as the eigenvalues of B corresponding to eigenvalues of A close to σ will dominate the process.

The eigenvectors of A, $A - \sigma I$ and $(A - \sigma I)^{-1}$ are all the same. The iteration with B may be done by performing an LU decomposition and successively solve the linear

system

$$Bx^* = x^k$$

and scale x^* with the appropriate factor to obtain x^{k+1}.

A process called *deflation* may be used in order to compute the next eigenvalue when the one of largest modulus is found. One may manipulate the matrix such that the largest eigenvalue is displaced to a lower value. Then a new iteration process will find the eigenvalue of highest magnitude.

The power method is appropriate if one seeks only a few eigenvalues and when a lot of information on A is known. More sophisticated methods are used if several eigenvalues and eigenvectors are needed, as in this thesis. Hence, similar to Gaussian elimination it is rarely used except for the case where only a very few eigenvalues are needed.

Simple Subspace Iteration. The power method and the inverse power method generates information at each iteration which is one-dimensional, i.e., only one vector is generated at each step. It would be nice if we instead choose m vectors, iterated these and used the extra information to obtain faster or more results, in some way. This is the idea of *the simple subspace iteration method.*

We start out with an $N \times m$ matrix X^0 whose column vectors span an m-dimensional subspace $V_m \subset \mathbb{C}^N$. If we use the power method on each column vector by iterating

$$X^{k+1} = A \cdot X^k \cdot D,$$

where D is an $m \times m$ diagonal matrix with renormalization factors, i.e., $D_{ii} = 1/\alpha_i^{(k)}$, each column vector of X^k will converge to the eigenvector with eigenvalue of highest magnitude. In other words, the columns will gradually lose their linear independence. The idea is then to re-establish linear independence once in a while. The column vectors of X^k will then converge to *several* eigenvectors if this is done properly.

The re-orthogonalization is simply a Gram-Schmidt process, creating a set of orthonormal vectors from a set of linear independent vectors. Every once in a while one computes the QR factorization of X^k and uses Q as starting point for further iterations, i.e., the orthogonal basis for the subspace spanned by the column vectors of X^k.

For further details on this algorithm see for example Ref. [41].

Krylov Subspace Methods. The methods based on Krylov subspaces are definitely the most complicated algorithms both to implement and use. They are however very fast, converge quickly and are applicable to very general matrices. The algorithm used in ARPACK is called *the implicitly restarted Arnoldi method* (IRAM) and is perhaps the most powerful Krylov method around. All the problems in this text are Hermitian, and traditionally most eigenvalue methods are developed for such systems. The reason is double: Most physics applications give rise to Hermitian problems and Hermitian problems are the easiest to solve. The IRAM finds eigenvalues and eigenvectors of real and complex matrices, both symmetric and non-symmetric and also handles generalized problems easily. It is also used in the commercial computational software Matlab for solving large and sparse eigensystems.

Similar to the Krylov subspace methods in section 5.2.3 for solving systems of linear equations the eigenvalue methods employ Krylov subspaces V_k. Notice the similarity with the definition of the Krylov subspace $V_k(A, x)$ and the simple iteration method. One basic idea in the Krylov subspace method is to exploit more of the information generated by a simple iteration sequence. This is described in detail in Ref. [47]. Some of the information is used in the subspace iteration but there is much more to gain. Actually, the IRAM is defined in terms of a Galerkin condition and hence there is a close relationship between the finite element methods, conjugate gradient-like methods for linear systems of equations and the IRAM.

Diving further into the IRAM is out of the scope of this text. The implementation in ARPACK is well-tested and robust and we will use it without hestitating. The important things to know is *what ARPACK can compute*, *what information* ARPACK needs to perform the iteration process, *how much each iteration costs* and *how much storage space we must set aside*.

As for what ARPACK can compute, it finds eigenvalues and eigenvectors in different parts of the spectrum. It may find the largest eigenvalues, the smallest, those with largest real or imaginary parts, centered around a shift σ and so on. We are for the most part interested in the lowest eigenvalues of the Hamiltonian which is the default.

Fortunately, the only information needed to solve a *standard problem* is simple matrix-vector products, used in subspace iterations inside the IRAM. Internally ARPACK sets aside an amount of memory proportional to the space required by about Nk real (or complex) numbers, i.e., an $N \times m$ matrix. The computational cost of each iteration varies slightly, but it is proportional to the number k of eigenvalues (or eigenvalues) sought and the number of operations needed for a matrix-vector product. The number of iterations needed varies however. In fact, seeking more eigenvalues does not necessarily slow down the process as it may take *fewer* iterations before convergence. This is due to the extra information in the Krylov subspace generated from more search vectors.

If we introduce shifting to the algorithm, i.e., seek eigenvalues around σ, linear systems must be solved with coefficient matrix $A - \sigma I$.

For a *generalized problems* we need to solve linear systems with B and $A - \sigma B$ as coefficient matrix. In other respects the method is performed similar to the standard problem.

We will keep an empirical approach to these matters and use ARPACK as a black box. In addition to Ref. [47], see the Simula report available from Ref. [5].

Chapter 6

Quantum Mechanical Eigenvalue Problems

In this chapter we will return to the time independent Schrödinger equation, i.e., the eigenvalue equation for the Hamiltonian:

$$H\Phi = E\Phi, \tag{6.1}$$

where E is a scalar. We will study discretized versions of the equation using finite difference and finite element methods.

Assume that Φ_n and E_n are the (orthonormal) eigenvectors and eigenvalues of H; we assume a discrete set of eigenvalues for simplicity and we always order the eigenvalues in increasing order as is the convention in quantum mechanical formalism. The time development of an initial state $\Psi(0) = \sum_n c_n(0)\Phi_n$ is given by

$$\Psi(t) = \mathcal{U}(t,0)\Psi(0) = \sum_n e^{-iE_n t/\hbar} c_n \Phi_n.$$

In other words, the coefficients evolve as

$$c_n(t) = e^{-iE_n t/\hbar} c_n(0).$$

Their change in time is only a phase change; the magnitude of c_n is conserved.

Introducing a spatial discretization leads to a natural discrete form of the eigenvalue problem, viz.,

$$H_h u = Eu,$$

where H is an $N \times N$ Hermitian matrix and u is an N-dimensional complex vector. The eigenvalues and eigenvectors of H_h will reflect the eigenvalues and eigenvectors of the full Hamiltonian H. Intuitively, the better approximation H_h is to H, the better the correspondence will be. We may also view H_h as the Hamiltonian for an altogether different physical system with finitely many degrees of freedom. The approximative character of H_h to H makes us expect that the behavior of this new system reflects the behavior of the original one; a line of thinking often applied in physics to capture essential features of a very complicated model.

As for the time development of an initial state given as a linear combination of the eigenvectors of H_h, it is of course given by the ODE (4.37). Indeed,

$$u(t) = \sum_{n=1}^{N} e^{-iE_n t/\hbar} c_n(0) u_n.$$

As discussed in chapter 5 we typically search for some $k \ll N$ eigenvalues and eigenvectors. On the other hand, when starting with a sufficiently regular initial condition, the coefficients c_n are neglible whenever $n > k$. In this way we may actually solve the time dependent Schrödinger equation for stationary problems.

For time dependent problems the time dependent part is typically some perturbation that is turned on and off. Before and after the perturbation we have a stationary problem, and the spectrum of the initial and final state with respect to the unperturbed Hamiltonian contains important information. For example, if we study a Hydrogen atom perturbed with a laser beam and start out in the ground state, the spectrum after the perturbation shows us the probability of exciting the ground state to higher states with a laser. Hence, the eigenvalue problem of the Hamiltonian is of great relevance also for time dependent problems.

In this chapter we will focus on time independent problems, solving numerically some analytically solvable problems. We will also treat a simple one-dimensional problem, finding the numerical solution analytically.

6.1 Model Problems

To investigate the properties of the finite element method we will investigate some model problems. These problems are analytically solvable and provides excellent means for testing finite element discretizations with different grids; in particular we will employ square grids and grids that approximate a circular region.

6.1.1 Particle-In-Box

First, we consider the particle-in-box problem, in which our particle is free to move inside the domain whose boundary defines the shape of the box. The Hamiltonian is

$$H = -\frac{\hbar^2}{2\mu}\nabla^2 + V(\boldsymbol{x}),$$

where the potential V is zero inside the domain Ω and infinite everywhere else. This leads to the boundary condition

$$\Psi(\boldsymbol{x}) = 0, \quad \forall \boldsymbol{x} \in \partial\Omega,$$

and to the time independent Schrödinger equation

$$-\nabla^2\Psi = \lambda\Psi, \quad \lambda = \frac{2\mu E}{\hbar^2}, \boldsymbol{x} \in \Omega.$$

For some particular shapes of Ω this problem can be solved analytically. We will consider two-dimensional geometries, namely a square and a circular domain Ω.

Square Domain. First consider the square domain $\Omega = [0,1] \times [0,1]$. We may use separation of variables, i.e., $\Psi(x,y) = u(x)v(y)$, which leads to the identical equations

$$\begin{aligned} -u''(x) &= \lambda_x u(x), \quad u(0) = u(1) = 0, \\ \text{and} \quad -v''(y) &= \lambda_y v(y), \quad v(0) = v(1) = 0. \end{aligned}$$

The solution is readily obtained, viz.,

$$u_n(x) = C\sin(n\pi x),$$

where the quantum number $n = 1, 2, \ldots$ labels the eigenfunctions and where C is an irrelevant normalization constant. Hence,

$$\Psi_{nm}(x,y) = C^2\sin(n\pi x)\sin(m\pi y) \quad \text{and} \quad \lambda_{nm} = \pi^2(n^2+m^2).$$

Note that whenever $n = m$ the eigenvalue is not degenerated. Otherwise, we have $\lambda_{nm} = \lambda_{mn}$, i.e., twice degenerated eigenvalues. (There could also be some accidental degeneracy due to solutions of the equation $n_1^2 + m_1^2 = n_2^2 + m_2^2$.)

Circular Domain. Second, consider a circular domain, viz.,

$$\Omega = \left\{(x, y) : x^2 + y^2 \leq 1\right\}.$$

This problem is more complicated to solve and involves Bessel functions and Neumann functions, see Ref. [24]. We will sketch some of the main steps in the solution. For a complete account, see Ref. [48].

Due to rotational symmetry it is wise do employ polar coordinates, i.e.,

$$x = r\cos\phi, \quad \text{and} \quad y = r\sin\phi.$$

The Laplacian in polar coordinates is given by

$$\nabla^2 = \frac{\partial^2}{\partial r^2} + \frac{1}{r}\frac{\partial}{\partial r} + \frac{1}{r^2}\frac{\partial^2}{\partial \phi^2},$$

as is readily obtainable by use of the chain rule for differentiation. We use separation of variables and write

$$\Psi(r, \phi) = e^{im\phi}R(r).$$

This yields for $R(r)$ the eigenvalue equation (called the radial equation[1])

$$\left[-\frac{\mathrm{d}^2}{\mathrm{d}r^2} - \frac{1}{r}\frac{\mathrm{d}}{\mathrm{d}r} + \frac{m^2}{r^2}\right] R(r) = \lambda R(r).$$

As for the number m it must be an integer for the wave function to be periodic and differentiable. Furthermore, the energy λ must be positive (due to H being positive definite) and we write $\lambda = k^2$. By means of changing variable to $\rho = rk$ and multiplying the radial equation by ρ^2 we obtain

$$\rho^2 R'' + \rho R' + (\rho^2 - m^2)R = 0,$$

which is called Bessel's equation. Its solutions are

$$R(\rho) = C_1 J_m(\rho) + C_2 N_m(\rho),$$

where J_m are called *Bessel functions of the first kind*, and where N_m are called *Neumann functions of the first kind*. As $N_m(0) = -\infty$ they give unphysical solutions, hence

$$R(\rho) = C J_m(\rho).$$

To impose the boundary conditions, note that J_m has infinitely many zeroes r_{ms}. These are typically tabulated, and Table 6.1 shows some of the numerical values for various integral m. Note that $J_{-m} = (-1)^m J_m$, and this yields degeneracy of the eigenvalues λ whenever $|m| > 0$. We obtain an eigenfunction $R(r)$ whenever k_{ms} is a zero of $J_m(\rho)$. Thus, we label the energies λ_{ms}. The energies are given by

$$\lambda_{ms} = k_{ms}^2$$

and are tabulated in Table 6.1. Fig. 6.1 shows the probability densities of the ground state and the state $m = 2$,$s = 3$, i.e., $|\Psi_{0,1}|^2$ and $|\Psi_{2,3}|^2$.

[1] In general, for rotationally symmetric potentials $V(r)$ the operator on the left hand side has the form $-\mathrm{d}^2/\mathrm{d}r^2 - r^{-1}\mathrm{d}/\mathrm{d}r + m^2/r^2 + V(r)$.

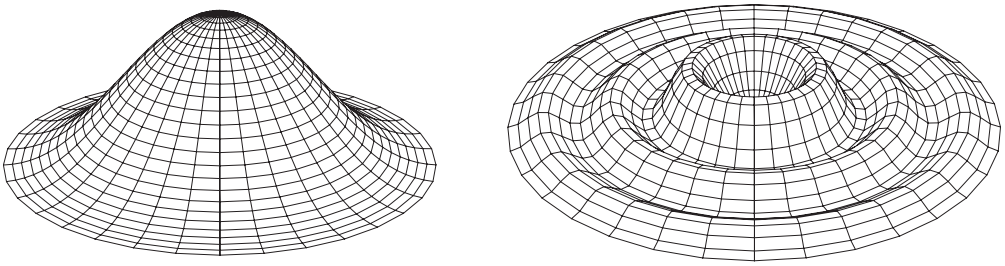

Figure 6.1: The two states $m = 0, s = 1$ (left) and $m = 2, s = 3$ (right) of the circular particle-in-box.

	$s = 1$	$s = 2$	$s = 3$	$s = 4$
k_{0s}	2.404825558	5.520078110	8.653727913	11.79153444
k_{1s}	3.831705970	7.015586670	10.17346814	13.32369194
k_{2s}	5.135622302	8.417244140	11.61984117	14.79595178
k_{3s}	6.380161896	9.761023130	13.01520072	16.22346616
λ_{0s}	5.783185964	30.47126234	74.88700679	139.0402844
λ_{1s}	14.68197064	49.21845632	103.4994540	177.5207669
λ_{2s}	26.37461643	70.84999891	135.0207088	218.9201891
λ_{3s}	40.70646582	95.27757254	169.3954498	263.2008542

Table 6.1: Some zeroes k_{ms} of the Bessel functions J_m and the corresponding energies $\lambda_{ms} = k_{ms}^2$

6.1.2 Harmonic Oscillator

The one-dimensional harmonic oscillator was solved in section 2.2. We now consider the two-dimensional isotropic harmonic oscillator, i.e.,

$$H\Psi = E\Psi,$$

where

$$H = -\frac{\hbar^2}{2\mu}\nabla^2 + \frac{1}{2}m\omega^2(x^2 + y^2).$$

Multiplying with $2\mu/\hbar^2$ and defining $\gamma^2/4 = \mu^2\omega^2/\hbar^2$ and $\lambda = 2\mu E/\hbar^2$ yields

$$\left[-\nabla^2 + \frac{\gamma^2}{4}(x^2 + y^2)\right]\Psi(x, y) = \lambda\Psi(x, y)$$

We use separation of variables and write $\Psi(x, y) = u(x)v(y)$. The corresponding equations read

$$\begin{aligned} -u''(x) + \frac{\gamma^2}{4}x^2u(x) &= \lambda_x u(x) \\ \text{and} \quad -v''(y) + \frac{\gamma^2}{4}y^2v(y) &= \lambda_x v(y). \end{aligned}$$

These equations are simply two one-dimensional harmonic oscillators.

From section 2.2 we have $E_n = \hbar\omega(n+1/2)$ for the eigenvalues of a one-dimensional oscillator. The total energy of the two-dimensional oscillator is thus

$$\lambda_{nm} = \gamma(n + m + 1), \quad n, m = 0, 1, 2, \ldots$$

where we have used $\lambda = 2\mu E/\hbar^2$. Hence, the eigenvalues are precisely $\gamma\nu$ where ν is any positive integer. The multiplicity of each eigenvalue is easily seen to be ν.

Note that due to rotational symmetry we could write $\Psi(r,\phi) = e^{im\phi}R(r)$ as with the circular particle-in-box. However, the radial equation for the two-dimensional harmonic oscillator is more difficult to solve.

When solving the harmonic oscillator numerically, we will choose a circular and finite domain. This will in effect set the potential to infinity at a finite distance from the origin, a source of error in our eigenvalues and energies. However, if we choose the radius of the disk sufficiently large we should be able to reproduce many of the lowest-lying energies accurately.

6.1.3 Two-Dimensional Hydrogen Atom

The two-dimensional hydrogen atom was discussed in section 3.2.2. We did not however discuss the time independent Schrödinger equation for this problem.

In polar coordinates (r,ϕ) and in the symmetric gauge the Hamiltonian is given by

$$H = -\nabla^2 - i\gamma\frac{\partial}{\partial\phi} - \frac{2}{r} + \frac{\gamma^2}{4}r^2, \tag{6.2}$$

where the units of length, energy and the magnetic field is given by Table 3.1.

Weak Field Limit. In Ref. [29] the limits in which the field γ is weak and strong, respectively, are treated perturbatively. The limit $\gamma = 0$ yields a pure Coulombic Hamiltonian, viz.,

$$H^{\mathrm{w}} = -\nabla^2 - \frac{2}{r}.$$

Similarly to the harmonic oscillator and the particle-in-box, we employ separation of variables, i.e., $\Psi(r,\phi) = e^{im\phi}R(r)$. The radial equation then reads

$$\left[-\frac{\mathrm{d}^2}{\mathrm{d}r^2} - \frac{1}{r}\frac{\mathrm{d}}{\mathrm{d}r} + \frac{m^2}{r^2} - \frac{2}{r}\right]R_n(r) = \epsilon_{nm}R_n(r). \tag{6.3}$$

The energies are given in Ref. [29] as

$$\epsilon^{\mathrm{w}}_{nm} = -\frac{1}{(n-1/2)^2}, \quad m = -n+1, -n+2\ldots, n-2, n-1.$$

Hence, we have a $2n-1$-fold degeneracy of the eigenvalues. We omit the derivations of the energies and the radial functions as they are lengthy and not particularly interesting.

The ground state energy in normal units is

$$E_0 \approx 13.603 \text{ eV} \cdot \epsilon^{\mathrm{w}}_{10} = -54.41 \text{ eV}.$$

Thus, the electron in the two-dimensional hydrogen atom is more bound than in the three-dimensional hydrogen atom, in which $E_0 \approx -13.603$ eV.

Strong Field Limit. If we consider the strong field limit the Coulomb term is considered as vanishing,[2] viz.,

$$H^{\mathrm{s}} = -\nabla^2 - i\gamma\frac{\partial}{\partial\phi} + \frac{\gamma^2}{4}r^2.$$

In other words, the electron is considered free except for the influence of the magnetic field. The energies are given by

$$\epsilon^{\mathrm{s}}_{NM} = 2\gamma(N+1/2).$$

[2] Although we have a singularity in the potential at $r=0$ the expectation value $\langle 1/r\rangle$ is dominated by $\gamma^2\langle r^2\rangle$ for large γ. This justifies the perturbative treatment.

The azimuthal quantum number m is given by the quantum numbers M and N through $m = N - M$.[3] The number N numbers the so-called Landau-levels. We see that the Landau levels are degenerate, and in fact they are infinitely degenerate.

In particular the ground state energy (i.e., energy of the lowest Landau level) is $\epsilon_{0M} = 1$ and the corresponding wave function is a Harmonic oscillator ground state multiplied with an arbitrary function of $x + iy$, i.e.,

$$\Psi_0 = f(x + iy) \cdot e^{-r^2/4},$$

where f is analytic. A basis for the ground state eigenspace is then $\psi_{0,m} = (x + iy)^m e^{-r^2/4}$ with $m = 0, 1, 2, \ldots$. This is the solution found in the *symmetric gauge*, see Ref. [7]. We may change the Hamiltonian by performing a gauge transformation of the vector potential. An example of a different but physically equivalent gauge is

$$\boldsymbol{A} = \gamma(-y, 0, 0),$$

whose corresponding magnetic field is easily seen to be identical to that if $\boldsymbol{A}_{\text{symm}}$, i.e., $\boldsymbol{B} = \gamma\hat{k}$. The solution to a gauge transformed problem is equivalent to the original solution and given by an explicit unitary transform, see section 1.6. The magnetic system with the alternative gauge is solved in Ref. [48].

The infinite degeneracy of the ground state is easy to fathom intuitively. There is no sense of localization in the problem definition; the electron is free and the magnetic field is constant throughout space. If $e^{-\boldsymbol{x}^2/2}$ was a ground state, so must $e^{-(\boldsymbol{x}-\boldsymbol{x}_0)^2/2}$ (although these states are not orthogonal). Every choice of $\boldsymbol{x}_0$ gives rise to a different ground state and the set of these states are easily seen to span an infinite dimensional space.

If the domain is on the other hand a disk with radius r_0 (such as in our simulations below) the degeneracy is instead

$$g = \frac{r_0^2}{2}.$$

See Refs. [7, 48].

In the symmetric gauge, the states $\psi_{0,m}$ are states that are easy to interpret with the correspondence principle. One can show that for large m the probability density $|\psi_{0,m}|$ is concentrated in a circle with radius $r_m = \sqrt{2m/\gamma}$. Classically an electron in a magnetic field moves in circular paths with a radius that increases with the kinetic energy. The kinetic energy of the quantum particle also increases with m. Furthermore, the classical path's radius is given by

$$r = \frac{v^2}{a} = \frac{2cT}{qvB} = \frac{\sqrt{2\mu T}c}{qB},$$

which, if we assume $T = \hbar^2 m^2/r^2$ quantum mechanically, becomes precisely

$$r = \sqrt{\frac{2m}{\gamma}}$$

in our case.

6.2 The Finite Element Formulation

In this section we will describe how we arrive at the discretized eigenvalue problem for the discrete Hamiltonian H_h with the finite element method.

[3] The quantum number N is not to be confused with the dimension of the matrices in the numerical problem. It should be clear from the context which N we refer to.

Assume that we are given a set of m finite element basis functions $N_i(\boldsymbol{x})$. Hence, the subspace $V \subset \mathcal{H}$ has dimension m. These functions must be defined appropriately when given a grid G of m nodes representing an (approximate) subdivision of our domain Ω. As usual with the finite element method we assume that the exact solution Ψ is well approximated with an element in the finite element space V, viz.,

$$\Psi(\boldsymbol{x}) \approx \hat{\Psi}(\boldsymbol{x}) = \sum_{j=1}^{m} u_j N_j(\boldsymbol{x}).$$

Inserting $\hat{\Psi}(\boldsymbol{x})$ for Ψ in the time independent Schrödinger equation, multiplying with $N_i(\boldsymbol{x})$ and integrating over the domain Ω yields

$$\sum_j u_j \int_\Omega N_i(\boldsymbol{x})[HN_j(\boldsymbol{x})] = E \sum_j u_j \int_\Omega N_i(\boldsymbol{x}) N_j(\boldsymbol{x}). \tag{6.4}$$

If we write u for the vector whose jth component is u_j, we arrive at the generalized eigenvalue equation

$$Au = \lambda M u, \tag{6.5}$$

where M is the mass matrix and A is the element matrix obtained by integrating by parts any ∇^2 term in H. As an example, consider the two-dimensional Hydrogen atom Hamiltonian, viz.,

$$\begin{aligned} H &= -\nabla^2 - i\gamma \frac{\partial}{\partial \phi} - \frac{2}{r} + \frac{\gamma^2}{4} r^2 \\ &= -\nabla^2 - i\gamma \left(-y \frac{\partial}{\partial x} + x \frac{\partial}{\partial y} \right) - \frac{2}{\sqrt{x^2 + y^2}} + \frac{\gamma^2}{4}(x^2 + y^2). \end{aligned} \tag{6.6}$$

The first term becomes the stiffness matrix K, and the second term yields a matrix L whose elements are

$$L_{ij} = -i\gamma \int_\Omega N_i(\boldsymbol{x}) \left(-y \frac{\partial}{\partial x} + x \frac{\partial}{\partial y} \right) N_j(\boldsymbol{x}).$$

The Coulomb term yields a matrix C given by

$$C_{ij} = \int_\Omega N_i(\boldsymbol{x}) \frac{2}{\sqrt{x^2 + y^2}} N_j(\boldsymbol{x}),$$

and the last harmonic oscillator term yields a matrix O, viz.,

$$O_{ij} = \frac{\gamma^2}{4} \int_\Omega N_i(\boldsymbol{x})(x^2 + y^2) N_j(\boldsymbol{x}),$$

The total element matrix is then $A = K + L + C + O$.

Note that no boundary conditions have been imposed as of yet. Hence A contains unknown boundary terms from the integration by parts. These terms will however be eliminated.

How do we impose the boundary conditions in an eigenvalue problem such as this? The homogenous boundary conditions state that $u_k = 0$ whenever $\boldsymbol{x}^{[k]} \in G$ is a boundary node of our grid. If we diagonalize the pair (A, M) as it stands in Eqn. (6.5) we have no guarantee whatsoever that a boundary component u_k of an eigenvector is zero. Furthermore, doing what is usually done in finite element applications, i.e., letting $A_{kk} = 1$ and $A_{kj} = 0$ whenever $j \neq k$ gives no meaning in this case as we do not deal with equations on the form $Au = b$. (In addition to modifying A we would set b_k equal to the boundary condition; in our case zero.)

Imposing homogenous conditions in an eigenvalue problem fortunately turns out to be very simple in principle. It amounts to erasing row k and column k from both matrices A and M, and at the same time erase u_k from u. In other words, we reduce the dimension of our problem with 1 for each boundary node and remove every reference to u_k in the equations. As matrices may be represented in many different ways in the computer, e.g., dense matrices (`Mat` in Diffpack), banded matrices (`MatBand`) or sparse matrices (`MatSparse`), implementing this principle must be done for every kind of matrix used in the finite element context; a task that involves non-trivial algorithms and speed considerations.

Let us first state the boundary condition imposing process in a more abstract way to make it clearer. Let P be the projection matrix onto a subspace W of $\mathbb{C}^m \simeq V$. We want W to be the subspace that corresponds to the interior of Ω, i.e.,

$$P : \mathbb{C}^m \longrightarrow \mathbb{C}^{m-n},$$

where n is the number of boundary nodes in G, and P is hence an $(m-n) \times m$ matrix. Indeed, it is the identity matrix I_m with the row number k removed for all boundary nodes $\boldsymbol{x}^{[k]}$. The matrix P then maps $u \in \mathbb{C}^m$ to a vector $v \in \mathbb{C}^{m-n}$ with the boundary node values removed.

The above reduction on dimensionality for all boundary nodes can then be written as

$$\tilde{A}v = \lambda \tilde{M}v, \tag{6.7}$$

where $v = Pu$ and where $\tilde{X} = PXP^T$ for any $m \times m$ matrix X. Note that $\tilde{X}$ equals X with the rows and columns corresponding to boundary nodes removed.

If $\boldsymbol{x}^{[k]}$ is a boundary node, then u_k does not appear anywhere in the equations. On the other hand, if u_k was *a priori* known to be zero, then the equations would look like Eqn. (6.7). Notice that the mentioned boundary terms in the matrix A resided precisely in row and column k, which are now removed from the problem.

As for the implementation of the matrices A and M, i.e., the integrals of Eqn. (6.4), numerical integration must be used in general. The mass matrix M is evaluated exactly if we use Gaussian quadrature of sufficiently high order.[4] In fact, Diffpack provides class methods that creates the mass matrix automatically. As in the above example, the element matrix A may contain more complicated terms. Each potential term must be considered in particular. For example, both the harmonic oscillator term O and the angular momentum term L contain a simple polynomial expression that is very well tackled by Gaussian integration. On the other hand, the Coulomb term C contains an $1/r$ term which may or may not be well integrated with Gaussian integration as it has a singularity. In all our applications we will use default Gaussian integration.

6.3 Reformulation of the Generalized Problem

Solving a generalized eigenvalue problem is much more difficult than solving a standard problem. The numerical methods presented in chapter 5 were mostly only applicable to standard problems with Hermitian matrices.

We observe that the mass matrix M is symmetric and positive definite. Let V_h be our finite element space and let $u_h \in V_h$ be arbitrary and nonzero. Let N_i, $i = 1, 2, \ldots, m$ be our finite element basis. Hence, $M_{ij} := (N_i, N_j)$ and M is a real, symmetric matrix (if we assume N_i to be real). For the norm of u_h we obtain

$$0 < (u_h, u_h) = (\sum_i u_i N_i, \sum_j u_j N_j) = \sum_{ij} u_i^* u_j (N_i, N_j) = u^\dagger M u,$$

[4] Gaussian integration evaluates the integral of a polynomial of a certain degree exactly. See Ref. [34] for a description of common Gaussian integration rules that are implemented in Diffpack.

and hence M is positive definite. Here u is the $\mathbb{C}^N$ vector whose components are u_j.[5]

Any positive definite matrix is invertible. Hence, our generalized eigenvalue problem is equivalent to

$$M^{-1}Hx = \lambda x.$$

Alas, the matrix $M^{-1}H$ is not Hermitian, rendering the algorithms from chapter 5 useless. Luckily, the *Cholesky decomposition* comes to our rescue.

It is well-known that every symmetric and positive definite (real) matrix M can be rewritten as

$$M = L \cdot L^T,$$

where L is a lower triangular matrix. This decomposition is called the Cholesky decomposition.

We now state a theorem.

Theorem 18
Given the generalized eigenvalue problem

$$Hu = \lambda Mu,$$

where H is Hermitian and M is symmetric, real and positive definite. Let (u, λ) be an eigenpair. Let $M = LL^T$ be the Cholesky decomposition of M. Then the matrix $C = L^{-1}H(L^{-1})^T$ is Hermitian and has the eigenpair (L^Tu, λ).

Proof: Hermiticity is easily seen to hold, viz.,

$$C^\dagger = (L^{-1}H(L^{-1})^T)^\dagger = [(L^{-1})^T]^\dagger H^\dagger (L^{-1})^\dagger = L^{-1}H(L^{-1})^T.$$

Multiplying $Hu = \lambda Mu$ with L^{-1} from the left yields

$$L^{-1}Hu = \lambda L^T u,$$

and writing $v = L^T u$ gives

$$L^{-1}H(L^{-1})^T v = \lambda v,$$

and we are finished. ■

From this theorem it is easy to see that solving the eigenvalue problem

$$Cv = \lambda v$$

gives the correct eigenvalues. Furthermore, obtaining $u = (L^T)^{-1}v$ is efficiently done with backsubstitution. One potential problem is the fact that C is a dense matrix and hence matrix-vector products, if they may be computed at all, is an $\mathcal{O}(N^2)$ process. However, to compute $v' = Cv = [L^{-1}H(L^T)^{-1}]v$ we may follow these steps:

1. Solve the equation $L^T x = v$ by backsubstitution; an $\mathcal{O}(N^2)$ process in worst case but much faster if L^T is banded.
2. Compute $y = Hx$ by matrix multiplication.
3. Solve $Lv' = x$ by forward substitution.

It is easy to see that if H is a sparse matrix and M is sparse and stored in a banded format, this process is efficient.

We mention that the matrix L of the Cholesky decomposition of a banded matrix is again banded with the same bandwidth. Hence, a general sparse structure of M should

[5] I apologize for the ambiguous notation here.

not be chosen but rather a banded structure, allowing the Cholesky decomposition to be made *en place*.

Unfortunately, there was not time to implement this strategy in the HydroEigen solver for the two-dimensional hydrogen atom. Even though it does not represent any addition of flexibility except for the ability to solve sparse generalized eigenvalue problems, it would represent a considerable speed-up in the simulations. It is on the other hand an obvious future project to implement the Cholesky factorized mass matrix for finite element eigenvalue problems.

6.4 An Analysis of Particle-In-Box in One Dimension

As an introduction to the work with discretized eigenvalue problems we will analyze the particle-in-box in one dimension, using both finite difference methods and finite element methods. The exact eigenfunctions and eigenvalues can be obtained, and will serve as an indicator of the behavior of both methods.

Recall the eigenvalue equation for a particle in a box, viz.,

$$-u''(x) = \lambda u(x), \quad x \in [0, 1],$$

with the boundary conditions

$$u(0) = u(1) = 0.$$

This problem has the eigenvectors and corresponding eigenvalues given by

$$u_k(x) = \sin(k\pi x), \quad \lambda_k = (k\pi)^2,$$

as stated in section 6.1.1.

For the discretized version, we use a uniformly spaced grid with $N+1$ points, hence the grid spacing is $h = N^{-1}$ and the grid points are given by $x_j = hj$, $j = 0, 1, \ldots, N$. If we use the standard difference scheme

$$[-\delta_x \delta_x v(x) = \lambda v(x)]_j,$$

we obtain a finite-dimensional eigenvalue problem, viz.,

$$Av = \lambda v, \tag{6.8}$$

where v is the components of the discrete solution, i.e., $v_j = v(x_j)$, $j = 1, \ldots, N-1$. (The components v_0 and v_N are identically zero due to the boundary conditions.) The matrix A is an $(N-1) \times (N-1)$ symmetric and positive definite matrix.[6] Hence, we expect $N-1$ positive and real eigenvalues for this problem.

Let us assume that the discrete eigenfunctions are given as

$$v_k(x) = \sin(k\pi x),$$

i.e., we guess that the eigenfunctions have the same form as in the continuous problem. They clearly fulfill the boundary conditions and the components are given by

$$v_j = \sin(k\pi hj).$$

(We omit the subscript k to economize.) We will not consider the matrix A explicitly, but instead write out the difference equations, viz.,

$$-\frac{1}{h^2}(v_{j-1} - 2v_j + v_{j+1}) = \lambda v_j, \quad j = 1, \ldots, N-1.$$

[6] See for example Ref. [33].

Using the trigonometric identity

$$\sin(x+y)+\sin(x-y)=2\cos y\sin x, \tag{6.9}$$

we obtain

$$v_{j-1}+v_{j+1}=2\cos(k\pi h)\sin(k\pi hj)=2\cos(k\pi h)v_j,$$

and hence,

$$-\delta_x\delta_x v_j=\frac{1}{h^2}(2-2\cos(k\pi h))v_j.$$

Using $1-\cos(x)=2\sin^2(x/2)$, we arrive at

$$-\delta_x\delta_x v(x_j)=\lambda_k v(x_j),$$

with

$$\lambda_k=\frac{4}{h^2}\sin^2(\frac{k\pi h}{2}),$$

and indeed the assumed form of $v(x)$ is valid. The integer k now numbers the eigenvalues, but there are not infinitely many as A has at most $N-1$ eigenvalues. It is easy to see that $v_N\equiv 0$, so it is not a proper eigenfunction, and furthermore v_{N+k} does not yield further eigenvectors, this due to periodicity of $v_k(x)$. But for $k=1,\ldots,N-1$ we have distinct eigenvalues.

In summary, the eigenfunctions are identical to the $N-1$ first ones of the continuous problem, but the eigenvalues are not the same. We may estimate the deviation by using a Taylor expansion for $\sin^2(x)$, viz.,

$$\sin^2(x)=x^2-\frac{1}{3}x^4+\mathcal{O}(x^6).$$

This yields

$$\lambda_k=k^2\pi^2\left(1-\frac{1}{12}\pi^2k^2h^2+\mathcal{O}(k^4h^4)\right).$$

Now we turn to a simple finite element approach, employing linear and uniformly sized elements with the node points defined by x_j as in the finite difference case. We will obtain a generalized eigenvalue problem reading

$$Kv=\lambda Mv, \tag{6.10}$$

where K and M are the stiffness matrix and mass matrix, respectively. These are $(N-1)\times(N-1)$ matrices given by

$$K_{i,i}=\frac{2}{h},\quad K_{i,i\pm1}=-\frac{1}{h},\quad K_{ij}=0 \text{ otherwise},$$

and

$$M_{i,i}=\frac{4h}{6},\quad M_{i,i\pm1}=\frac{h}{6},\quad M_{ij}=0 \text{ otherwise}.$$

Note that $K=hA$, i.e., the stiffness matrix is essentially the finite difference operator used in the above analysis. In this case we also expect $N-1$ real eigenvalues. If we try the same discrete eigenfunctions $v_k(x)$ as above, the left hand side becomes

$$[Kv]_j=\frac{4}{h}\sin^2(\frac{k\pi h}{2})v_j.$$

If we write out the jth component of Mv, we get

$$[Mv]_j=\frac{h}{6}\left(v_{j-1}+4v_j+v_{j+1}\right).$$

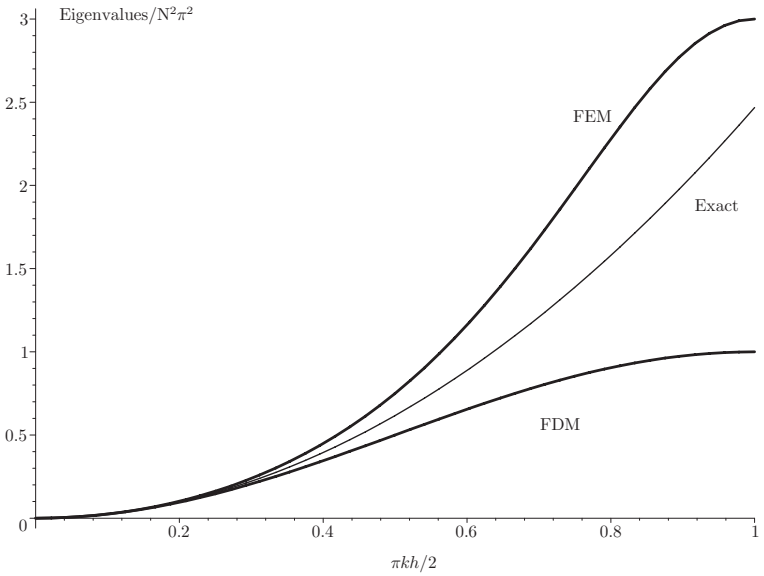

Figure 6.2: Plot of eigenvalues

We use the trigonometric identity (6.9) and obtain

$$[Mv]_j = \frac{h}{3}(2 + \cos k\pi h)v_j.$$

With the identity $1 + \cos(x) = 2\cos^2(x/2)$ we obtain

$$[Mv]_j = \frac{h}{3}\left(1 + 2\cos^2\frac{k\pi h}{2}\right)v_j.$$

Hence, $v_k(x)$ is an eigenfunction of M also. We may convert the generalized eigenvalue problem to a standard one by writing

$$M^{-1}Kv = \lambda v,$$

and hence we obtain

$$\lambda_k = \frac{12}{h^2}\frac{\sin^2\frac{k\pi h}{2}}{1 + 2\cos^2\frac{k\pi h}{2}}.$$

Using the Taylor expansion for λ, we get

$$\lambda_k = k^2\pi^2\left(1 + \frac{1}{12}\pi^2k^2h^2 + \mathcal{O}(k^4h^4)\right).$$

Figure 6.2 shows a plot of the normalized eigenvalues for the finite difference, finite element and exact calculations, respectively. Clearly, the finite difference eigenvalues underestimate the exact ones, while the finite element eigenvalues overestimate the eigenvalues. Both numerical methods yield good approximations for the lowest eigenvalues, but the finite element calculations are clearly qualitatively more correct than the finite difference calculations for the higher eigenvalues.

Several questions and interesting topics arise already at this point. The finite difference method is equivalent to *lumping the mass matrix,* i.e., replace M by the diagonal matrix M' whose element $M'_{i,i}$ is the sum of the elements in row i of M. Seemingly, the finite element method yielded qualitatively better results, but is this

true in general? We shall see that the answer is not affirmative: Lumping the mass matrix may improve the convergence of the eigenvalues for some systems.

Actually, if lumping the mass matrix turns out to be fortunate one must not immediately conclude that the finite element method is worthless in the case at hand, as for example the geometry may play a significant role in the precision. A combination of lumping the mass matrix with using higher-order elements and exotic geometries may turn out to be very powerful.

Speaking of which: How fast does the eigenvalues of the discrete problem converge to the exact values? In the above application the convergence was $\mathcal{O}(h^2)$ in both finite difference and finite element approximations, and in fact the leading terms in the error were identical in the two cases. If we use higher-order elements, will the convergence be correspondingly higher? This could clearly give the finite element method an advantage.

We will address these questions when doing numerical experiments below.

Actually, the solution method used in this section may be extended to two (or more) dimensions by using separation of variables analogously to the continuous problem. If we write

$$U_{kl}^{nm} = v^n(x_k)v^m(y_l),$$

where the superscript indicates the quantum numbers in the different directions we obtain

$$(-\delta_x\delta_x - \delta_y\delta_y)U_{kl}^{nm} = (\lambda_n + \lambda_m)U_{kl}^{nm},$$

where

$$\lambda_{nm} = \frac{4}{h^2}\left(\sin^2(\frac{n\pi h}{2}) + \sin^2(\frac{m\pi h}{2})\right),$$

is the numerical eigenvalues of the two-dimensional finite difference formulation. As for the mass matrix, one may find similar results, making it possible to obtain the exact eigenvalues also in the finite element case with linear elements.

Nevertheless, we will use the computer program described in section 6.5 to numerically diagonalize our particle in box. This in order to test our implementation which may be used with more complicated problems.

6.5 The Implementation

When programming with Diffpack one usually implements a class derived from one of the preexisting solver classes. In this case we derive `class HydroEigen` from `class FEM`, the base class for finite element solvers. `HydroEigen` is instantiated in the `main()` function and the simulation is then run.

The class definition reimplements various virtual functions in `class FEM` which performs certain standardized tasks, such as initialization, evaluation of integrals and so on. In this way the programmer of a finite element solver does not need to bother with finite element algorithms or the sequence of actual functions from `class FEM` that is called, but instead focus on the formulation of the problem.

Listings for `class HydroEigen` can be found in appendix B. The source code is available from Ref. [5] as well. The `main()` function is not listed in conjunction with the eigenvalue solver. When we turn to time dependent implementations in chapter 7, we derive `class TimeSolver` from `class HydroEigen` to make a more general program. The `main()` function listed instantiates `class TimeSolver` instead.

The eigenvalue problems that are implemented in our class is the two-dimensional hydrogen atom with an applied magnetic field. Furthermore, terms in the Hamiltonian may be turned on and off at will so that other problems such as the free particle or the harmonic oscillator may be solved as well.

We will inevitably touch upon features of Diffpack that are beyond the scope of this text to describe further. Refs. [34, 45] contains a thorough description of most features used. However, the purpose of each feature should be clear from the context. The code is well commented and is recommended for additional reading.

The program was compiled in the Debian Linux operating system with GNU gcc 2.95.4.

We mention some important command line parameters for the application here. These are standard for Diffpack programs. The name of the executable is assumed to be `SchroedSolve.x`. (The makefiles are not listed in the appendix, but it can be obtained from Ref. [5].) If one passes `--GUI` as parameter, the menu system will be displayed in a graphical user interface. Otherwise, a command based interface in the console is used. If `--casename` *`cname`* is passed, the simulation casename will be set to *`cname`*. The casename is the base name of files produced by the simulator such as the log file (*`cname`*`.log`), the simres database holding fields, grids, curves et.c. (`.`*`cname`*`.simres`) and so on. The casename feature is particularly useful when performing a sequence of simulations with varying parameters, e.g., the magnetic field, the grid size and initial condition. Finally, we have the `--verbose 1` command line parameter that displays various runtime statistics, such as the time spent on solving linear systems of equations.

During execution Diffpack also provides a lot of error and warning messages in case of unforeseen events, such as memory overflow, erroneous grid parameters and so on.

6.5.1 Class methods of `class HydroEigen`

Here we describe the most important class methods. Hopefully this will draw a clear picture of how the class works. But first we list some important class members that have a central role:

`Handle(GridFE) grid;` This is a handle (i.e., a smart pointer) to a finite element grid.

`Handle(DegFreeFE) dof;` This is a handle to a Diffpack object that holds the mapping $q(e, r)$ from local node numbers and element numbers to global node numbers. It determines the correspondence between field components U_j and the nodes $\boldsymbol{x}^{[k]}$ in the grid.

`Handle(LinEqAdmFE) lineq;` This object is a wrapper around linear systems of equations and solver methods. The solvers are also implemented as classes and are connected to this object via the menu system described below.

`Handle(Matrix(NUMT)) M;` This is a handle to the mass matrix. Another handle `K` points to the element matrix of the Hamiltonian.

The following description of the member functions is not exhaustive. It only describes the most important functions and lines of thinking. As always in large programming projects a lot of small and big problems needs to be solved, and to describe them all in this text would be both lengthy and unnecessary. The code is however well documented, such that algorithms for such things as removing a row and a column in a sparse matrix should be easy to read and understand.

`define()`. An important feature of Diffpack is the *menu system*. Every parameter to a solver should be accessible via the menu system, and with proper usage experimenting with for example solver methods for linear equations become very easy. The menu system may be run in command-mode in which different items are set with commands from standard in like `'set gamma=1.0'`. A list of available commands is listed with the `'help'` command.

If a Diffpack class comes with a (virtual) `define()` method, it usually adds some knobs and handles to the menu system. The `define()` method of `class HydroEigen`

adds parameters that may be used to adjust the definition of the problem, desired matrix encoding scheme and so on.

The entries defined are as follows:

– `gridfile` File to read grid from or a preprocessor command that is used to create the grid.

– `nev` Number of eigenvalues to search for.

– `nucleus` Boolean (with value `true` or `false`) that turns on or off the Coulomb attraction in the Hamiltonian.

– `epsilon` When evaluating the Coulomb term in the Hamiltonian, $1/r$ must be calculated. If r is too small this may lead to overflow and loss of accuracy in the calculations. Therefore, if $r <$ `epsilon`, we use $1/$`epsilon` instead in the integrand.

– `gamma` Strength of magnetic field. If this is zero together with `false` for `nucleus`, the problem will turn into particle-in-box.

– `angmom` A boolean that turns on or off the term in the Hamiltonian (6.2) proportional to the angular momentum $i\partial/\partial\phi$. Setting the parameter to `false` together with the `nucleus` parameter turns the problem into a harmonic oscillator.

– `nitg` Number of integration points if trapezoidal integration is preferred over Gaussian integration. A zero value turns on Gaussian integration. See comments in listing for details on the trapezoidal rule used.

– `lump` A boolean variable indicating whether or not lumping the mass matrix should be done.

– `warp` The grading parameter w described on page 144. The grading makes the grid points and elements concentrate around the origin if $w > 1$. This feature may be used to improve the accuracy of the numerical integration around the origin.

– `scale` A scalar that is used to uniformly scale the grid *after* warping. For example, a value of 2 will double the size of the grid in each direction.

– `renum` A boolean variable indicating whether or not an optimization of the $q(e, r)$ mapping is to be done in order to minimize the bandwidth of the element matrices. If banded matrices are used this option should be set to `true`. For sparse matrices it has no practical significance except for that it spends some time. (A lot, actually, if the grid is large.)

– `savemat` A boolean that indicates if one wishes to save the element matrices after they are assembled. Could be useful if one wishes to compare the efficiency of the program with, e.g., Matlab or Maple. Only works for sparse matrices and they are stored in Matlab compliant `.m`-files.

– `use arpack` A boolean that indicates if ARPACK is to be invoked for solving the eigenvalue problem. Usually this is the case, but if one simply wishes to test other features (such as timing the assembly process) it could be handy to be able to turn it off.

– `gauge` A string literal with value `symmetric` or `non-symmetric` indicating the gauge to be used for the magnetic field.

– `store_evecs` If set to a value outside [0,`nev`], all the eigenvectors are stored. They are chosen for storing in order of increasing eigenvalue. Storing eigenvectors take up a lot of drive space, so for large problems it could be useful to store, e.g., only the ground state.

– `store prob density` A boolean that indicates whether or not the probability density field is to be stored alongside the eigenvectors. The number of fields stored is given by `store_evecs`.

The `define()` method also attaches submenus accessing parameters for linear solvers and matrix storage schemes. From the menu system, the submenu controlling matrices and linear solvers are accessed with the command '`sub LinEqAdm`' or '`sub L`'. Two submenus `Matrix_prm` and `LinEqSolver_prm` can be accessed from here, making available settings for matrix storage and solver methods, respectively. The most important commands are '`set matrix type=`*`classname`*' and '`set basic method=`*`method`*', selecting matrix storage scheme and solver method, respectively.

`scan()`. This method initializes the `HydroEigen` solver class according to the settings from the menu system. It allocates memory for various arrays and objects, creates (or reads from file) the grid, opens a log file to which various statistics and results is written and so on. In short, an instance of the class is ready to solve an eigenvalue problem when `scan()` is finished.

`solveProblem()`. This method is the most central function in the program. It creates the matrix A (called `K` in the code) and the matrix M, imposes boundary conditions, instantiates `EigenSolver` and solves the eigenvalue problem. The eigenvalues and eigenvectors are then written to file by `HydroEigen::report()`.

`enforceHomogenousBCs()`. Boundary conditions are implemented differently in eigenvalue problems than in regular finite element simulators. The problem defining matrices have to be modified by erasing rows and columns corresponding to nodes on the boundary of the grid. This member function accomplishes this with both sparse matrices and banded matrices. It uses other member functions such as `eraseRowAndCol()` to do this.

`getElapsedTime()` and `reportElapsedTime()`. These functions perform simple timing tasks. `getElapsedTime()` simply returns the number of seconds since the first call to this function. `reportElapsedTime()` writes this to the log file as well.

`report()`. After `solveProblem()` has finished its main task, `report()` is called to write the eigenvectors (i.e., the discrete eigenfunctions) to a file. In addition, the eigenvalues and various other information is written to the log file. This information may be used in visualization programs to show the approximate eigenfunctions found by the program. Each eigenvector is labeled with its eigenvalue for reference.

`integrands()`. This method is perhaps *the* core of every finite element formulation. It evaluates the integrand in the numerical integration routines used in the element matrix assembly. Integrals in Diffpack are exclusively done numerically with Gauss integration of varying order. The `integrands()` method evaluates the integrand (multiplied with the integration weight and the Jacobian of the coordinate change mapping between local and global coordinates) at a given point (passed in a `class FiniteElement` parameter object) and updates the element matrix and vector also passed as parameters.

`calcExpectation_r()`. It would be useful if the simulator application could produce information on expectation values of various observables. Given two `FieldFE` objects

corresponding to two discrete functions u and v, this method calculates (u, rv). The method is implemented by integrating numerically over the elements in the grid similar to the assembly of the element matrix A. Even though only this and one other such method is implemented here (namely `calcInnerProd()`), it is easy to write new ones. In fact, the time dependent solver implements a more general expectation value method using a general matrix. This allows for computation of for example the total energy without integration over the elements as this is already done in the assembly process.

6.5.2 Comments

Some comments are given here, summing up some information gained during implementation, simulation and in the aftermath after having implemented the time dependent solver (in chapter 7).

The Matrices. The matrices stored in the handles `M` and `K` correspond to the mass matrix M and the matrix H, respectively. The matrices may be created in either `MatBand` or `MatSparse` format, the latter definitely being the most favorable as the matrices become increasingly sparse with increasing dimension of the problem. Banded matrices take up a lot more space and require much more operations during for example matrix-vector products. Furthermore, if a sparse matrix is used there is no need for optimizing the grid. In Fig. 5.1 this optimization is done. Without optimization the non-zeroes are scattered throughout the whole matrix, making a banded storage scheme not at all attractive.

Other matrix formats should not be used as their support is not implemented in for example `enforceHomogenousBCs()`.

The mass matrix can be lumped or not lumped. In the former case the full matrix is not used in the diagonalization process, but instead A (i.e., `K`) is multiplied with its inverse so that the eigenvalue problem becomes a standard problem. This speeds up the simulation considerably.

Linear Solvers. The linear solver method is chosen by the following commands:

```
sub LinEqAdm
sub LinEqSolver_prm
set basic method = method
```

The name *`method`* can be one of many choices, e.g., `GaussElim`, `GMRES` and `ConjGrad`. With complex sparse matrices Gauss elimination cannot be used because it is not implemented in Diffpack. The conjugate gradient method `ConjGrad` supposedly works best for positive definite systems, but it seemingly works well for other systems as well. The method used in the simulations in this thesis was `GMRES`, however. The method takes longer than Gaussian elimination for small systems, bot when the number of nodes increases, the method becomes much faster. Optimization of the grid does not seem to affect the efficiency. (This is intuitively so because only matrix-vector operations are used in the algorithm.)

The Invocation of ARPACK. The `class EigenSolver` instance used for diagonalization has support for many options that are not used at all. For example one might want to search for eigenvalues of intermediate magnitude instead of the lowest magnitudes. If one wishes to study for example the structure of highly excited states in a classically chaotic system, this may be the case. It is only minor modifications that is needed to incorporate this in the class.

`EigenSolver` is also somewhat limited with respect to what features of ARPACK is actually used, such as monitoring of convergence, reading the Schur vectors and so

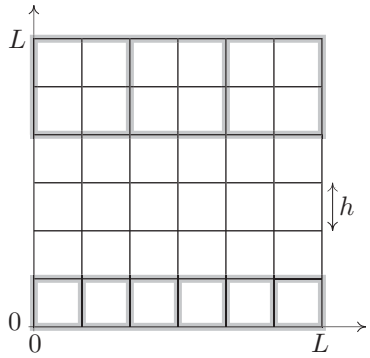

Figure 6.3: A square grid with $D = 6$. A few linear and quadratic element shapes have been drawn to indicate their distribution.

on. In a more complex implementation these features could be taken advantage of in `HydroEigen` as well.

6.6 Numerical Experiments

The `HydroEigen` class may be used to solve the particle-in-box, the harmonic oscillator and the hydrogen atom. In this section we will go through some numerical experiments with the program. There are several parameters that may affect the precision of the eigenvalues, such as mesh width, element type and order and the numerical geometry. In addition, the presence of the mass matrix M in the problem may or may not improve the quality of a finite element approximation in comparison with a finite difference approximation.

6.6.1 Particle-In-Box

Square Domain

When studying the particle-in-box in a square domain, the most natural choice for grid is a uniform square grid. The analytic eigenfunctions have the form $\sin(k_x \pi x)\sin(k_y \pi y)$ and do not concentrate in some regions of the square. Hence, the grid should be uniform. Fig. 6.3 shows a square grid with sides L. Each side is subdivided into D intervals of length $h = L/D$, where h is the mesh width, and the grid has $(D+1)^2$ nodes. When using linear elements, each elements requires $2 \times 2 = 4$ nodes, and when using quadratic elements $3 \times 3 = 9$ nodes are required. If we then choose D as an even number we may use both linear and quadratic elements on the same subdivision and hence the same mesh width.

To study the quality of the finite element method (and the finite difference method in this case) we study the relative error of the eigenvalues as function of h and the element order p, i.e.,

$$\delta(h,p) = \frac{\lambda^{\text{num}}}{\lambda} - 1.$$

The qualitative behavior of the relative error, e.g., if it is increasing slowly or rapidly will also indicate the quality of the methods.

Furthermore, comparing a simulation where the mass matrix is lumped and a simulation with the full mass matrix may indicate if lumping improves the quality of the eigenvalues.

This particular problem of a particle-in-box was solved analytically in one dimension and also partially in two dimensions in section 6.4. These results should be set in

		n			
		20	40	60	∞
k	1	19.77982922	19.74935767	19.74371889	19.73920881
	2,3	49.69408652	49.43433728	49.38636767	49.34802202
	4	79.60834382	79.11931689	79.02901644	78.95683523
	5	100.3720148	99.1128192	98.88109094	98.69604404

Table 6.2: Results from numerical simulations of particle-in-box with linear elements

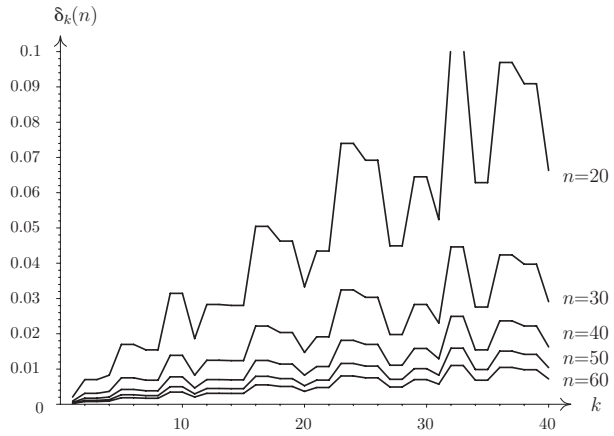

Figure 6.4: Relative error of eigenvalues, linear elements

conjunction with the present discussion.

Linear Elements. A sequence of simulations was run with $n = 20$, 30, 40, 50 and 60 subdivisions of the sides in the grid. Table 6.2 shows the first few numerical eigenvalues compared to the analytic eigenvalues for some grid sizes. Fig. 6.4 shows the relative error for each simulation. Note that many eigenvalues are pairwise equal, also in the discrete formulation, and this shows up as many pairwise equal relative errors.

If we assume that $\delta(h) \sim Ch^\nu = Cn^{-\nu}$ we obtain

$$\ln \delta = C - \nu(\ln n),$$

i.e., a straight line if we plot $\ln \delta$ against $\ln n$. The constant should depend on the eigenvalue number k. Fig. 6.5 shows such plots for a few eigenvalues λ_k^{num}. As seen from the figure, we obtain perfectly linear plots; hence the assumed form of δ fits well. Notice that increasing eigenvalues yield increasing Cs, reflecting that higher eigenvalues tend to have higher error, as seen in Fig. 6.4 as well. By inspection, $\nu = -2$ for all the sample plots. Hence,

$$\lambda^{\text{num}} = \lambda + \mathcal{O}(h^2)$$

is a reasonable guess.

Quadratic Elements. Performing the same experiments but with quadratic elements yields very similar results. Simulations with $n = 20$, 30, 40 and 50 were done. Fig. 6.6 shows the relative error for each eigenvalue when using quadratic elements, while Fig. 6.7 shows $\ln \delta$ as function of $\ln n$. Clearly,

$$\delta = \mathcal{O}(h^4),$$

which is a much better convergence rate than for linear elements.

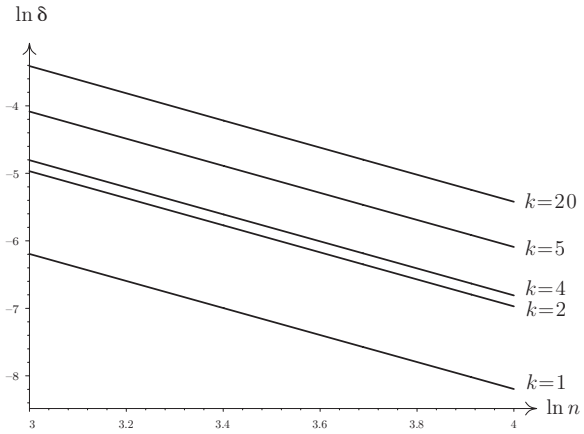

Figure 6.5: $\ln|\delta|$ versus $\ln n$. The graphs are straight lines with slope -2.

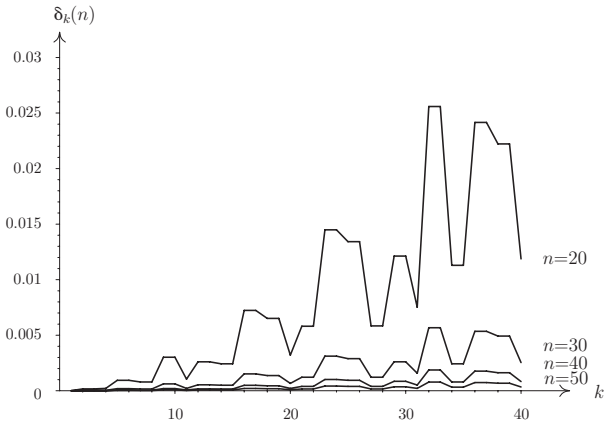

Figure 6.6: Relative error of eigenvalues, quadratic elements

The *same* number of nodes and hence the same dimensions of the matrices yields *much faster* convergence with quadratic elements. What we have to pay is an increase in the bandwidth of the matrices, as quadratic elements contain more nodes, and hence couplings between nodes farther away from each other than in linear elements.

We notice that $\delta > 0$ for all performed experiments, i.e., the finite element method seems so overestimate the eigenvalues.

Lumping the Mass Matrix. It is a well-known fact that when solving the one-dimensional wave equation $u_{tt} = u_{xx}$ the standard finite difference method $\delta_t\delta_t u = \delta_x\delta_x u$ yields the exact solution. This difference scheme is actually equivalent to using linear elements and lumping the mass matrix; hence an improvement of the numerical results is a consequence of lumping in this case.

Lumping the mass matrix when using linear elements creates an eigenvalue problem equivalent to using the finite difference method, as shown in section 6.4. Furthermore, lumping the mass matrix makes our eigenvalue problem a standard eigenvalue problem which is easier and quicker to solve numerically as we do not need to solve linear systems of equations. Hence, it is of importance whether or not lumping improves or degrades the eigenvalues and eigenvectors.

A few numerical experiments shows that $\delta = \mathcal{O}(h^4)$ also for the lumped eigenvalue problem with quadratic elements, see Fig. 6.7. In addition, $\delta < 0$, i.e., the lumped system tends to underestimate the eigenvalues.

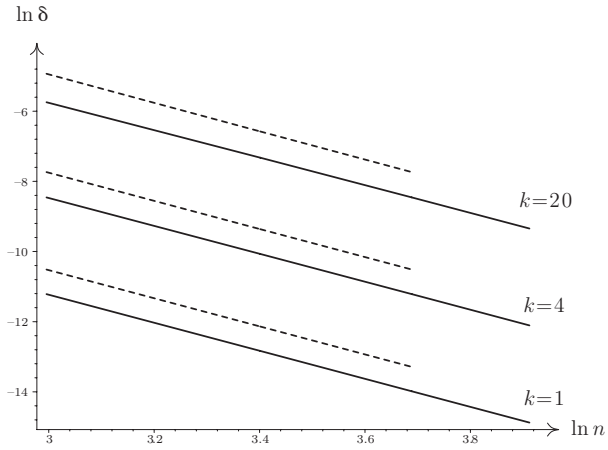

Figure 6.7: $\ln|\delta|$ versus $\ln n$. The graphs are straight lines with slope -4. The dashed lines are for the lumped eigenvalue problem.

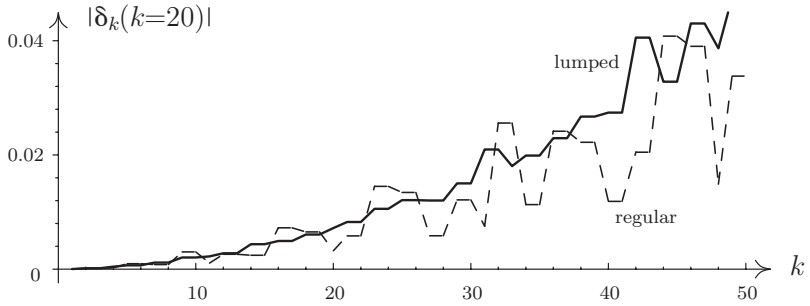

Figure 6.8: Comparison of the relative error $|\delta|$ in the eigenvalues for lumped system and regular system. A grid with $n = 20$ was used.

.

The qualitative behavior of the relative error is however different in the lumped system when compared to the regular eigenvalue problem. The regular problem has a relative error that fluctuates a great deal with increasing eigenvalue number. The lumped version shows a smoother behavior, see Fig. 6.8 in which a comparison is made. As we see, the relative error grows faster with increasing eigenvalue number than in the original problem. This effect can also be seen in Fig. 6.2, section 6.4.

As we see, lumping might be an option when only the lowest eigenvalues are required. Furthermore, with a lumped mass matrix the eigenvalues are in principle much easier to compute, an important fact. As the behavior of the eigenvalues are qualitatively very different when the mass matrix is lumped, it should not come as a surprise if the method is superior for some Hamiltonians. Indeed, when we study the two-dimensional hydrogen atom we see that lumping the mass matrix drastically improves the behavior of the method.

Circular Domain

When discretizing a circular domain we use triangulation. This is supported by Diffpack, see Ref. [34]. Parameters to the circular grid are its radius r, the number of line segments on the boundary and the total number of desired elements. The only element type supported is linear triangular elements. As the node locations and element shapes are not known *a priori* we will have to estimate the mesh width h. If we by $\tilde{A}$ denote

the average area of the elements, we have $\tilde{A} \propto h^2$, i.e.,

$$\tilde{A} = \frac{\pi r^2}{n_{\text{el}}} \propto h^2,$$

where n_{el} is the number of elements. Thus,

$$h \propto \frac{r}{\sqrt{n_{\text{el}}}}.$$

In section 3.5 of Ref. [34] the suggested number of elements to produce a uniform grid with optimal triangle angles are

$$n_{\text{el}} = \frac{(n-1)^2}{2\pi},$$

where n is the number of line segments used to discretize the border of the circular region. (In the grid generation utility, n_{el} must be supplied, but it is treated as a "wish,", i.e., the program will try to create this number of elements.) Using the formula for n_{el}, we obtain

$$h \propto \frac{r}{n-1},$$

hence using evenly spaced n will produce meshes comparable to the ones used for the square domain. We will use $n = 50$, 70, 90, 110 and 130. Fig. 6.12 shows two grids used in the simulation.

We will do numerical simulations similar to the square domain case. We will do a comparison between the lumped system and the regular system in addition to finding the relative error order.

Fig. 6.9 shows the relative error $\delta(k)$ for the different grid sizes. Fig. 6.10 shows the same for the lumped system. The relative error is negative for the lumped system, hence, eigenvalues are underestimated in this case. For the regular system, eigenvalues are exclusively overestimated.

Fig. 6.11 shows $\ln|\delta|$ as function of $\ln n$ for selected eigenvalues for both the regular and lumped system. Clearly, $\nu = -2$ in the case of linear, triangular elements (both lumped and regular) similar to the rectangular domain case. I.e.,

$$\lambda^{\text{num}} = \lambda + \mathcal{O}(h^2).$$

The irregularity of the curve shapes are probably due to the definition of h. The grids are not related in a simple way as with the rectangular grid used above.

Clearly, the lumped system performs *better* than the regular for this system as the relative error is lower for all eigenvalues.

6.6.2 Two-Dimensional Hydrogen Atom

The Hamiltonian for the two-dimensional hydrogen atom with an applied magnetic field in polar coordinates is

$$H = -\nabla^2 - \frac{2}{r} - i\gamma\frac{\partial}{\partial\phi} + \frac{\gamma^2}{4}r^2,$$

where we have used the symmetric gauge in which the magnetic vector potential is given by

$$A = \frac{\gamma}{2}(-y, x, 0).$$

The parameter γ is the strength of the magnetic field which is aligned along the z-axis perpendicular to the plane of motion of the electron. The time independent Schrödinger equation reads

$$H\Psi_{nm} = \epsilon_{nm}\Psi_{nm}$$

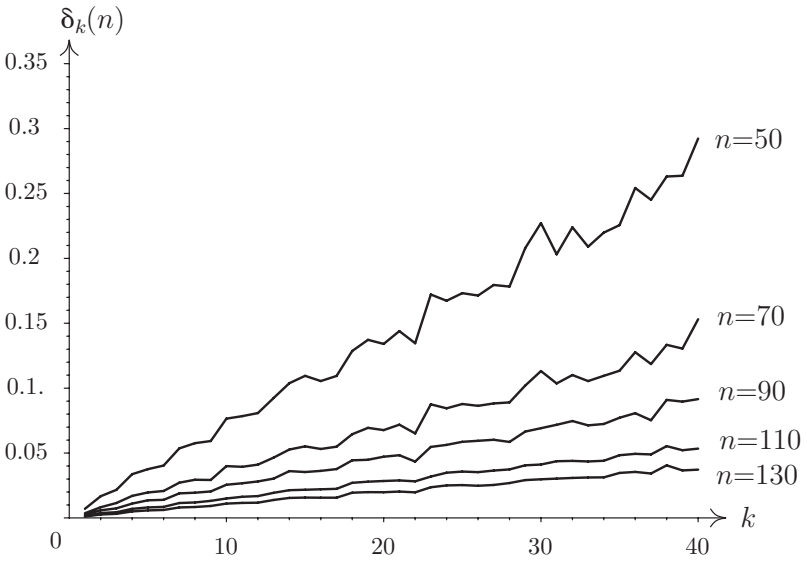

Figure 6.9: Relative error for each eigenvalue for the different grids in the regular system.

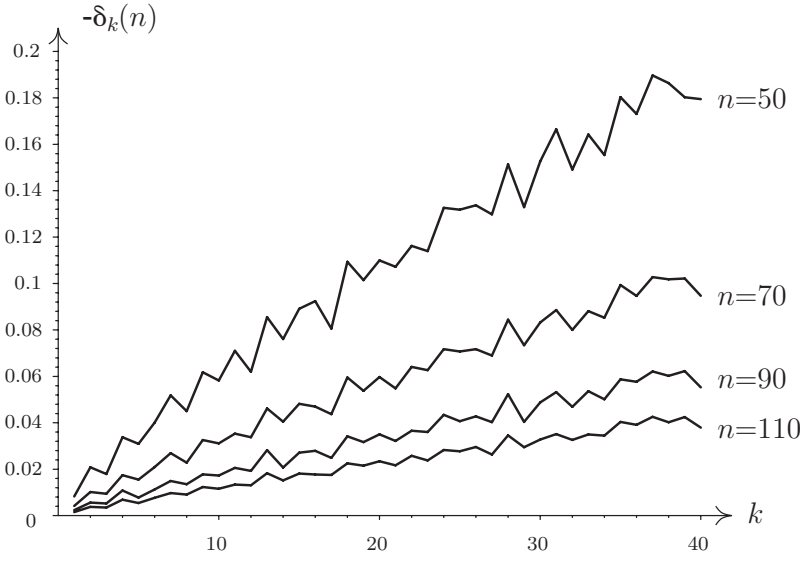

Figure 6.10: Relative error for each eigenvalue for the different grids in the lumped system.

.

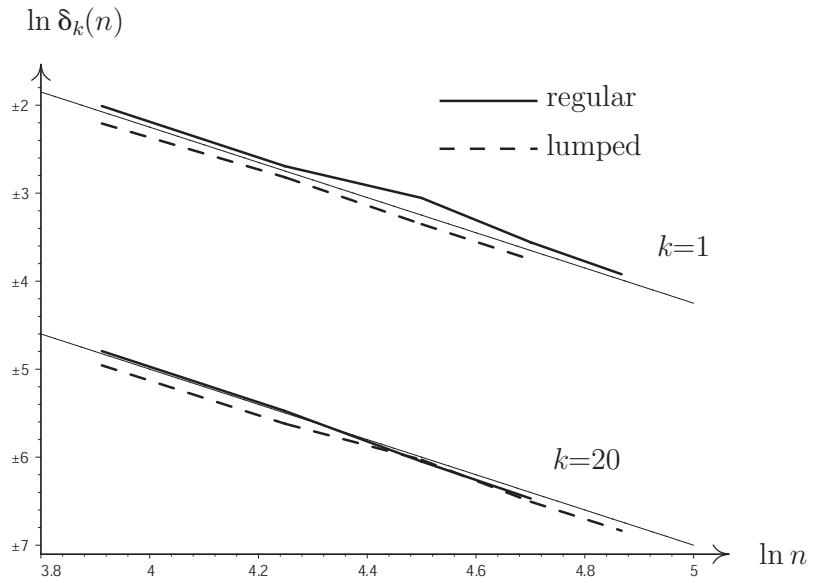

Figure 6.11: $\ln|\delta|$ as function of $\ln n$ for a few eigenvalues λ_k^{num}. Reference lines with slope -2 are shown.

.

where the eigenvalues are given by

$$\epsilon_{nm} = -\frac{1}{(n+1/2)^2}, \quad n = 1, 2, \ldots, \quad m = -n+1, -n+2, \ldots, n-2, n-1.$$

We see that each eigenvalue has degeneracy $2n-1$. The corresponding eigenfunctions are obtained by separation of variables, viz.,

$$\Psi_{nm} = e^{im\phi} R_{nm}(r),$$

where $R_{nm}(r)$ fulfills a differential equation we cannot solve in closed form. The numerical values of the first few eigenvalues are

$$\epsilon_{10} = -4,\ \epsilon_{2m} = -\frac{4}{9} \approx -0.4444,\ \epsilon_{3m} = -\frac{4}{25} = -0.16,\ \epsilon_{4m} = -\frac{4}{49} \approx -0.08163.$$

Table 6.3 shows the standard deviation $\sigma_{nm} = \sqrt{\langle r^2 \rangle}$, quoted from Ref. [29]. The width of the wave function tends to grow rapidly with higher energy. Our domain of discretization must be large enough in order to capture the essential features of Ψ_{nm} for as many states as we wish. On the other hand we have a singularity at $r = 0$ and the potential varies rapidly here. In order to reproduce the Hamiltonian faithfully near the origin we must have an appropriately fine mesh. As $\sigma_{00} = \sqrt{3/8}$ is a small number it is not surprising if the region $r < 1$ needs a quite fine mesh.

Numerically we do not use the natural numbering of the eigenvalues, simply because we do not know *a priori* if they are degenerate or not. We simply choose to sort them in increasing order and we label them with an integer k. The eigenvectors found for the finite element matrix corresponds to an approximate eigenfunction through the mapping

$$u_h = \sum_j U_j N_j(\boldsymbol{x}),$$

and therefore we will use the term eigenvector and eigenfunction interchangeably in this section.

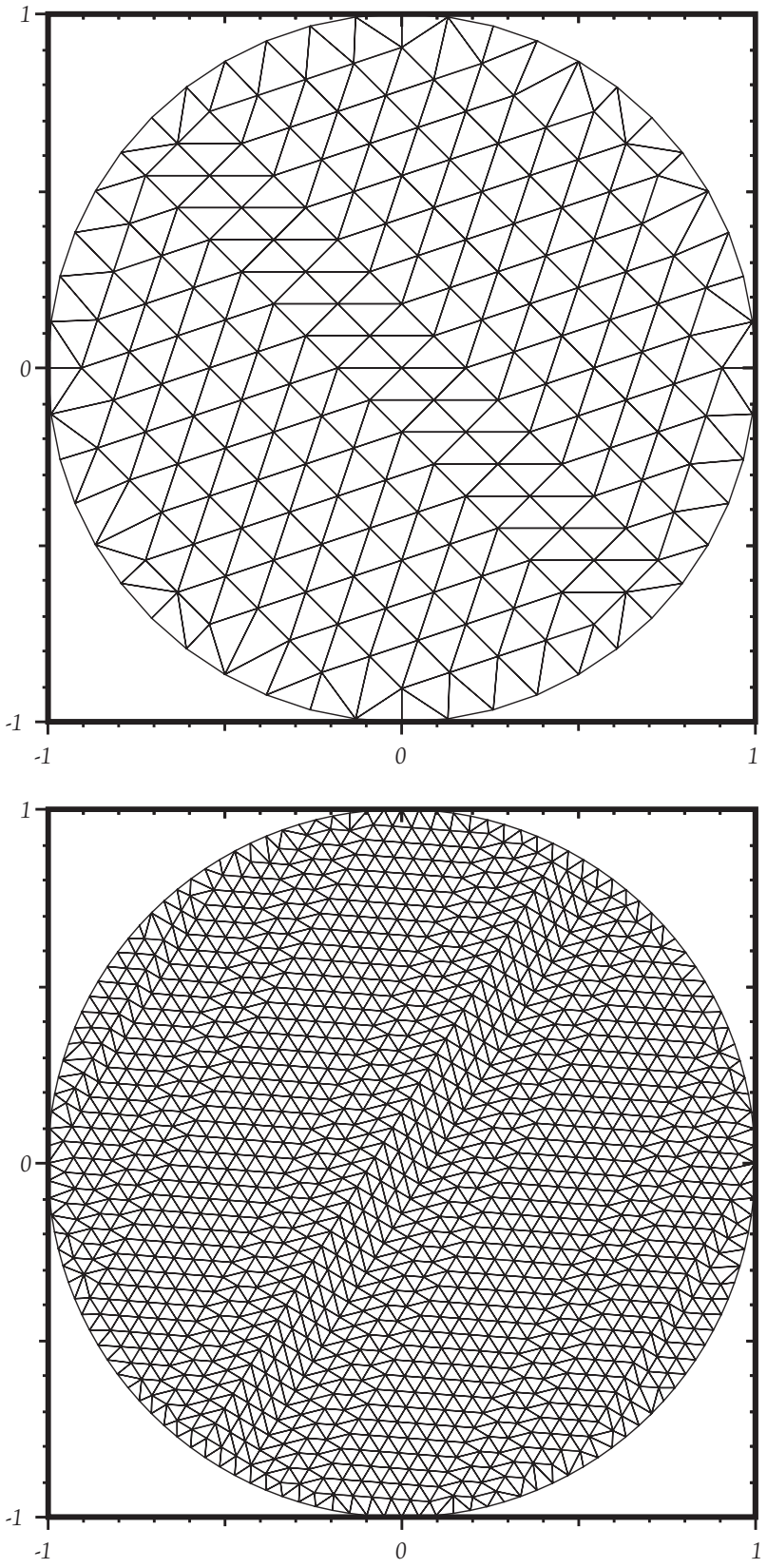

Figure 6.12: Grids for $n = 50$ (above) and $n = 130$ (below).

.

	$\|m\| = 0$	$\|m\| = 1$	$\|m\| = 2$	$\|m\| = 3$
$n = 1$	$\sqrt{\frac{3}{8}} \approx 0.612$			
$n = 2$	$\sqrt{14\frac{5}{8}} \approx 3.825$	$\sqrt{11\frac{1}{4}} \approx 3.354$		
$n = 3$	$\sqrt{103\frac{1}{8}} \approx 10.155$	$\sqrt{93\frac{3}{4}} \approx 9.682$	$\sqrt{65\frac{5}{8}} \approx 8.101$	
$n = 4$	$\sqrt{385\frac{7}{8}} \approx 19.644$	$\sqrt{367\frac{1}{2}} \approx 19.170$	$\sqrt{312\frac{3}{8}} \approx 17.764$	$\sqrt{220\frac{1}{2}} \approx 14.849$

Table 6.3: Standard deviation $\sqrt{\langle r^2 \rangle}$ for different eigenstates.

The simulations for the two-dimensional hydrogen atom is very similar to the particle-in-box-simulations. However, in this system we do not have a bounded geometry, hence we must truncate it. The obvious first-choice is to create a rotationally symmetric domain, i.e., a disk with radius r_0. However, this limits the element type to linear triangles due to constraints in Diffpack's grid preprocessors.

We will use triangulated disk grids for our simulations in this section. The grid generation utility from the particle-in-box simulations had to be replaced, due to apparent instabilities of the algorithm. Nevertheless, the grids are parameterized by the radius r_0 and the typical element size h. To sum up, we have

$$\begin{aligned} n_{\text{el}} &= \frac{1}{2\pi}(n-1)^2 \\ \tilde{A} &= \frac{\pi r^2}{n_{\text{el}}} = h^2, \end{aligned}$$

where n is the number of line segments used to approximate the boundary of the disk.

To keep the number of elements fixed but increase the accuracy near the origin we may introduce a *grading* to our grid. The grading is a transformation of the nodes in the grid given by

$$\boldsymbol{x}^{[k]} \mapsto \left(\frac{\|\boldsymbol{x}^{[k]}\|}{r_0} \right)^{w-1} \boldsymbol{x}^{[k]}.$$

The effect of this mapping is to change the length of each point $\boldsymbol{x}$ into $\|\boldsymbol{x}\|^w$ and then rescale such that the size r_0 of the grid is conserved. With $w > 1$ this will create a grid with smaller triangles near the origin. The parameter must be chosen with some care, however, as a value which is too large will stretch some triangles and produce undesiredly small angles. (This may introduce round-off errors in the computations.) Notice that grading with this procedure keeps h fixed as the average element area is preserved.[7]

Importance of grading. To illustrate the importance of the mesh width near the origin we present simulations with $r_0 = 20$, $h = 0.2$ and with different grading parameters, viz.,

$$w \in \{1.0, 1.5, 1.8, 2.0, 2.5\}.$$

The grid had $n_{\text{no}} = 1275$ nodes (of which 1174 remained after incorporating boundary conditions) and $n_{\text{el}} = 2447$ triangular elements. Fig. 6.13 shows the first few eigenvalues for each simulation compared to the analytic eigenvalues. We have used a lumped mass matrix in each simulation. It is worthwhile to mention that the simulation time increased with increasing w, even though the dimension of the system was the same in each simulation. The simulation with $w = 2.0$ took 13 times as long time to finish as the homogenous grid, i.e., $w = 1.0$, while $w = 1.8$ took 6.5 times as long.

[7] More sophisticated methods for grading and/or refining the grid can be used as well, such as specifying each element's desired in a second pass of grid generation. This is out of scope for this thesis. See for example Ref. [46].

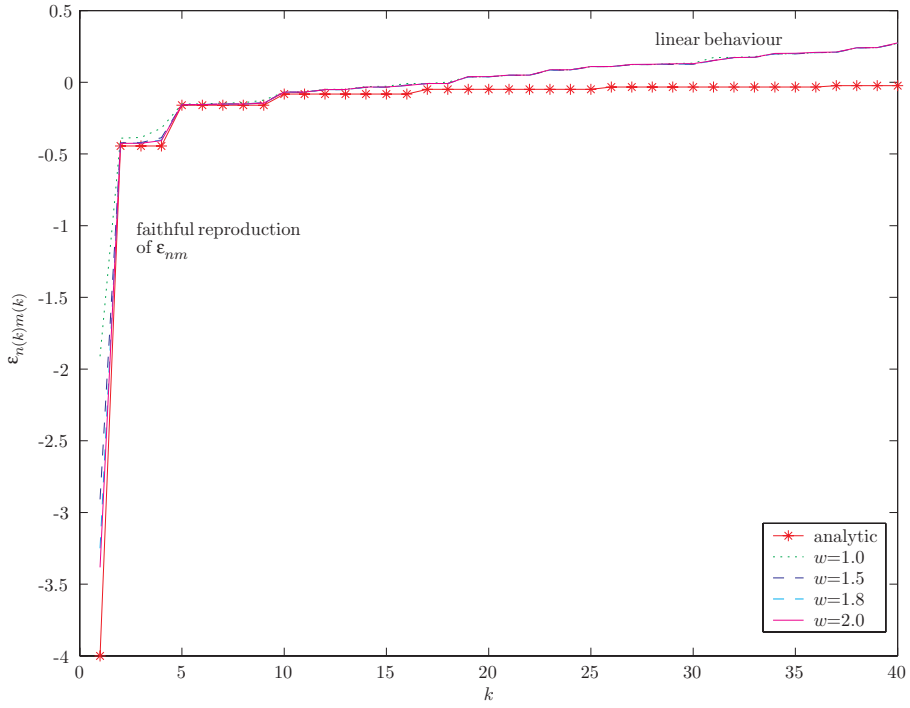

Figure 6.13: Eigenvalues for grids with different grading factors w. Parameters for grid: $r_0 = 20$, $h = 0.2$.

The Numerical Eigenfunctions. Fig. 6.13 shows the numerical eigenvalues together with the analytic eigenvalues. Clearly, up to a certain k, the qualitative behavior is reproduced. At some point the numerical eigenvalues seems to grow linearly instead of the expected $1/n(k)^2$ behavior. Intuitively, the eigenfunctions of such high states must analytically go beyond the limits of the disk, i.e., $\sigma > r_0$, and this is the reason for the wrong behavior of the eigenvalues. For high energies the particle-in-box features of the effective potential (i.e., Coulomb attraction plus infinite box potential) dominate.

Fig. 6.14 shows plots of a few eigenstates (the real parts) for the grid with parameters $(r = 120, h = 2)$. By looking at the imaginary parts (not shown here) they are easily seen to be proportional to the real parts, hence the eigenfunctions found by ARPACK can be written as $e^{i\alpha}\Psi(\boldsymbol{x})$, with $\alpha \in \mathbb{R}$ and Ψ a real function.

When solving the eigenvalue problem analytically one typically uses separation of variables, i.e., $\Psi_{nm} = e^{im\phi}R_{nm}(r)$. The radial function $R(r)$ is a solution of Eqn. (6.3), and it depends on *both* the quantum number n and the magnitude $|m|$. For a given ϵ_n we have an eigenspace of dimension $2n-1$ for which we may find an orthonormal basis, and indeed the functions $e^{im\phi}R_{nm}(r)$ are orthogonal. The IRAM also finds orthogonal eigenvectors, but this time discrete eigenvectors. Incidentally, it finds something very similar to what we find when using separation of variables in this case. Consider the three eigenfunctions that clearly correspond to $\epsilon_2 \approx -0.4444$. We see that the first one is approximately independent of ϕ, hence it corresponds to Ψ_{20}. The two next are clearly a radial function modulated with a periodic function approximately equal to $\cos\phi$ and $\sin\phi$. We obtain these by taking linear combinations of $e^{i\phi}$ and $e^{-i\phi}$, viz.,

$$2\cos\phi R(r) = (e^{i\phi} + e^{-i\phi})R(r), \quad 2i\sin\phi = (e^{i\phi} - e^{-i\phi})R(r).$$

In other words,

$$\Psi_3^{\text{num}} \propto \Psi_{2,-1} + \Psi_{2,1}$$

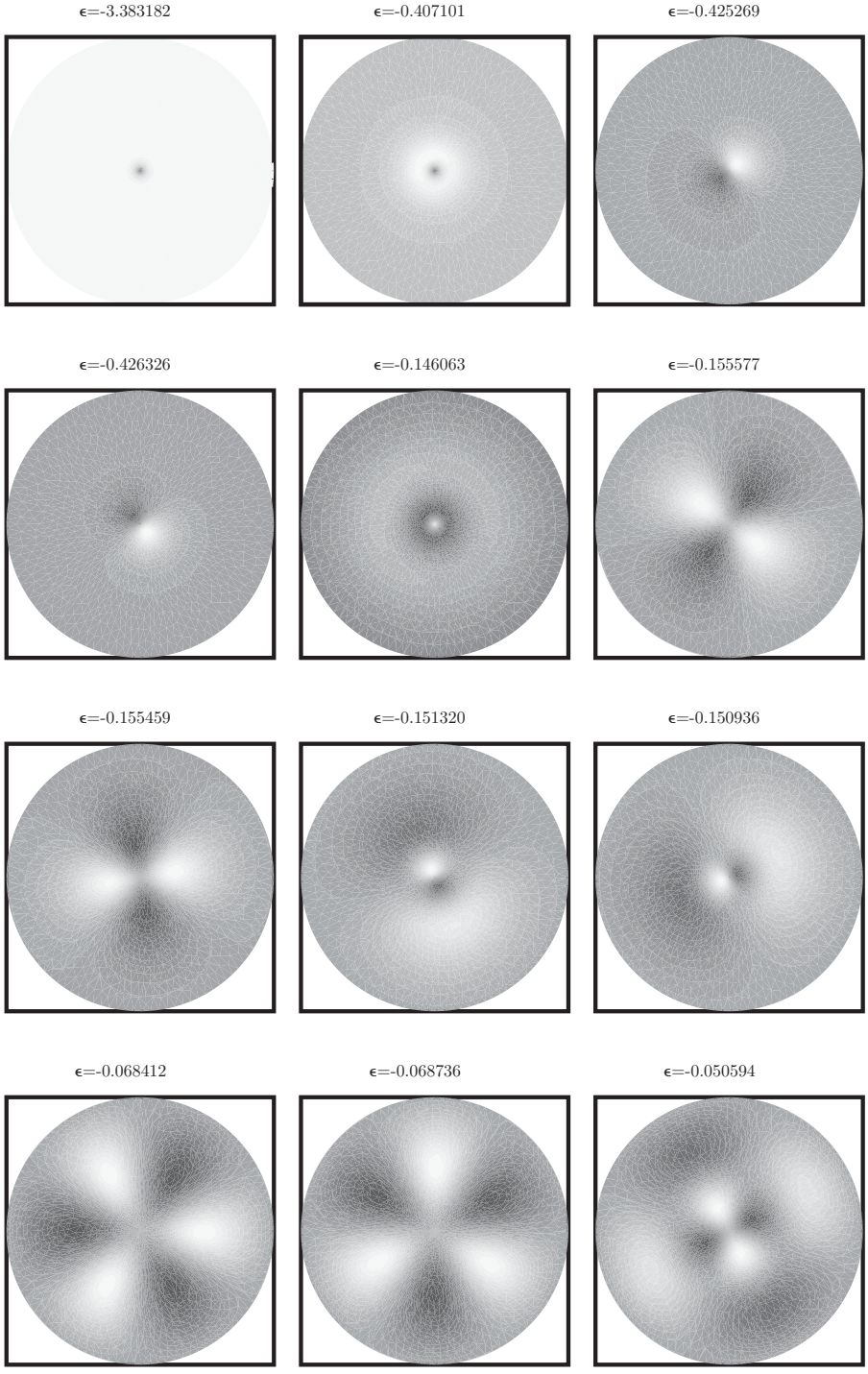

Figure 6.14: Eigenstates found with $r_0 = 20$, $h = 0.2$, $w = 2.0$. Gray tones indicate low values (white) and high values (black). Lowest-lying states are upper left.

and

$$\Psi_4^{\text{num}} \propto \Psi_{2,-1} - \Psi_{2,1}.$$

In the same manner we proceed with the five graphs with $\epsilon \approx -0.15$, i.e., they corresponds to $n = 3$. Again, one of these is independent of ϕ and corresponds to $m = 0$. Two functions are proportional to $\sin\phi$ and $\cos\phi$, and the last two are proportional to $\cos 2\phi$ and $\sin 2\phi$, corresponding to $m = 2$. In addition we see that the radial functions R_{30}, R_{31} and R_{32} are different, as expected from the radial equation's dependence on $|m|$.

We must emphasize that this structure of the eigenfunctions are partially a coincidence, probably due to the approximate rotational symmetry of the grid. There is no other reason why ARPACK should not have chosen a different orthogonal basis for the eigenspace corresponding to ϵ_n. Furthermore, the eigenvalues are only approximately degenerate in the numerical case.

Increasing the Grid Size. It was mentioned that the eigenvalues tend to grow linearly when $k > 10$. We can find an explanation in terms of the eigenfunction plots. For the states with the lowest eigenvalues the dominating features are near the origin. For higher-lying states we see that the features approach the rim of the disk. Therefore the particle-in-box part of the numerical model starts to show up in both the eigenvalues and eigenfunctions. Already for $n = 4$ we have $\sigma_{nm} \sim r_0$ and hence we are losing features of the functions.

To study this behavior in more detail simulations with increasing disk sizes were carried out. The grids had radii given by

$$r_0 \in \{20, 30, 40, 50\},$$

and they all had $h = 0.2$ and $w = 1.5$. As seen in Fig. 6.16 the eigenvalues get progressively better as we increase the grid size, *also for the lowest-lying states*. Physically, this corresponds to the slow dying out of the Coulomb potential. Fig. 6.15 shows the 10th eigenstate for $r = 50$. If we compare this to the corresponding plot in Fig. 6.14 we see that more of the features are captured by this large grid.

6.7 Strong Field Limit

We have studied the eigenvalue problem for the weak-field limit, i.e., $\gamma = 0$. We now turn to the limit in which the Coulomb interaction is treated as a perturbation, i.e., we turn it off and set $\gamma = 1$ in order to study the numerical properties of the eigenvalue problem.

If we find a common grid in which both the weak-field limit (i.e., the pure Coulomb system) and the strong-field limit (i.e., the pure magnetic system with $\gamma = 1$) yield results that agree with analytic results, it is reasonable to believe that also the combined system (i.e., Coulomb attraction *and* magnetic field) will be solved with accuracy.

Our goal in this thesis is not to achieve high-precision results in this respect. In order to achieve this we must perform simulations that are too heavy for the resources available at this stage in the work. Instead we aim at a good understanding of the behavior of the numerical problem for different grid parameters and physical parameters. When more resources are available we can solve the eigenvalue problem with higher accuracy.

The Hamiltonian for the pure magnetic system reads

$$H = (-i\nabla + \boldsymbol{A}_{\text{symm}})^2 = -\nabla^2 - i\gamma\frac{\partial}{\partial\phi} + \frac{\gamma^2}{4}r^2.$$

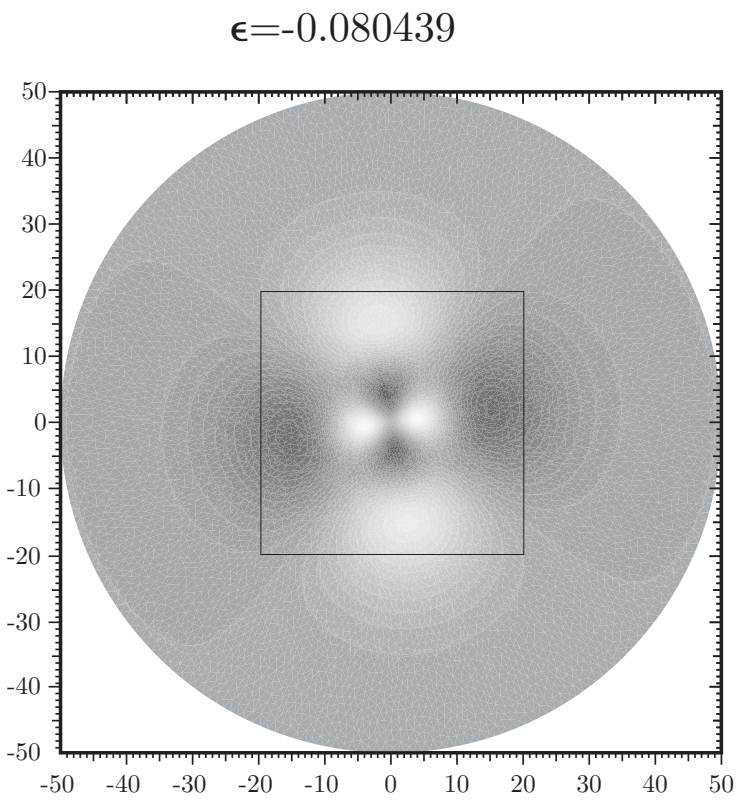

Figure 6.15: Sample eigenstate of a simulation with $r_0 = 50$, $h = 0.2$, $w = 1.5$. Grid size from Fig. 6.14 shown for reference.

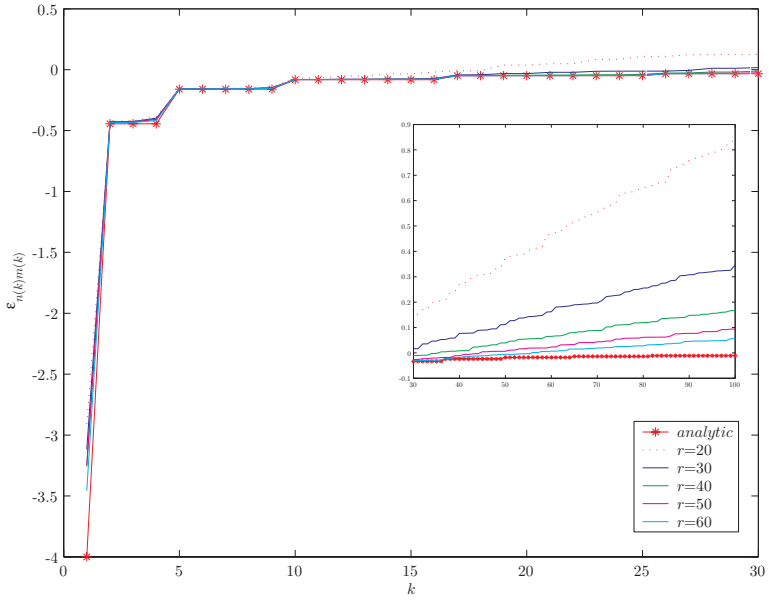

Figure 6.16: The eigenvalues for successively larger grids. Here, $h = 0.2$, $w = 1.5$. The inset graph shows the continuation of the larger.

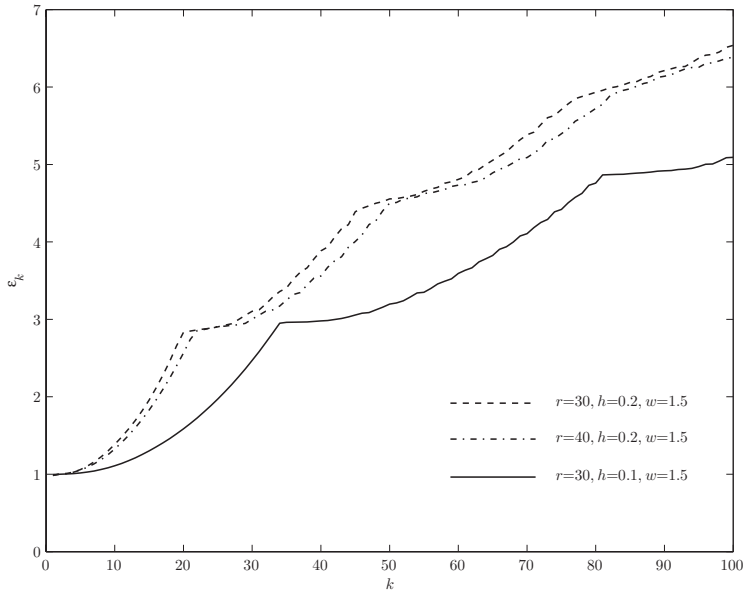

Figure 6.17: Eigenvalues for thee simulations of the magnetic Hamiltonian

The eigenvalues (called the Landau levels) are infinitely degenerate if the domain is $\mathbb{R}^3$. The symmetric gauge is employed here. Recall that any other gauge will give different eigenfunctions but the same eigenvalues. Recall that the degeneracy of each landau level if the domain is a disk with radius r_0 such as in our simulations is

$$g = \frac{r_0^2}{2}. \tag{6.11}$$

There are two important facts to be aware of. First, we do not know which basis of eigenvectors our numerical method will choose for us. (As we shall see it is a quite interesting basis.) Second, we do not know if the numerical methods are indeed gauge-invariant such as the original problem. If we diagonalize in a different gauge, will we find the same eigenvalues? We will exclusively use the symmetric gauge in our simulations.

First we consider two simulations with

$$r_0 = 30, \quad h = 0.2, \quad w = 1.0 \text{ and } 1.5,$$

and compare the results. The eigenvalues are shown in Fig. 6.17. Clearly, the graded grid yielded better eigenvalues than without grading. Next, we do a simulation with

$$r_0 = 30, \quad h = 0.1, \quad w = 1.5, \tag{6.12}$$

i.e., we reduce the mesh width. The eigenvalues are shown in Fig. 6.17. Clearly, reducing the mesh width has a dramatic impact on the approximate degeneracy of the lowest Landau level. We also see the effect on the energy levels that correspond to $2N + 1$.

Analytically, the eigenvalue $\epsilon = 1$ is multiply degenerate. Numerically, the eigenvalues starts out close to 1 but rapidly increase. Clearly, a smaller mesh width enhances the convergence of the lowest eigenvalues. It is intuitively clear that all $g = r_0^2/2$ states for every Landau level will not be found with a finite mesh width, as the number of states equals the number of internal grid points in the mesh which is perhaps much less than g to start with. It also becomes clear that decreasing the mesh width should increase the approximate degeneracy of the lowest Landau level in particular.

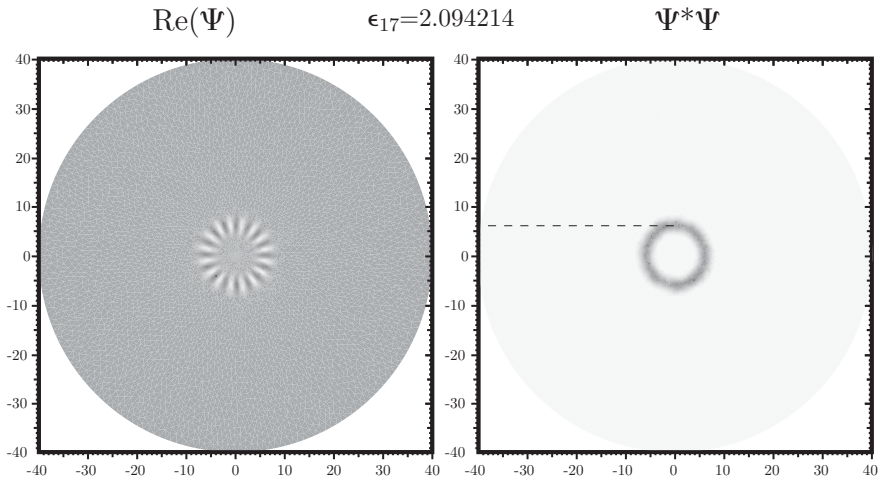

Figure 6.18: The 17th eigenfunction of the Landau Hamiltonian in the symmetric gauge. Light colors are low values and black colors are high values.

We notice that large fractions of the numerical eigenvalues tend to cluster around the analytical eigenvalues $2N+1$, creating a (crude) approximation to a ladder function. In fact, as $h \to 0$ one expects that the function ϵ_k converges to

$$\epsilon(k) = 2\lfloor k/g \rfloor + 1,$$

representing the exact eigenvalues although we have no proof of this.

Recall that for the hydrogen atom ARPACK tended to find eigenfunctions that were closely related to the eigenfunctions found analytically by separation of variables. This is also the case in the Landau system and in a both interesting and amusing way. Fig. 6.18 shows the state Ψ_{16} from the simulation with $h = 0.1$. It belongs to the lowest Landau level as seen from Fig. 6.17. First of all the quantum number $m = 16$ is readily identified if we assume $\Psi_{0M} = e^{im\phi}R_{0M}(r)$. Second, notice the localization of the probability density around $r_{16} = \sqrt{2 \cdot 16} \approx 5.6$.

This state is qualitatively representable for the eigenfunctions found for the lowest Landau level. The angular quantum number m is readily identified and the probability is concentrated in rings. As the momentum is proportional to the gradient of Ψ, it is easily seen that in this case it is tangent to the ring at the centre. (At the edges of the ring the gradient points in a slightly different direction.) Hence the probability density corresponds to that for a particle moving in a circular path. To sum up, ARPACK finds a basis that is "identical" to the analytical basis, and in addition we can see the correspondence principle in work from the numerical results!

6.8 Intermediate Magnetic Fields

The final experiment is a series of simulations with a constant grid but with varying magnetic field γ. Both the Landau level simulations in the previous section and the pure hydrogen atom simulations yielded qualitatively correct eigenvalues for the lowest-lying states. As we have pointed out it is therefore natural to expect that with an intermediate magnetic field $\gamma \in [0, 1]$ we will also obtain qualitatively correct eigenvalues and eigenfunctions. More information on the intermediate magnetic field regime can be found in Refs. [29, 30].

We shall not perform a thorough analysis. The simulation data is too coarse to find numerical values that are very interesting to present. The simulation produces

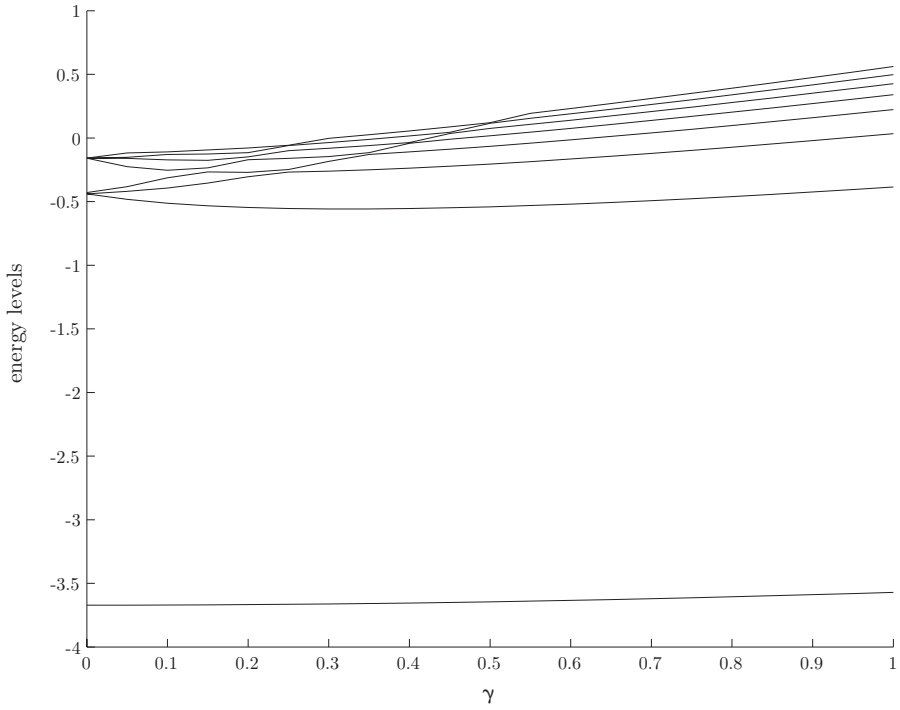

Figure 6.19: The energy levels of the hydrogen atom for intermediate magnetic fields

however qualitatively very interesting results.

We will reuse the grid from Eqn. (6.12) for our simulations. We use a magnetic field with values

$$\gamma \in \{0, 0.05, 0.1, \ldots 0.95, 1.0\}.$$

Fig. 6.19 shows the eight lowest energy levels as function of the magnetic field.

To produce the figure the eigenvalues from each simulation were sorted before they were plotted. We easily recognize the degenerate energy levels at $\gamma = 0$, and as the field is turned on, the degeneracy is broken, producing fork-like structures in the graph.

We can also easily recognize *crossings* of the eigenvalues. As argued in Ref. [29] there must be infinitely many crossings. Furthermore, the eigenvalues accumulate at the Landau levels $2\gamma(N + 1/2)$. Intuitively this is so because the Landau levels are infinitely degenerate, needing infinitely many hydrogen eigenvalues that are *finitely* degenerate to converge. For example, we see a crossing around $\gamma = 0.15$ of the $\epsilon_{2,-1}$ and $\epsilon_{3,-2}$ hydrogen levels. Here, we interpret the curves in terms of the states Ψ_{nm} found with separation of variables, and use the well-known fact that it is indeed these states that wander to the split energy levels, see Ref. [29] for the perturbative treatment of the magnetic field. We also see hints of crossings with curves not shown in the graph, such as the abrupt change of slope of the $\epsilon_{2,0}$ hydrogen state around $\gamma = 0.6$.

6.9 Discussion and Further Applications

Was This a Good Idea? We have spent quite a lot of time finding eigenfunctions and eigenvalues of the two-dimensional hydrogen atom with numerical methods that are quite complicated. We could have found much better eigenvalues and eigenfunctions by concentrating on the radial equation instead of the full Hamiltonian. However, when introducing more complicated Hamiltonians we can no longer use the separation of variables, and the full Hamiltonian must be considered anyway. For example, consider

a two-dimensional non-isotropic harmonic oscillator potential with an applied magnetic field. This system has no rotational symmetry, and hence techniques such as separation of variables are no longer applicable.

Furthermore, we have gained a thorough insight into the eigenvalue problem in the finite element context. Besides its value as a catalyst for insight into the numerical methods, information can be extrapolated to time dependent problems as the quality of eigenvalues have a profound effect on the quality of the time dependent simulations.

Time Evolution of Time-Independent Systems. If the system under consideration is independent of time, i.e.,

$$\frac{\partial H}{\partial t} \equiv 0,$$

we know from section 1.4 that the solution to the time-dependent Schrödinger equation is easy to calculate:

1. Diagonalize H_h, i.e., find the k (where $k \ll N$ is a possibility) lowest eigenvalues and their corresponding eigenvectors, viz.,

$$H_h \Phi_n = E_h \Phi_n, \quad n = 1, 2, \ldots k.$$

2. Pick an initial condition $\Psi(0)$ assumed to be sufficiently smooth, i.e.,

$$\Psi(0) \approx \sum_{n=1}^{k} c_n(0) \Phi_n.$$

   The coefficients are calculated by

$$c_n(0) = \frac{(\Phi_n, \Psi(0))}{\sqrt{(\Phi_n, \Phi_n)}}.$$

3. To find $\Psi(t)$, form

$$\Psi(t) \approx \sum_{n=1}^{k} e^{-iE_n t/\hbar} c_n(0) \Phi_n.$$

This can be used to calculate the time development of the initial state to the final state. If the initial state is *equal* to the superposition of the k first eigenstates the evolution is perfect. The cost of diagonalizing H_h may however be much more than using for example the leap-frog scheme or the theta rule. Then again, these stepping methods introduce numerical errors in the phases of the various components.

If the wave function is desired at very many instants the cost of time-stepping may be cheaper or more expensive than simply forming the superposition in point 3.

Forming the linear combination of eigenvectors requires $\mathcal{O}(kN)$ operations. A time stepping procedure typically includes a forward or backsubstitution of an LU factorized matrix (costing $\mathcal{O}(N^2)$ operations) or using an iterative method for solving the linear systems at hand (with *a priori* unknown price.)

Consequences for the Time Dependent Problems. In addition to providing a means for analyzing probabilities for excitations in time dependent problems, knowledge of the eigenvalues and eigenvectors of the numerical Hamiltonian can tell us a lot about the development of an arbitrary chosen initial state. Such a state can be written as a linear combination of the eigenvectors.

Let us assume that the Hamiltonian is independent of time. In that case, it was explained in section 1.4 that solving the time dependent Schrödinger equation was equivalent to diagonalizing the Hamiltonian.

Assuming that we can solve the spatially discretized Schrödinger equation exactly, i.e., that we can solve

$$\dot{y} = -iH_h y,$$

we know that its solution operator is given by

$$\mathcal{U}(t) = e^{-itH_h}.$$

Expanding y in the eigenvectors ϕ_n of H_h, i.e., the discrete eigenvectors of H, viz.,

$$y(0) = \sum_n c_n \phi_n,$$

then yields

$$y(t) = \mathcal{U}(t)y(0) = \sum_n c_n e^{-it\epsilon_n} \phi_n.$$

For the particle in box we know that linear elements reproduce the *exact* eigenvectors. It is reasonable to believe that with quadratic (or higher-order) elements the eigenvectors are still (or very close to) exact. Hence, for the time development it is easy to see that the better the eigenvalues we have obtained, the better the time development of a wave packet will be.

If our ODE solving algorithm is exact, or close to exact, it is clear that the eigenvalues of H_h will enter the numerical solution operator of the ODE. For example, the Crank-Nicholson scheme has a solution operator U_Δ explicitly given by the eigenvectors and eigenvalues of H_h; see section 1.4. The better eigenvalues we obtain, the better will the solution of the time dependent Schrödinger equation look as well.

This gives us a means for predicting whether or not a spatial discretization of some system is reasonable or not. Thus, if we find that the eigenvalues of, say, the two-dimensional hydrogen atom is way-off the analytic eigenvalues, we must think twice before we start solving time dependent problems with this discretization.

Doing Large-Scale Simulations. The simulations in this chapter were done on various computers, but in general they were ordinary workstations that happened to be available at the time with moderate CPU speed and amounts of memory. For example, the hydrogen simulations were performed on a machine running Debian Linux with 1 GB of memory and a 2.4 GHz Intel Pentium 4 processor. The simulation times ranged from a few minutes to two hours, and the simulation time of course increases with the dimension of the matrix. The increase is not optimal (i.e., linear) but neither is it quadratic. (Detailed results are not sought in this thesis.)

For realistic simulations however, we will typically use a cluster with dozens of CPUs and lots of memory. Both Diffpack and ARPACK has parallelization support and this is (not only in principle) possible to take advantage of, even though it requires some details in the implementation area that are not yet sorted out.

Simulating for example the eigenvalues of a hydrogen atom with an arbitrary magnetic field with a grid with for example 10^6 grid points is then easily done, yielding results superior to those presented in this text. Of course, the various details of memory and storage requirements, the expected accuracy from such simulations and so on must be investigated before embarking on such a mission.

Quantum Chaos. Quantum chaos is the study of quantum mechanical systems corresponding to classical systems whose trajectories are chaotic, e.g., a double pendulum, three-body systems, particle in a stadium-shaped box, et.c. In section 15.6 of Ref. [9] eigenstates of the latter system is depicted, and perhaps the most characteristic feature is the so-called *scarring* of the highly-excited states. The classically periodic paths,

which are unstable in the sense of Lyapunov, "shine through" the otherwise noisy structure of the eigenstates. This is obviously connected with the correspondence principle, but the fundamental mechanisms are not fully understood at the time of writing.

When we studied the strong-field limit of the two-dimensional hydrogen atom with an applied magnetic field we saw the correspondence principle in work as the classically periodic trajectories were seen in the quantum mechanical eigenstates. The idea is then to use the techniques presented here for *arbitrary systems* to produce scarred eigenstates of high energy. In addition one may do statistical computations on the eigenvalues, e.g., as function of mesh width.

To what extent the finite element method has been applied to such systems I have not investigated, but it is on the other hand clear that rich insight into both the understanding of the finite element method and the spectrum and eigenfunctions of the finite element matrices is possible to gain.

Gauge Invariance. We know that quantum physics is invariant with respect to gauge transformations as described in section 1.6. What we have not studied up to this point is whether or not the numerical methods are gauge invariant or not. By gauge invariance of the numerical methods in the eigenvalue context we mean that the eigenvalues are left unchanged under a gauge transformation and that the eigenfunctions are given by a unitary change of basis similar to that of the continuous problem. Our `HydroEigen` simulator class actually implements a non-symmetric gauge in addition to the symmetric gauge used in the simulations in this chapter. Due to time limitations no simulations worth mentioning has been done for this thesis.

Gauge invariance of numerical methods in general is a very interesting subject to do further studies on. First of all, it has a profound impact on our understanding of numerical methods. Second, it will automatically yield valuable confirmative or dismissive information on the quality of the results of a numerical experiment. If the methods are known to be gauge invariant up to a term proportional to for example $\mathcal{O}(\Delta t^4)$, we know that it is much less than the error from a simulation with the leap-frog scheme or the split-operator method used in section 4.3.

Improvement of the Program and the Methods. Even though we have used the finite element method quite generally, the complete picture has not been drawn. So-called *adaptive finite element methods* may be used to successively improve the grid based on local *a posteriori* error estimates. Such estimates must be derived for each PDE in question and allows for estimating the error in for example the norm of the error over each element. If the error is above some threshold one refines the grid around the element and performs a new simulation. In Ref. [49] such a technique for the Schrödinger equation is presented. Adaptive finite element methods are treated to some extent in Ref. [34].

One can imagine that adaptive FEM can be used to find a suitable grid for the lowest-lying states of the physical problem, and then use this grid in a time-dependent simulation.

As for the program, many improvements can be done. Besides cosmetic changes, a more robust handling of matrix types would be valuable. As for now, only `MatSparse` and `MatBand` are supported, and this only in a limited manner. Not all storage options inside the matrices are handled properly. A complete and robust eigenvalue program for the Hamiltonian could also serve as a starting point for more general eigenvalue problem implementations in Diffpack, as the built-in support for this is virtually non-existent at the time of writing.

Chapter 7

Solving the Time Dependent Schrödinger Equation

7.1 Physical System

We will implement the solver for a single charged particle in an attractive Coulomb potential with an applied time-dependent magnetic field. We will choose a simple model for an ultra-short laser pulse with amplitude γ_0, viz.,

$$\gamma(t) = \gamma_0 \sin^2\left(\frac{\pi t}{T}\right) \cos[2\pi\omega(t - T/2 + \delta)]. \tag{7.1}$$

Here, ω is the frequency of the laser and δ is a phase shift. The envelope rises from zero to γ_0 at $t = T/2$ and falls back again to zero at $t = T$, i.e., at the end of the simulation.

Such laser pulses are actually possible to create experimentally, see Ref. [50]. Usually one considers very long pulses in which $\omega \gg 1/T$. With this model perturbative methods may yield very accurate answers, but when $\omega T \sim 1$ we are outside the regime for such treatment. The physical system is also possible to create in a laboratory, and variants include electrons trapped in anharmonic oscillator potentials, ions in Paul traps, et.c., see Ref. [51]. Therefore it poses an interesting starting point for doing simulations. Furthermore, simpler systems such as the free particle can be created as special cases in the program by turning on and off the various parameters available.

In this thesis we will only consider this magnetic field as a toy model with which we test our simulator.

We will implement the possibility of either using an eigenfunction of the Hamiltonian (typically with $\gamma = 0$) or a Gaussian wave packet as initial condition. This is described further in the next section.

We have to make a comment on the Hamiltonian. The vector potential $\boldsymbol{A}$ in both the symmetric and non-symmetric gauges is obtained via the dipole-approximation, i.e., one assumes that the electromagnetic fields depend only on time. This is reasonable if one imagines a very distant source of radiation. However, as was pointed out on page 54, we are actually neglecting Maxwell's equations. A spatially independent magnetic (or electric field) must be a constant field in order to be consistent. We *should* have an extra term proportional to the scalar potential in the Hamiltonian. The electric field is given by

$$\boldsymbol{E} = \frac{1}{c}\int^t \mathrm{d}\tau\ \boldsymbol{A}(\tau),$$

but this is easily seen to be spatially dependent! It is also easy to see that the physical electric field becomes *very* different in the two gauges.

We will ignore the difficulties in this chapter, and simply note that introducing rigor to this topic is an interesting extension of the discussion.

7.2 The Implementation

The implementation of the time dependent solver (`class TimeSolver`) is derived from `class HydroEigen`. There are several advantages of reusing the code from the eigenvalue solver. First, many features of the problems are common, e.g., the Hamiltonian and the finite element assembly process. This reduces the need for code. Indeed, the source code for the time dependent solver is a great deal shorter than the `HydroEigen` class definition even though one hardly can say that the problem is less complex for that reason. Second, the final application may be used to solve both the eigenvalue problem and the time dependent problem. It is also easier to keep the interfaces common for the two simulation types, which is highly desirable. Unfortunately some loss of efficiency is inevitable with this approach.

7.2.1 Time Stepping

Although we have described the numerical methods in details elsewhere, we here present in explicit terms the updating algorithms for the numerical wave functions in the leap-frog and theta-rule methods.

The Leap-Frog Scheme. The updating rule for the leap-frog scheme (4.27) in matrix form reads

$$u^{\ell+1} = u^{\ell-1} - 2i\Delta t M^{-1} H^{\ell} u^{\ell},$$

where H^{ℓ} is the finite element Hamiltonian used in the eigenvalue solver and where M is the mass matrix; a positive definite and symmetric matrix. The strategy is to build a linear system $Mv = b$ at each time step, where

$$b = H^{\ell} u^{\ell}.$$

The solution at the next time step then becomes

$$u^{\ell+1} = u^{\ell-1} - 2i\Delta t v.$$

The linear system becomes very easy to solve explicitly if we lump the mass matrix. This is taken advantage of in the implementation with a special hand-coded diagonal solver.

Given the initial condition u^0 (which may be a Gaussian wave packet or an eigenstate of the Hamiltonian) we need a special rule to find u^1. This special rule is simply a single iteration with the theta-rule.

The implementational details of building the linear system is somewhat complicated due to the structure of Diffpack's solution algorithms and the wish for reusing the code for calculating the Hamiltonian from `class HydroEigen`. The resulting code should be quite effective, though. The overhead of extra function calls does not dominate the time spent in the new element assembly routine `TimeSolver::makeSystem()`.

The Theta-Rule. In matrix form the updating rule (4.19) for the theta-rule reads

$$u^{\ell+1} = [M + i\theta\Delta t H^{\ell+1}]^{-1}[M - i(1-\theta)\Delta t H^{\ell}]u^{\ell}.$$

If we define $b = [M - i(1-\theta)\Delta t H^{\ell}]u^{\ell}$, our new wave function becomes the solution to the linear system $Au^{\ell} = b$, with

$$A = M + i\theta\Delta t H^{\ell+1}.$$

The theta-rule is somewhat trickier than the leap-frog scheme to implement in an efficient manner. This is due to the fact that the Hamiltonian at *different* time levels is needed on the left hand side and the right hand side of the linear system.

We will always use $\theta = 1/2$ as we know this is the best choice. For this reason we refer to the Crank-Nicholson method instead of the theta-rule in the following.

Solving Linear Systems. For the time dependent solver the total time spent at the assembly process can be considerable. The linear systems must also be solved in an efficient manner as we deal with matrices of potentially very large sizes. It is obvious for two reasons that we must avoid the use of banded matrices. First, banded matrices store a lot of zeroes for very sparse systems, as can be seen in Fig. 5.1. Second, the linear solvers are very slow with banded matrices due to numerous superfluous multiplications and additions with zeroes. For our simulations we will therefore exclusively use iterative methods and sparse matrices.

For the simulations in this chapter we stick to the *Generalized Minimal Residual* method (`GMRES` in Diffpack, see Ref. [34]). It performs well on both the types of linear equations we deal with.

Keeping Track of the Solution. For the purpose of analyzing data from the simulation, the log file is written as a Matlab script. The script defines arrays and fills them with simulation data, such as the time (the `t` array), the square norm of the solution (the `u_norm` array) and so on. The arrays are located in a `struct` variable *`casename`*`_data`.

Some of the variables and arrays that are defined are:

- `casename`: The case name for the simulation.
- `n_rep`: The number of reports. Equals the number of rows in each array.
- `t`: The time array.
- `gamma`: The magnetic field array.
- `dt`: The time step.
- `theta`: The parameter for the theta rule.
- `gamma0`, `omega` and `delta`: The parameters for the magnetic field.
- `gaussian_params`: A string with the parameters for the Gaussian initial condition.
- `u_norm`: Array with the norm of the solution.
- `u_energy`: Array with the energy of the solution, i.e., the expectation value of the Hamiltonian.
- `u_pos`: Array with the expectation value of the position. The first column is the x-coordinate and the second column is the y-coordinate.
- `system_time`: An array with the time for each assembly of the linear system.
- `solve_time`: The time for each solution of a linear system. (Not reported if the leap-frog method is used with a lumped matrix.)
- `solve_niter`: The number of iterations taken for the solution to converge.
- `solve_total` and `system_total`: The total time for the solution of linear systems and the assembly of the systems, respectively.

7.2.2 Description of the Member Functions

Here we describe the most important new methods of `class TimeSolver`. Many member functions are inherited from `class HydroEigen` and therefore not described.

`define()`. The menu system is extended with several parameters. The menu items from `class HydroEigen` such as the parameters for turning on and off terms in the Hamiltonian are kept. All the new parameters except for `simulation in time` are added inside a submenu to distinguish them from the eigenvalue problem parameters. The submenu can be reached with the command `'sub time'`.

– `simulation in time` This boolean parameter selects the time dependent solver if its value is `true` and solves an eigenvalue problem otherwise.

– `time method` This parameter is used to select the time integration method. The only legal values are the strings `theta-rule` and `leap-frog`.

– `T` This is the time at which the simulation is to be stopped. In other words the simulation is performed for $t \in [0, \mathtt{T}]$.

– `dt` This is the time step. Thus, about `T/dt` steps are taken in the simulation.

– `n_rep` This parameter is an integer signifying how many reports are desired throughout the simulation. Thus, at the end of every time interval of length `T/n_rep` fields and various physical quantities are written to the field database and the log file. A report is automatically issued before any time integration is done, making the total number of reports `n_rep` + 1.

– `theta` This is the parameter for the theta-rule.

– `gamma0`, `delta` **and** `omega` These are the parameters for the time dependent magnetic field in Eqn. (7.1).

– `ic type` Two kinds of initial conditions are implemented. If `ic type` is set to `gaussian`, a Gaussian wave packet will be used. If `ic type` is equal to `field`, a field is read from a database and used as initial condition.

– `field database` This is the casename of a previously stored set of eigenvectors from an eigenvalue simulation. The fields are read with the method `loadFields()` and overwrites any grid defined in `gridfile`, see section 6.5. If the value of the parameter is `none`, no field database will be loaded.

– `ic field no` This integer is the index of the field to use as initial condition. If the index is larger than the number of fields in the `field database`, the ground state, i.e., field number one, is automatically used.

– `gaussian` This is a string with parameters for the Gaussian initial condition. It is a string with switches such as `-x` followed by a real number. The default value of `gaussian` is `'-x -10 -y 0 -sx 2 -sy 2 -kx 0 -ky 1'`. The switches `-x` and `-y` sets the mean position, the switches `-sx` and `-sy` sets the standard deviations in the x and y directions, and finally `-kx` and `-ky` sets the mean momentum.

`fillEssBC()`. This method imposes the essential boundary conditions, i.e., the homogenous Dirichlet conditions. It loops through the grid nodes and gives the `DegFreeFE *dof` object information on where to prescribe zeroes. This object is then responsible for altering the element matrix before any linear system of equations is solved.

setIC(). This method sets the initial condition according to the choices in the menu. The two fields `u` and `u_prev` are set equal to a Gaussian or an eigenvector from the `field database`.

The two fields represent the current solution (`u`) and the previous solution (`u_prev`). In the theta-rule the `u_prev` field is not really necessary to use because we only need the current solution to find the next.

scan(). The `scan()` function is updated heavily, with support for reading previously stored fields from a simres database. It is important to notice that if such a database is loaded, the grid defined with the `gridfile` menu entry is overruled with the grid stored in the database.

The `scan()` function also allocates memory for various objects, such as scratch vectors needed in the time integration. In other respects the function is similar to the `scan()` function in `class HydroEigen`.

gammaFunc(). This method updates the magnetic field according to the time passed as parameter.

calcExpectationFromMatrix(). This function calculates the matrix element of a `Matrix(NUMT)` object with respect to two given fields. This is particularly useful if one wants to calculate the expectation value of observables available as matrices, e.g., the Hamiltonian.[1]

calcExpectation_pos(). It is useful to be able to calculate the mean position of the wave function during the course of a simulation. This is the purpose of this method which returns a `Ptv(real)` object holding $\langle x \rangle$ and $\langle y \rangle$.

loadFields(). This function reads fields stored in a simres database. The fields are assumed to have been stored by `HydroEigen` as the method selects only fields with an integer as label.

If fields are actually read, the `GridFE` object in the simulator is overwritten with the grid from the fields.

integrands(). The new `integrands()` method calculates the integrand used in the assembly of the left and right hand side of the linear equations. It can also compute the Hamiltonian; this is necessary if one wants to solve an eigenvalue problem instead of a time dependent problem.

We will not go into details on the `integrands()` method; the code should contain more than enough comments. We mention though that the use of the local-to-global mapping of nodes $q(e, r)$ is central when calculating the right hand side terms. This mapping can be found in the (public) member `VecSimple(int) loc2glob_u` of `class ElmMatVec`. This class contains an $n \times n$ matrix `Mat(NUMT) A` and an n-dimensional vector `Vec(NUMT) b`, where n is the number of nodes in the current element. These are the local element matrix and element vector, respectively, and in the assembly process each term A_e and b_e of Eqn. (4.16) is given in terms of these smaller matrices and the $q(e, r)$ mapping, in a way generalizing the one-dimensional example from section 4.4.2.

solveProblem(). This is a central class member function. It first checks whether a time dependent simulation actually is wanted (and branches to `HydroEigen::solveProblem()` if not) and then loops through the desired time levels.

[1] The matrix used is the Hamiltonian modified due to essential boundary conditions and then symmetrized, see Ref. [34]. If this matrix has the same expectation value as the original matrix when we know that the BCs are fulfilled is unclear at the moment, but when the wave function is essentially zero near the boundary we may assume that it is very close.

`reportAtThisTimeStep()`. During the simulation it is desirable that the simulator provides some quantitative results, such as the norm of the solution, the mean position, the energy and so on. This is provided by the `reportAtThisTimeStep()` method. It combines dumping the solution u^ℓ and the corresponding probability density field to the simres database (named with *`casename`*; see the description of `class HydroEigen`) with writing statistical data to `stderr` and to *`caseneme`*`.log` in Matlab format.

7.2.3 Comments

More on the Initial Conditions. The wave packet used for initial conditions is implemented as a subclass `GaussFunc` of the Diffpack class `FieldFunc`, i.e., a *functor* that encapsulates functions that can be evaluated at a space-time point. For example, a `FieldFE` object such as the current solution `u` can be initialized to a Gaussian with this code:

```
GaussFunc gaussian; // create functor
gaussian.scan(menu.get("gaussian")); // init functor
u.fill(gaussian); // fill u
```

The mathematical expression used for the nodal values of the Gaussian is

$$U_j = \exp\left(-\frac{(x^{[j]}-x_0)^2}{2\sigma_x^2} - \frac{(y^{[j]}-y_0)^2}{2\sigma_y^2} + i\boldsymbol{k}\cdot\boldsymbol{x}^{[j]}\right).$$

The parameters x_0, y_0, σ_x and σ_y correspond to the switches `-x`, `-y`, `-sx` and `-sy` of the initialization string, respectively. The momentum $\boldsymbol{k}$ corresponds to `-kx` and `-ky`. Numerical normalization is provided by the `setIC()` method and `calcInnerProd()`.

As described above one can use a field read from a field database as initial condition as well as a Gaussian wave packet. In the eigenvalue solver `class HydroEigen` the fields are stored and labeled with the integer index k from the eigenvalue ϵ_k of the field. (The corresponding probability densities are labeled `prob_`k.) The `loadFields()` method reads the fields with integer labels only and stores them in an array of `Handle(FieldFE)` objects. The field objects are useful when one wants to compute the spectrum of a numerical state, i.e., the overlap with the eigenstates of the (stationary) Hamiltonian. This can be done easily with `HydroEigen::calcInnerProduct()`.

Analytical Integration. It is claimed in this thesis that Diffpack solely relies on numerical integration to compute the systems of linear equations. This is not entirely true as one can override the `FEM::calcElmMatVec()` method with another one which actually computes the element vectors and matrices analytically. For the Coulomb interaction term this is actually possible, and we may avoid any numerical inaccuracy due to the limited order of the Gaussian quadrature.

This is not investigated any further in this thesis, but might be interesting in a future project.

Obtaining a Finite Difference Scheme. We may obtain the standard finite difference scheme (at least for some classes of Hamiltonians) with the finite element formulation and a special set of parameters. More specifically, we use a uniform grid, linear elements and simple nodal point numerical integration. Let us demonstrate this for a one-dimensional example.

Let h be the grid spacing. Let there be m nodes such that $\Omega = [0,(m-1)h]$ and the grid points are given by $x_k = (k-1)h$. The nodal-point integration is defined by

$$\int_\Omega f(x)\,\mathrm{d}x \approx h\sum_{k=1}^{m} f(x_k).$$

The mass matrix becomes

$$M_{ij} = h \sum_k N_i(x_k) N_j(x_k) = h \sum_k \delta_{ik}\delta_{kj} = h\delta_{ij},$$

i.e., we obtain the lumped mass matrix. For an arbitrary operator $V(x)$ we similarly obtain the matrix

$$V_{ij} = h\delta_{ij} V(x_i),$$

i.e., a diagonal operator. Notice that the nodal point integration corresponds to zeroeth order Gaussian quadrature in local coordinates. Hence, it integrates constant functions analytically. The derivatives $N_i'(x)$ are constant over each element. Therefore, the stiffness matrix is integrated analytically, and yields the standard finite difference operator (multiplied by h), viz.,

$$K_{ii} = \frac{2}{h}, \quad K_{i,i\pm1} = \frac{-1}{h}.$$

Thereby, any Hamiltonian on the form

$$H = -\frac{\partial^2}{\partial x^2} + V(x)$$

is discretized with a scheme identical to the standard finite difference scheme.

The argument generalizes to more dimensions, but we must be careful with the angular momentum operator and other first order differential operators which do not fit so easily in. Anyway we can consider the finite element method with linear elements and nodal point integration (and a lumped mass matrix) as corresponding to a (modified) finite difference scheme. This rule also in some sense generalizes the finite difference concept if we include non-uniform meshes and perhaps triangular elements as well.

7.3 Numerical Experiments

In this section we do a collection of numerical experiments with various settings to figure out some properties of the numerical methods. We do not focus on physics right now, because it is more important to know whether our methods work or not and what methods are best to use.

We will use a mixture of analytical reasoning and physical intuition when analyzing the results. The aspects we wish to study are:

- *The time spent on building and solving linear systems.* When using finite element methods (and implicit finite difference methods) most of the computational effort is put into solving linear systems of equations. For time dependent problems the time spent on building the systems is also an important factor. We try to determine the relationship between the size (i.e., the number of nodes) in the grid and the time spent on these operations for different methods.
- *Comparison of the Crank-Nicholson scheme and the leap-frog scheme.* The schemes may seem similar when it comes to computational cost, at least for finite element methods. There are however some differences, and we will try to study some of them.
- *Different element types.* The finite element method allows a wide range of element types, such as triangular elements, quadratic elements and so forth. We do a quick comparison between linear and quadratic finite elements.
- *A simulation of the full physical model.* Finally we are ready to do a simulaton of the full system. We will describe the results qualitatively and compare simulations with different methods.

Hopefully, these experiments will provide us with valuable information on the performance of the numerical methods. Along the way we will also come across several interesting questions that may lay grounds for exciting future work, both physical and numerical.

It is important to keep in mind that the experiments performed here are very simple and to not exploit the full capability of the finite element method, such as the extremely flexible geometry of the grid. Analyzing numerical results are much simpler on highly structured grids. When one wants to perform "real" experiments, the flexibility in the location and size of the elements should be exploited.

7.3.1 Building and Solving Linear Systems

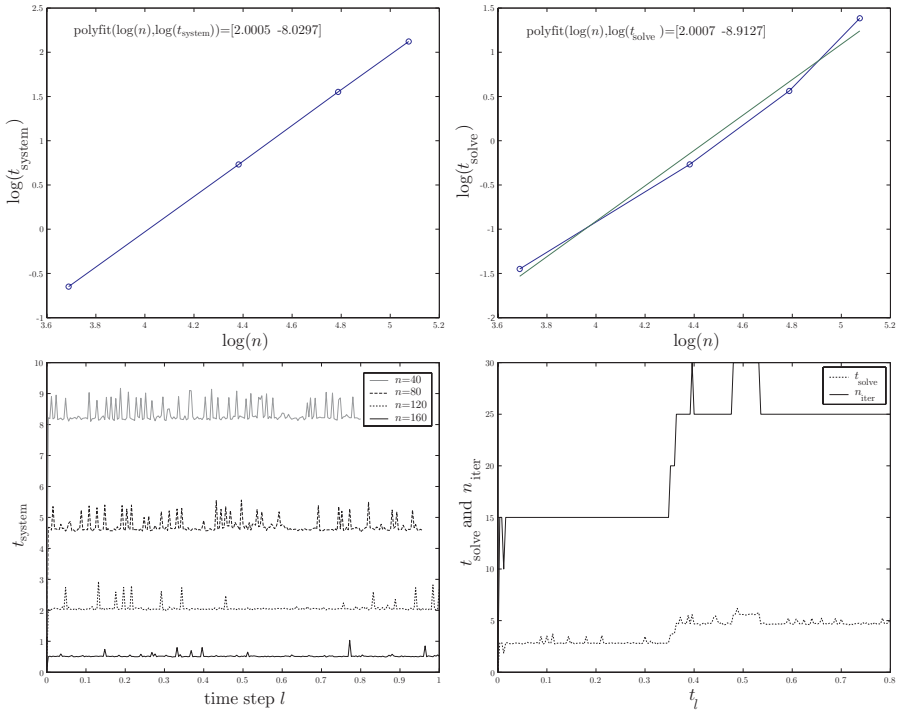

Figure 7.1: Lower left: the time for creating linear systems for each time step. Upper left: Log-log plot of the average time spent on building the system. Lower right: The time for solving the linear systems with $n = 160$ and the number of iterations needed. Upper right: Log-log plot of the average time spent on the solving.

The most time consuming part of a finite element simulation is the solution of linear systems. These tend to be very large and sparse. Here we will investigate the increase in the time spent on solving linear equations as function of the number of nodes in the grid. Here and in the rest of the simulations on this section, we will employ a "finite difference grid," i.e., a grid with uniform spacing in every direction and elements with a square shape.

The time spent on *building* linear systems should be approximately the same from time step to time step. In the theta-rule we build *two* Hamiltonians and mass matrices (for the left and right hand side, respectively) at each time step, roughly doubling the time spent in the leap-frog scheme.

The linear systems are solved with the `GMRES` method. The iteration process requires

a start vector, and we use the previous solution u^ℓ as initial vector when solving for $u^{\ell+1}$. At least for the Crank-Nicholson scheme it is intuitively close to the new solution and therefore we expect that the iteration process converges fast. One may also expect that the time step affects the convergence rate: The smaller the time step, the faster the convergence. This can turn out to be very important if true! Indeed, then the simulation time does not increase linearly with the total number of time steps, but *slower* than linear, i.e. $\mathcal{O}(\Delta t^{-1/p})$, with $p \geq 1$.

We perform a series of simulations with constant physical problem definition but with increasing number of subdivisions n in each spatial direction.

- We use the full Hamiltonian with the Coulomb term and the time-dependent magnetic field with $\omega = \delta = 0$ and $\gamma_0 = 1$ in order to capture effects from every aspect of the full problem.
- Our initial condition is a Gaussian wave packet located at $\boldsymbol{x}_0 = (-10, -10)$ with momentum $\boldsymbol{k} = (\sqrt{2}, \sqrt{2})$ and width $\boldsymbol{\sigma} = (4, 4)$. It is located safely away from the origin but moving towards it with velocity $2\boldsymbol{k}$. (Its speed is $2\|\boldsymbol{k}\| = 4$.)
- We simulate for $t \in [0, 1]$ with $\Delta t = 0.002$.
- We utilize a square domain $\Omega = [-20, 20] \times [-20, 20]$ subdivided into n^2 squares with sides $40/n$.
- We let n vary over a series of single simulations, viz.,

$$n = (40, \quad 80, \quad 120, \quad 160). \tag{7.2}$$

- We utilize the leap-frog scheme in these simulations.

We monitored the time $t_{\text{system}}(n, \ell)$ spent on building the linear systems and the time $t_{\text{solve}}(n, \ell)$ spent on solving the linear equations, where ℓ is the time step. It is expected that the time spent on building the linear system is independent of ℓ because it is the exact same sequence of operations that is performed each time. This is also observed, see Fig. 7.1 in which we have graphed the time spent on building the linear systems for each simulation. All the simulations produced qualitatively the same time-dependent solutions. (They are not very interesting so we do not show them here.) The time spent on solving the equations is also expected to not depend too much on the time step number ℓ, and this is also seen in Fig. 7.1.

The time step was unfortunately chosen arbitrarily and believed to be within the stability condition for the leap-frog scheme with linear elements, while it actually violated the condition for the largest grids. The simulations for the two finest grids developed instabilities near the middle and end of the simulations. This is the reason why the corresponding graphs are shorter.

The average (over ℓ) time in seconds spent on building linear systems is

$$\bar{t}_{\text{system}} = (0.5226, \quad 2.0796, \quad 4.7194, \quad 8.3457).$$

The order of the numbers is from low to high n. The average time spent on solving the systems is

$$\bar{t}_{\text{solve}} = (0.2346, \quad 0.7656, \quad 1.7548, \quad 3.9920).$$

We conjecture that the average time $\bar{t}(n) = \mathcal{O}(n^p)$ for some $p \geq 1$ for both cases. To check this we graph the logarithm of $\bar{t}$ versus the logarithm of n, similar to what we did in section 6.6. Fig. 7.1 shows these plots. The data were run through Matlab's regression function `polyfit` and the resulting polynomials are shown as well. (For $\bar{t}_{\text{system}}$ the linear function is not plotted because it is indistinguishable from the data.) Clearly, we have

$$\bar{t}_{\text{system, solve}} \sim n^2 = N$$

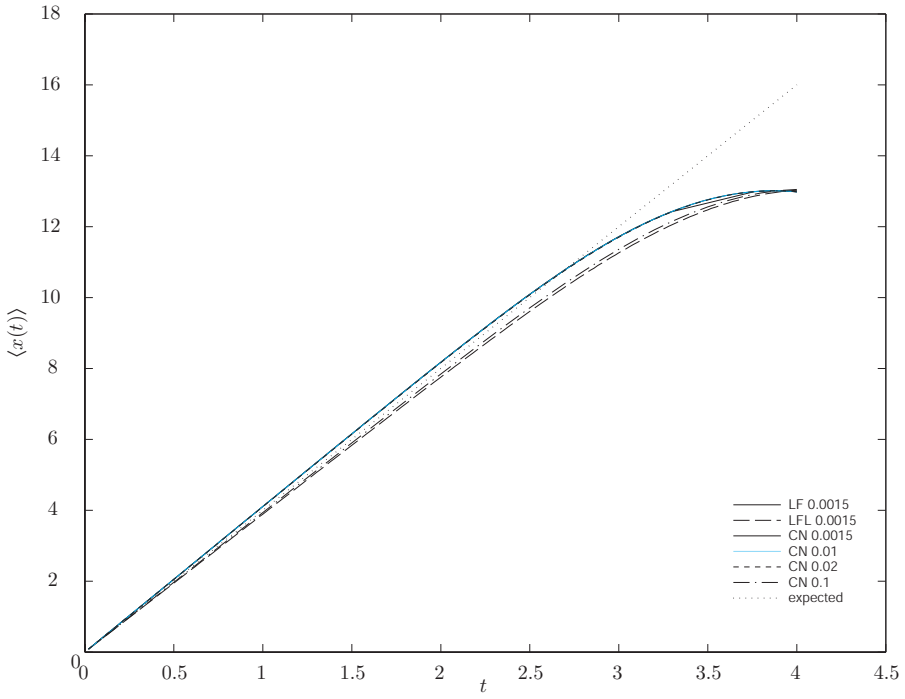

Figure 7.2: Expectation value $\langle x \rangle$ for the different simulations

where N is the dimension of the element matrix. In other words we have approximately a *linear scaling of the computational cost* with the leap-frog method. It is not unreasonable to guess that this holds also for the Crank-Nicholson scheme, because the matrices have exactly the same structure in the two cases.

Contrast the $\mathcal{O}(N)$ result of solving linear systems with the $\mathcal{O}(N^3)$ result for gaussian elimination of dense matrices. It would be interesting to check this scaling property with other element types, such as quadratic elements that generate denser matrices.

As for Fig. 7.1 a few comments are in place. First, the plots with t_ℓ on the horizontal axis (the lower plots) show small fluctuations. This may be due to for example other processes running on the computer. The number of iterations n_{iter} is clearly proportional to t_{solve}, reflecting that each iteration requires a fixed amount of numerical work.

7.3.2 Comparing the Crank-Nicholson and the Leap-Frog Schemes

The Crank-Nicholson scheme and the leap-frog scheme might look similar at first when it comes to numerical cost. If we do not lump the mass matrix in the leap-frog scheme, we need to solve a linear system of equations at each time step. Considering the stability criterion for each scheme, the leap-frog scheme might look useless when compared to the Crank-Nicholson scheme. As we saw in section 4.5.4, the unitarity of the scheme (for a time-independent Hamiltonian) is only secured if the time step was smaller than the inverse of the largest eigenvalue of H, viz.,

$$\Delta t \leq \frac{1}{\epsilon_{\max}}.$$

The Crank-Nicholson scheme has no such restrictions in the time-independent case.

However, there are two questions that must be answered. First, is the accuracy of the Crank-Nicholson scheme just as good as the leap-frog scheme if we take longer time steps in the former? Maybe we need to reduce the time step in order to achieve the

same precision as the leap-frog scheme. Second, are the solutions to the linear equations utilizing M as the coefficient matrix just as expensive as the ones used in the Crank-Nicholson scheme? Intuitively no, because M is positive definite and symmetric, while the coefficient matrix in the Crank-Nicholson scheme is non-symmetric. Furthermore, lumping the mass matrix in the leap-frog scheme makes the linear systems trivial to solve, improving the efficiency drastically. (If we also use nodal-point integration and linear elements, we obtain a scheme equivalent to the standard finite difference scheme.)

To determine some answers experimentally we perform a series of simulations with a time independent Hamiltonian with different methods and time steps. We use the particle-in-box Hamiltonian and a Gaussian initial conditions, because we know (at least before the packet hits the boundary) the exact qualitative behavior of the wave-packet and expectation values of position. Indeed, the free wave packet has mean momentum $\boldsymbol{k}$, so the mean position should be

$$\langle \boldsymbol{x} \rangle = \boldsymbol{x}_0 + 2t\boldsymbol{k},$$

where $\boldsymbol{x}_0$ is the initial mean position and where $\boldsymbol{k}$ is the momentum.

The wave packet used for our simulations start out at $\boldsymbol{x}_0 = (0,\, 0)$ with momentum $\boldsymbol{k} = (2,\, 0)$. The expected velocity is therefore $\boldsymbol{v} = 2\boldsymbol{k} = (4,\, 0)$. The width of the wave packet is $\boldsymbol{\sigma} = (5,\, 5)$.

The grid is a "finite difference grid" discretizing the domain $\Omega = [-16, 16]^2$ with $n = 160$ subdivisions along each side. We use linear elements. The stability criterion for the leap-frog scheme is

$$\Delta t \leq \frac{h^2}{8} = 0.005$$

for the finite difference method, and roughly

$$\Delta t \leq \frac{h^2}{24} = 0.0016666$$

for the linear finite elements. (The latter is only an estimate because the highest energy in two dimensions in the finite element method is not exactly twice the one-dimensional energy.) Therefore we use $\Delta t = 0.0015$ for the leap-frog method, hopefully staying on the right side of the stability criterion. In the simulations we let $t \in [0, 4]$.

We perform six different simulations with the following numerical parameters:

1. The leap-frog method with the full mass matrix, $\Delta t = 0.0015$. This case is referred to as LF 0.0015 in the figures et.c.

   The total simulation time (in seconds) for this case was $t_{\text{total}} \approx 15.5 \cdot 10^3$.

2. The leap-frog method with a lumped mass matrix and nodal point integration, i.e., the finite difference method, and using the same time step as before. This is referred to as LFL 0.0015.

   The total simulation time was $t_{\text{total}} \approx 10.1 \cdot 10^3$.

3. The Crank-Nicholson method with $\Delta t = 0.0015$. This is referred to as CN 0.0015.

   The total simulation time was $t_{\text{total}} \approx 32 \cdot 10^3$.

4. Same as the previous, but with time step $\Delta t = 0.01$. This is referred to as CN 0.01.

   The total simulation time was (somewhat surprisingly) $t_{\text{total}} \approx 2.3 \cdot 10^3$.

5. Same as the previous, but with time step $\Delta t = 0.02$ This is referred to as CN 0.02.

   The total simulation time (in seconds) for this case was $t_{\text{total}} \approx 1.1 \cdot 10^3$.

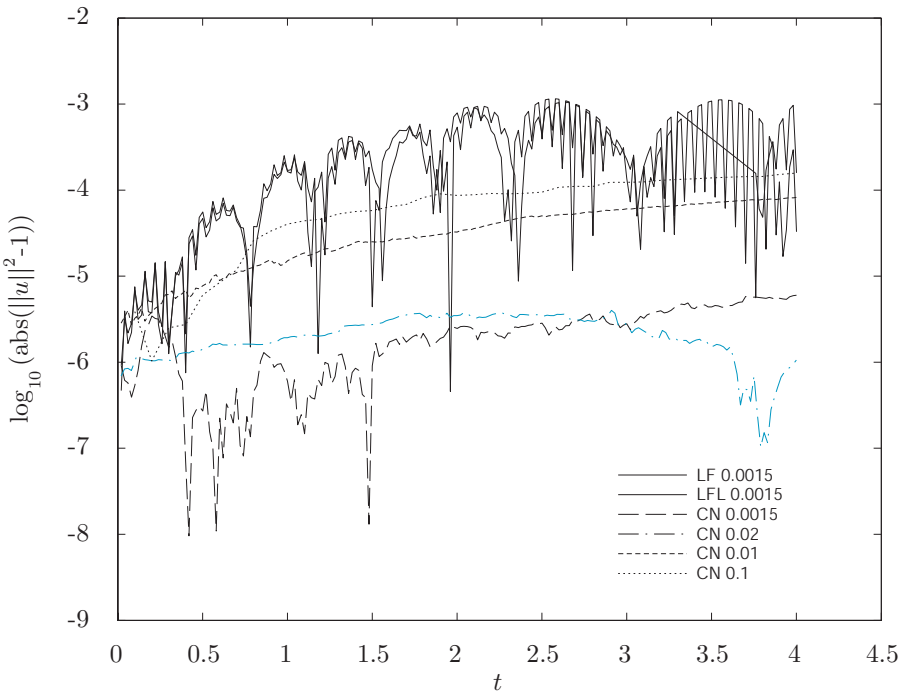

Figure 7.3: Deviation from unitarity in the simulations

6. Finally, we use a time step $\Delta t = 0.1$ in the Crank-Nicholson method. This we refer to as CN 0.1.

   The total simulation time was $t_{\text{total}} = 665.8$.

Animations of the simulations can be reached from Ref. [5]. These are only interesting inasmuch that qualitatively they are all the same, and they reproduce the expected qualitative behavior.[2]

The simulation times must be taken with a pinch of salt. The finite difference scheme LFL 0.0015 was not optimized in the implementation. On the contrary, building linear systems and solving them (even if they are trivial) represent much more work than needed in an implementation of the method. A proper implementation is much faster. The Crank-Nicholson method on the other hand needs solutions to linear systems, even in the finite difference case.

In Fig. 7.2 the mean position $\langle x \rangle$ as function of t is plotted for each simulation. The expected behavior is plotted as a dashed straight line. We notice that the expected behavior is largely reproduced. The slope of the curves drop near the end of the simulations due to collision with the boundary. Contrary to a classical particle which bounces right off the wall, the quantum particle turns around in a smooth fashion.

The different simulations show small but important variations in the mean position $\langle x \rangle$. Some of the simulations overestimate the velocity of the packet while others underestimate it. The simulations that overestimate the velocity are LF 0.0015, CN 0.0015, CN 0.01 and CN 0.02. The simulation LFL 0.0015, i.e. the finite difference method, yielded qualitatively the same result as CN 0.1, i.e., Crank-Nicholson with a very large time step. This indicates (bot it does not prove) that the finite difference leap-frog method is similar to Crank-Nicholson with (much) larger time step than the stability criterion.

[2] Actually, behavior characteristic to quantum mechanical collisions show up in the animations, namely the packet's interference with itself as it collides with the wall. See also the link to other animations and descriptions in Ref. [5] with further investigation of this phenomenon.

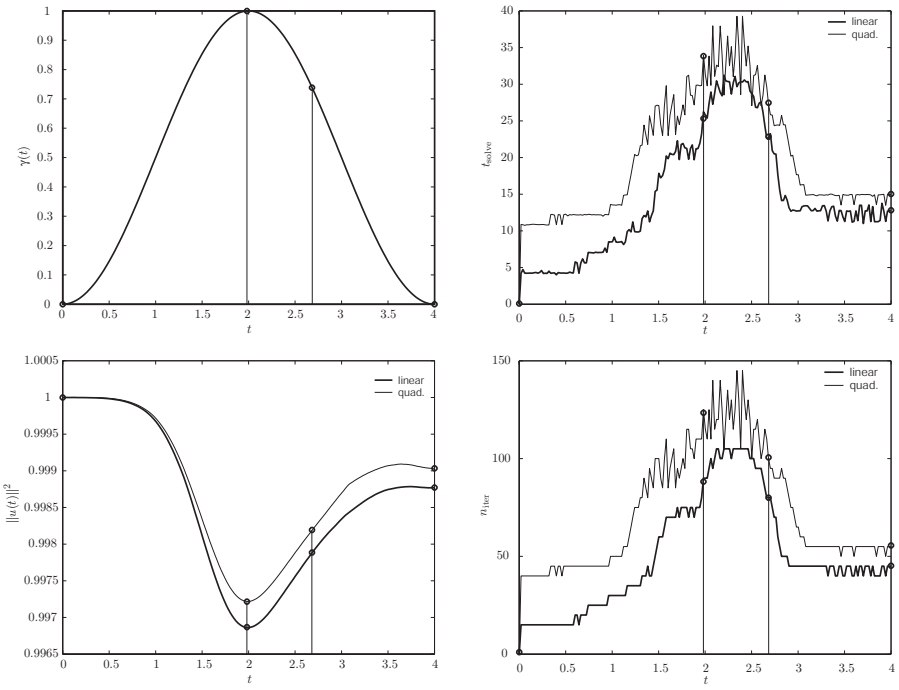

Figure 7.4: Various simulation data as function of time: The magnetic field (top left), the square norm (bottom left), the number of iterations per linear system (top right) and the time per linear system (bottom right)

The leap-frog method for finite element methods (LF 0.0015) used roughly half the simulation time than the corresponding Crank-Nicholson simulation.

We will not show it here, but for larger time steps, solving linear equations in the Crank-Nicholson method takes more time, indicating that the closeness of the previous solution plays some role on the performance of the time stepping. This is a subject for later studies.

The Crank-Nicholson method preserves the norm of the solution exactly, but as we employ iterative solvers there might be some fluctuations. The leap-frog scheme is not exactly unitary because we have violated the assumption we used when deriving the stability criterion by using the Crank-Nicholson integration for the first time step.[3] Fig. 7.3 shows the logarithm of the deviation from unity of the square norm of the numerical solution. Clearly, Crank-Nicholson preserves unitarity better, with variations around 10^{-6}. The leap-frog method shows clear fluctuations, but never larger than 10^{-3}. This is acceptable if we normalize the wave function before calculating physical quantities such as expectation values.

In summary, it seems that the Crank-Nicholson method can take bigger time steps while reproducing the same results as the leap-frog scheme. The explicit finite difference method is quick to implement, but one loses the advantages from the finite element method, such as geometry freedom, the mass matrix (which yields higher accuracy of the eigenvalues) and so on.

7.3.3 Comparing Linear and Quadratic Elements.

We have several parameters that affect the finite element method, most important is perhaps the element order. Increasing the element order will increase the number of

[3]The "hole" in the LF 0.0015 curve is due to missing data because of a mishap during the simulation.

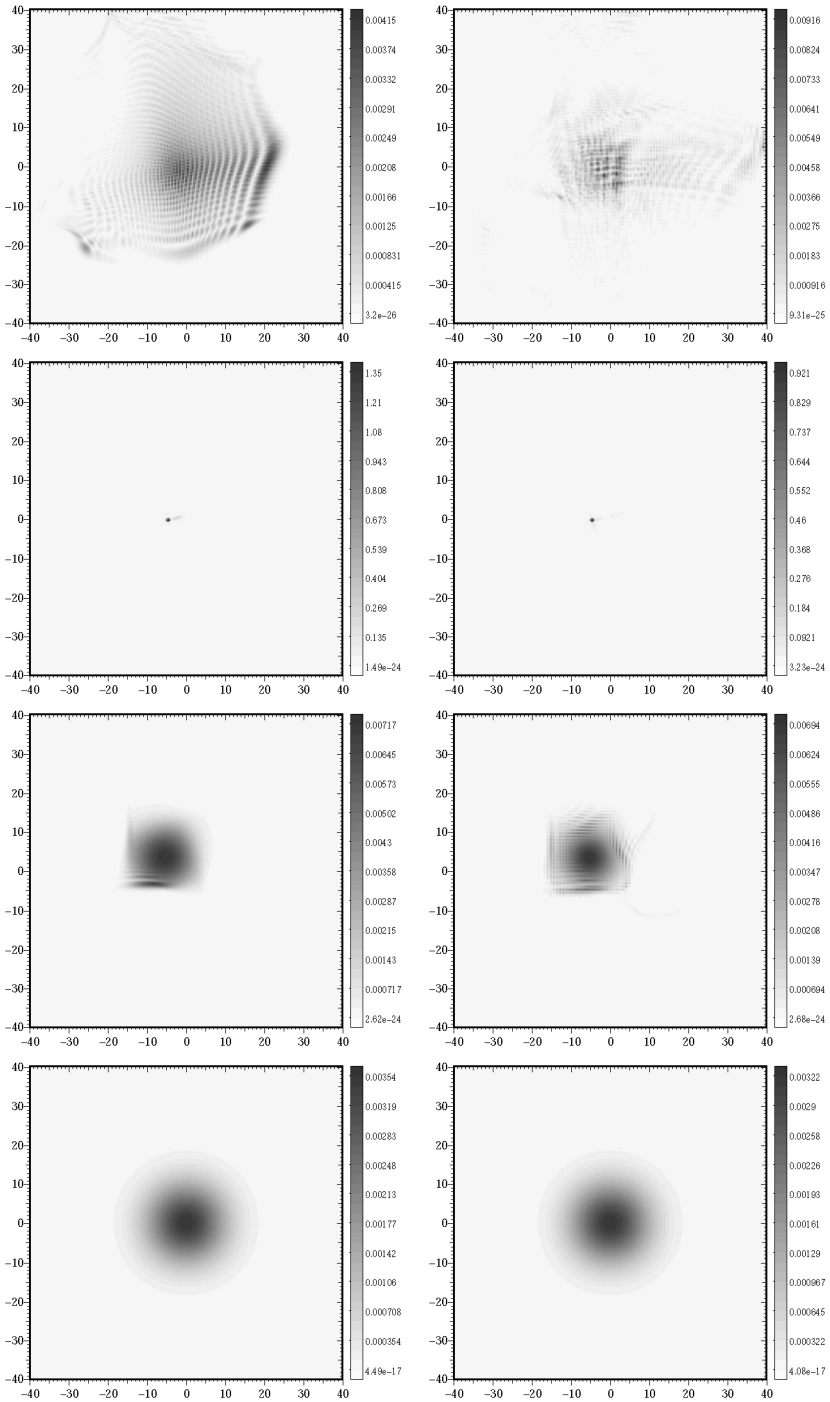

Figure 7.5: Stills from simulation with linear (left) and quadratic (right) elements. The time levels are (from bottom to top) $t = 0.0000$, $t = 1.9825$, $t = 2.6825$ and $t = 4.0000$.

non-zeroes in the matrices of the problem, thereby increasing the time needed for each iteration in the linear solver. On the other hand, increasing the element order (while keeping the number of grid points fixed) may affect the accuracy of the solution.

We will do a simulation with a varying magnetic field with linear and quadratic elements, respectively, to check if they give very different results. If they do, then one or both of the, experiences loss of numerical precision, because both simulations certainly cannot be correct. This will indicate whether or not the Schrödinger equation is sensitive to the order of the elements.

There are of course many more aspects to study, such as the grading of the grid, the choice of geometry and so on.

Again a square grid was used. This time the domain was given by

$$\Omega = [-40, 40] \times [-40, 40],$$

with $n = 200$ subdivisions along each side. Two simulations with linear elements and quadratic elements, respectively, were performed. Thus, we had $200^2 = 40,000$ elements in the linear case and $100^2 = 10,000$ elements in the quadratic case. The magnetic field parameters were $\omega = \delta = 0$ and $\gamma_0 = 1$ and the initial condition was a Gaussian with

$$\boldsymbol{x}_0 = (0,\, 0), \quad \boldsymbol{k} = (0,\, 2), \quad \text{and} \quad \boldsymbol{\sigma} = (10,\, 10).$$

The Coulomb attraction was turned off; hence we had a free particle with an applied magnetic field. The time interval was $t \in [0, 4]$ with $\Delta t = 0.0025$. We used the Crank-Nicholson scheme in both cases.

The only differing setting was the element type. Hence, if the simulations agree we must conclude that the Schrödinger equation appear insensitive to the element order (even though only two experiments is too little to make decisive conclusions.) If the simulations disagree, we must conclude that the element order do have some importance.

The particular system and initial condition we tested turned out to show very interesting dynamics: The wave packet moves like a free particle as expected until the magnetic field starts to become significant. The wave packet then shrinks very quickly, concentrating into almost a single point. As the magnetic field diminishes it spreads out again. When concentrated in an area covering only a few elements, all the information from the original wave packet is reduced to only a handful of numbers, i.e., the components of the numerical wave function. Hence, there is loss of information. This means that we are on thin ice when studying the wave function dynamics *after* this event.

Animations showing the development of the probability density can be reached from Ref. [5]. Fig. 7.5 shows some still images of the probability density for the two simulations at key points in the development. Fig. 7.4 shows the development of the square norm of the wave function, the magnetic field $\gamma(t)$ and the time and number of iterations spent at solving the linear systems for each time step. The time levels at which we have plotted the probability density is showed as circles on the graphs.

The qualitative behavior of the probability density is similar for linear and quadratic elements, at least before the "collapse" of the wave function into approximately a single point around $t = 2.6825$. A noteworthy difference is that the quadratic element simulation tend to develop high-frequency spatial oscillations near the edge of the wave packet. These can be seen at $t = 1.9825$. If this is a physical effect or a numerical effect is not easy to say, but certainly deserves an investigation. If this is a numerical effect it indicates that (at least for this magnetic Hamiltonian) that higher order elements introduce noise. On the other hand, if it is a physical effect, it indicates that using higher order elements captures features of the wave function that needs a refining of the grid in the finite difference method.

The wave functions collapse approximately at the same time and place as can be seen from the figure. After the collapse the wave functions do not bear resemblance to each other, except for spreading rapidly. The spreading is easily understood in terms of Heisenberg's uncertainty principle. Highly localized wave packets have a high uncertainty in the momentum, making the packet spread fast. The fact that the wave functions are so different means that the error must be large in either or both simulations after the collapse.

Turning to the graphs in Fig. 7.4, the norm is seen to be approximately conserved for both linear and quadratic elements, but shows a tendency to drop somewhat when the magnetic field is strong. The time spent on solving linear systems and the corresponding number of iterations show the opposite behavior. When the magnetic field is strong several times more iterations are needed for solving the systems than $\gamma = 0$. The time of collapse of the wave function (where the wave function changes very rapidly) this does not seem to behave differently with respect to this.

7.3.4 A Simulation of the Full Problem

As a final study we will do a numerical experiment with the complete physical system. We will start with a Gaussian wave packet in a Coulomb field and add an ultra-short laser pulse. We will not discuss the accuracy of this simulation, for that we need much more time and space. However, we will describe the results qualitatively and generate a motion picture of the resulting "trajectory" of the wave packet. We will do the simulations for three different parameter sets for the time integration for comparison.

The magnetic field has parameters

$$\gamma_0 = 1, \quad \omega = 1.23, \quad \delta = 0.13$$

and is shown in Fig. 7.7. The initial condition has parameters

$$\boldsymbol{x}_0 = (-5,\, -5), \quad \boldsymbol{k} = (2,\, 0), \quad \boldsymbol{\sigma} = (3,\, 3).$$

The initial condition along with other stills from one of the simulations is shown in Fig. 7.6.

We will use three representative time integration schemes.

- We use the leap-frog method with $\Delta t = 0.001$ and nodal point integration, i.e., we use the finite difference method. The time step is chosen based on $h = 80/150$ which gives

  $$\Delta t < 0.03555.$$

  We assume that the Coulomb potential has no eigenvalue of magnitude greater than the maximum energy for the particle-in-box. This is reasonable, since the ground state has magnitude $|\epsilon_0| \leq 4$ and since the system "looks like" a particle in box for very highly excited states. Due to the changing magnetic field we choose Δt even lower, namely

  $$\Delta t = 0.001.$$

  We refer to this simulation as LFL 0.001 in the figures.

- For the second simulation we use the Crank-Nicholson method with larger time step, viz.,

  $$\Delta t = 0.02.$$

  We use linear elements and Gaussian quadrature and refer to the simulation as CN 0.02.

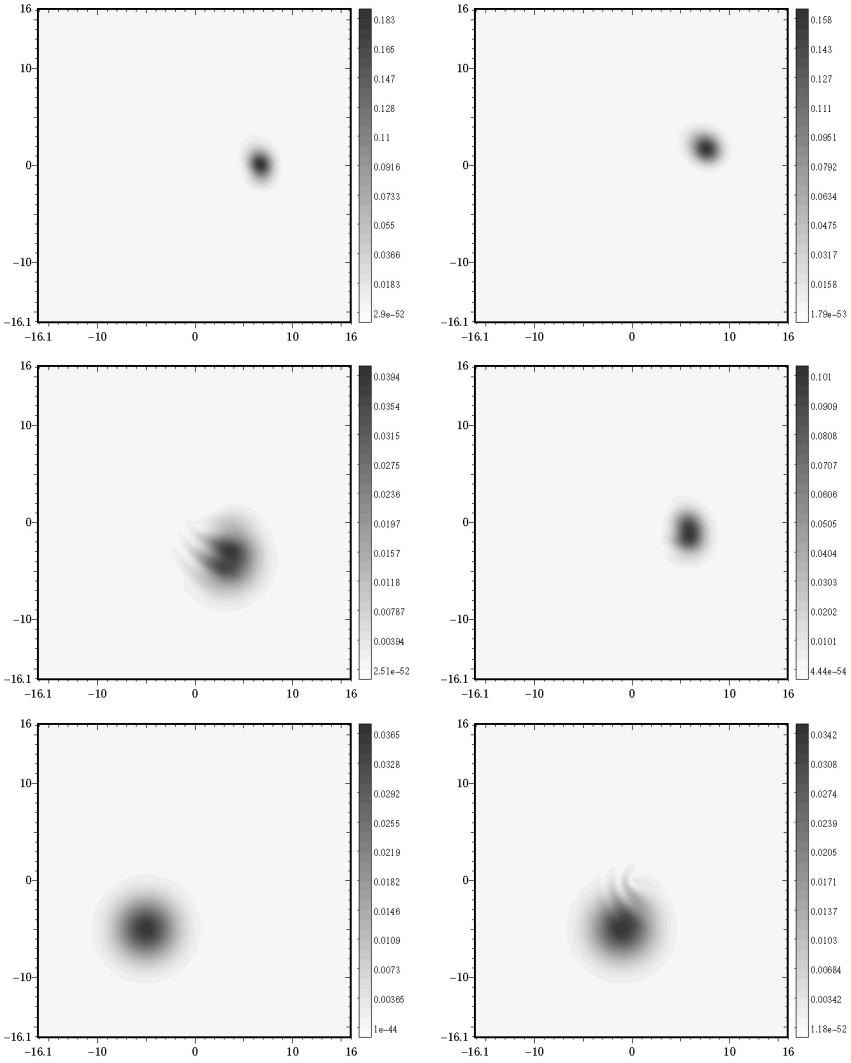

Figure 7.6: Stills from simulation. Time levels are $t = 0.00$ (bottom left), $t = 0.98$ (bottom right), $t = 1.98$ (middle left), $t = 2.98$ (middle right), $t = 3.48$ (top left) and $t = 3.98$ (top right).

– The third and last simulation also uses Crank-Nicholson but with quadratic elements. We refer to the simulation as CN 0.02 quad.

Stills from the LFL 0.001 simulation are shown in Fig. 7.6. Animations of all three simulations can be reached from Ref. [5]. The different simulations yielded qualitatively similar results for the probability density. Actually it is hard to spot any difference at all. Furthermore, we can not see any high-frequency oscillations in the quadratic elements case as we did in Fig. 7.5. In Fig. 7.7 we have graphed the magnetic field γ, the square norm $\|u\|^2$ of the wave function, the energy $\langle H \rangle$ and the expectation value $\langle \boldsymbol{x} \rangle$ of the position as function of time.

All the plots indicate that the Crank-Nicholson simulations behave very similar to each other. Furthermore, they are qualitatively different from the leap-frog simulation in all plots but the energy plot.

The square norm has large fluctuations in the Crank-Nicholson scheme. The fluctuations clearly follow the time development of the magnetic field. Recall from section

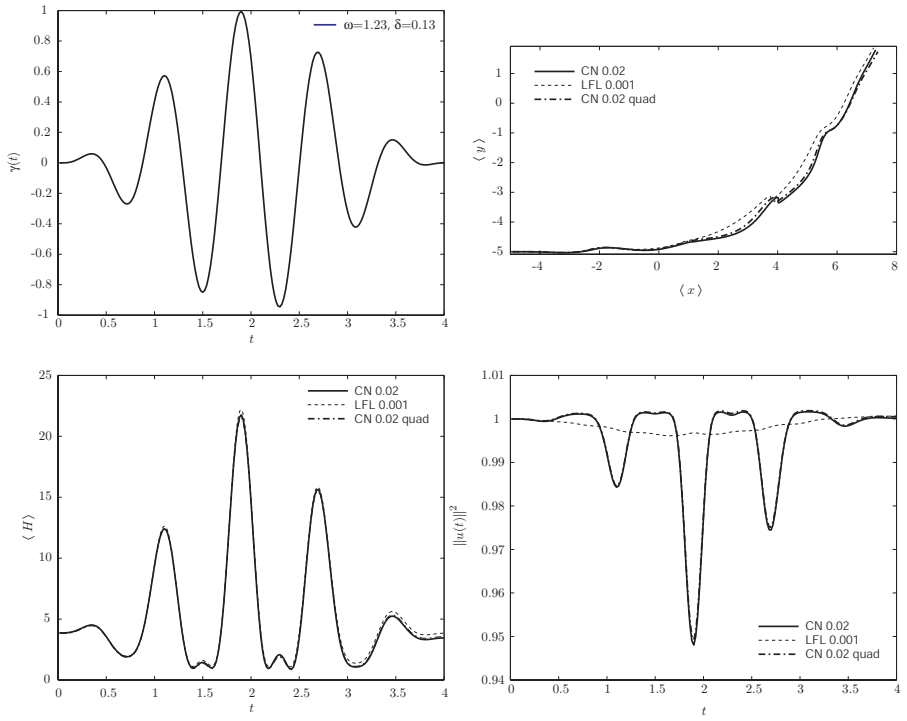

Figure 7.7: Plots of dynamical quantities: The magnetic field (top left), the mean position (top right), the mean energy (bottom left) and the square norm (bottom right).

4.5.4 that the norm should experience $\mathcal{O}(\Delta t^3)$ fluctuations depending on $\partial H/\partial t$ for both methods. This is clearly the case for all the simulations, and in fact the smaller fluctuations for the leap-frog method must be due to the smaller time step.

Even though we have fluctuations in the norm, the expectation value $\langle H \rangle$ seems insensitive to these fluctuations as all three simulations give the same result, both qualitatively and quantitatively.[4] The energy fluctuates and follows the "beat" of the magnetic field. This indicates that violating the strict unitarity not necessarily destroys the physical content of the wave.

In the graph of the expectation value $\langle \boldsymbol{x} \rangle$ (and in the animations) the wave packet actually *reverses* its motion, indicating that the kinetic energy should have local minima. This is clearly seen (if we assume that the potential energy is neglible). The changing direction of the magnetic field (as γ goes from positive to negative or vice versa) means that a changing torque is exerted on the particle. This classical line of thinking is of course possible due to the correspondence principle and indicate that our simulations behave according to the laws of quantum mechanics.

The expectation value $\langle \boldsymbol{x} \rangle$ of the position shows that the wave packet is being deflected by the Coulomb potential, similar to a classical system. An interference pattern is also observed connecting the wave packet with the origin with long "fingers" of probability.

At the end of the simulation we observe something interesting. Similar to the system in which the Coulomb potential was absent the wave packet starts to shrink. Thinking classically we would expect the opposite to happen, i.e., that the packet would spread out due to the uncertainty in the linear momentum. But in our simulations it seems that the magnetic field's effect on the particle is to localize it. This localization feature

[4]The program output is not normalized, but this is done prior to plotting.

is not present in the pure Coulomb system, see Ref. [26]. An additional animation showing a simulation in which $\gamma = 0$ is also found in Ref. [5].

We have obtained these qualitative results in different systems with different numerical integration schemes, indicating that there might be some interesting physics behind this behavior as well. Studying wave packet dynamics in magnetic fields should be an interesting future project, also because it poses a numerical challenge if the wave collapses into a very small area as in Fig. 7.5.

From chapter 6 we know that the time development of the wave function (at least for a time independent Hamiltonian) is given directly in terms of the eigenvalues and eigenvectors of the system. If the discretization reproduces the eigenvalues then the time development of the exact system will be accurately reproduced. On the other hand, if the eigenvalues are wrong, the time development of the discrete system will not reflect the exact development. Even though our different numerical methods all yielded very similar results we cannot say that it reflects the *true* time development. For that we need to know in some way whether the eigenvalues are correct. In the present case we could do a diagonalization of the Hamiltonian over the current grid for $\gamma = 0$. This would give further indications of a good numerical method and simulation.

7.4 Discussion

Even though the experiments were limited in extent we have come across several important indications on the performance and behavior of the numerical methods. Can we draw any conclusions from our work?

The most important benefit of the Crank-Nicholson method over the leap-frog scheme is *stability*. The method is stable for much larger time steps than the leap-frog method. The time step required to reproduce the accuracy of the leap-frog scheme (with finite differences in space) is much larger than the stability criterion for the explicit scheme, making this benefit even more important.

On the other hand, the *ease of implementing* the leap-frog scheme with finite differences makes it very attractive. The linear systems in the finite element method is cheaper to solve with the leap-frog method, which is an important fact.

As for the spatial discretization, finite differences are without doubt the *fastest* to implement and run. But we have seen that the finite element approach reproduces eigenvalues that are qualitatively more correct than the finite difference method. Furthermore, there might be something to gain from using higher-order elements. We saw that the time development of a wave function with quadratic elements was qualitatively different than with linear elements. Then again, the systems of linear equations are harder to solve.

This takes us to the next important fact we have learned. Choosing the linear solver appropriately is very important. The efficiency of the implicit time stepping improves *drastically* if we choose a somewhat more sophisticated linear solver (i.e., iterative methods) than the usual banded Gaussian elimination that is used in most two-dimensional approaches. In three dimensions it becomes vital. With the generalized minimal residual method the time spent on solving linear systems scaled linearly with the matrix dimension, and this is optimal. Fine-tuning the choice and parameters of the linear solver is an interesting future project.

Our simulations exclusively used a Gaussian wave packet for initial condition even though we implemented the possibility of using the eigenvectors of the Hamiltonian over the current grid. This was mostly due to limitations in computing time when the submission deadline approached. If the time-dependent simulations done here were interesting, using for example the ground state of the hydrogen atom over a properly graded and fine-tuned grid would be really cool, to put it that way.

Chapter 8

Conclusion

This short chapter summarizes the results and findings of this *cand. scient.* project. Along the way valuable experience with the numerical methods has been gained, and some ideas for possible future work have emerged as well. The focus has not been as much on physics as on numerical methods, but this is not a limitation. On the contrary, I believe that the experience gained with the numerical methods may serve as an invaluable basis when complicated simulations and work is to be done.

The Eigenvalue Problem. The eigenvalue problem of the finite element discretized Hamiltonian is closely related to the development of the time-discrete wave function, as we have seen. The eigenvalues dictate the evolution, but apart from this the eigenvalue problem has few consequences for the time dependent simulations. Of course, we still have the usefulness of being able to analyze the time dependent state with respect to the spectrum of the Hamiltonian, but this is not directly linked with the numerical methods.

On the other hand, the eigenvalue problem by itself is very interesting to consider. Many questions one might have concerning a physical system can be answered in terms of the allowed energy levels. Take for example the hydrogen atom, in which the spectrum tells us the frequency of light that is needed to be able to excite or ionize the electron from its ground state. By exploiting the flexible geometry specification in the finite element method we are able to concentrate our numerical work in areas where the potential energy varies rapidly. We saw examples of this when trying to find the spectrum of the two-dimensional hydrogen atom. With other methods such as finite difference methods, the choice of geometry is very limited.

Because of the flexible geometry in the finite element method, the possibility of studying quantum chaotic systems was mentioned in chapter 6. This implies the study of highly excited states, i.e., eigenvectors of the Hamiltonian far from the ground state. We saw that for the particle-in-box, the finite element method produced eigenvalues that *qualitatively* followed the analytic eigenvalues better than the finite difference method. If this is a general feature of finite element methods (or variational methods in general) they must be superior when studying such systems. Furthermore, if the energy of the wave function in a time dependent simulation is very high, we should use whatever method that most faithfully reproduces the higher eigenvalues. Hence, further studies of the finite element eigenvalue problem is very attractive, as it has implications that are far-reaching.

It must be mentioned that our approach to the eigenvalue problem was rather naïve and based on the idea that we could learn something about the time development. Very sophisticated methods for finding eigenstates of the Hamiltonian called *adaptive finite element methods* have been developed and applied to some classes of systems, even though they are not completely general. Doing a critical study of adaptive methods

could prove interesting, perhaps also for the time development of the wave function.

As starting point for the work on eigenvalue problems one could perform an extension of the one-dimensional analysis in section 6.4 to include higher order elements in several dimensions. This will then indicate lines of action with respect to time-integration as well.

An additional study is the energy levels for the two-dimensional hydrogen atom at intermediate magnetic fields. We discussed this system to some extent, finding interesting qualitative results. Extending this analysis to for example spatially varying magnetic fields could be interesting, as well as doing one-dimensional calculations on the radial equation. For the tree-dimensional atom this has been done with finite element methods as early as in the eighties, see Refs. [52, 53].

We have also mentioned *gauge invariance* of the eigenvalues and eigenvectors as an important physical and numerical concept. Establishing results concerning this may prove fruitful. This has never been properly done before.

Time Integration Methods. We have just scratched the surface of the wide range of time integration methods that exist for classical Hamiltonian systems and the Schrödinger equation, see Ref. [36]. A concept that becomes more and more emphasized in numerical analysis of ordinary differential equations is *geometric integration*, i.e., integration methods that preserve certain qualitative features of the differential equations, thereby incorporating stability in a natural way. It is clear that a thorough insight into what methods exist and to what kinds of systems they have been applied to would form an invaluable basis for future work on solving the time dependent Schrödinger equation. Trying out these methods with different spatial discretizations and physical systems is very interesting and should form interesting projects.

As for the methods studied in this text, there are still some unanswered questions. Most important is perhaps the question of *gauge invariance*, which in the literature is not even asked. For the time dependent methods this must be viewed in conjunction with the spatial discretization, since both spatial and temporal degrees of freedom enter the problem. A gauge invariant integration scheme will provide more accurate results in a simulation, simply because the original formulation is gauge invariant and that the chosen gauge is arbitrary in principle. If we must "choose the correct gauge" in order to be able to trust our results in each case, we automatically have a lot more parameters to adjust before starting the numerical simulations, and we must be able to find out what gauge is best in the first place.

In the extension of this analysis we may study what kinds of schemes that in general might prove gauge invariant, such as implicit schemes, explicit schemes and unitary schemes, if any.

We stress that gauge invariance is an area of numerical analysis that never before has been studied properly. On the other hand the concept of covariance of numerical methods is a field of research in progress.

Linear Solvers and the Schrödinger Equation. A lot of simulation time was spent on solving linear systems. Linear systems pervade most computational scientist's work, and having a thorough knowledge of what solver to use and when to use it really distinguishes an experienced practitioner from the crop. In this thesis we simply used a fixed solver because it worked and spent reasonably small time at doing its job.

The Schrödinger equation and the particular time integration method yield linear systems of a particular kind. The leap-frog scheme (and any other explicit scheme) have the mass matrix M as coefficient matrix. The Crank-Nicholson scheme have a coefficient matrix $M + i\Delta t H/2$. As we saw the solution of the latter systems required longer time than the former, indicating that knowing what method to use is important.

Are there any way of deciding what solver is the best for the Schrödinger equation? What about modifying an existing solver to fine-tune it for the Schrödinger equation?

Such questions also become increasingly important if higher-dimensional systems are to be solved, in which the matricial dimensions approach the limits of what is possible to handle with today's computer technology.

The technique of *preconditioning* the linear systems in order to improve the convergence rate for the Schrödinger equation is also a topic which could be interesting to analyze further, see Ref. [34].

As mentioned in the concluding remarks of chapter 6, both Diffpack and ARPACK has supports parallelization. Studying the parallelized version of the numerical methods will certainly make us able to do much more detailed simulations and give better physical results.

Physical Applications. In this thesis we have focused on two-dimensional systems. They were chosen for several reasons, most prominent were perhaps their ease of visualization and their intermediate difficulty of implementation. We can use moderately large grids and obtain accurate results, something that would be difficult in a three-dimensional problem. Two-dimensional systems also arise in a variety of applications, such as solid-state physics and plasma physics.

In Ref. [54] several atomic systems are considered, in particular Rydberg atoms (i.e., highly excited atoms) in electromagnetic fields. Several interesting conclusions and results are presented, several of which could prove interesting to reproduce and study in further details, such as the excitation of Rydberg atoms with laser pulses.

With the finite element method and accurate integration schemes we may study these aspects in a systematical manner and provide further insight into an exciting (no pun intended) area of physics. Atomic physics describe systems that we actually can create and control in a laboratory, providing basic insight into quantum mechanical systems and also possible technological advances in the future.

In the extension we may also study systems of ions trapped in arbitrary geometrically shaped potentials, such as anharmonic and anisotropic harmonic oscillators, quadrupole potentials, Paul traps and even quantum wires and nanotubes. See Ref. [51] for details.

We could also study the Schrödinger equation on more exotic manifolds, such as a graph with several one-dimensional manifolds that meet at common points (see Ref. [55]), a curved two-dimensional manifold such as a sphere and so on. Point-interaction Hamiltonians (i.e., Hamiltonians with δ-function potentials) could also be attacked in a direct manner with finite element methods in which the singularities are integrated out of the formulation, leaving a well-defined matrix equation instead.

Closely related to the point-interaction Hamiltonians are the systems from solid-state physics mentioned in section 3.2.1. These systems encourage in a natural way the use of the finite element methods. If the system is very regular we may find the exact solution, but if the particles in the model are displaced in a random fashion the behavior of the system is largely unknown. On the other hand we know that irregularities in such systems are responsible for many well-known phenomena, such as super-conductance at low temperatures.

The Schrödinger equation is linear, but in several areas of physics *non-linear* variants of the equation arise. Bose-Einstein condensates can be described by the Gross-Pitaevskii equation, see Ref. [54]. This equation has an extra term proportional to $|\Psi(x)|^2$ and describes the effective one-particle wave function for the interacting bosons. Solving the time-dependent Gross-Pitaevskii equation is a challenge and will certainly provide valuable insight into both the physics and the numerics of linear and non-linear systems.

Appendix A

Mathematical Topics

This appendix discusses some mathematical and technical points in the text in detail.

For a thorough discussion on linear operators on Hilbert spaces and in quantum mechanics, see [56]. The physics textbook [20] contains mathematically oriented appendices easier to grasp at first reading.

A.1 A Note on Distributions

The most famous example of a distribution is perhaps the Dirac distribution, also (improperly) called the δ-function. Originally, this was defined as the "function" $\delta(x)$ satisfying

$$\delta(x) = 0, \quad \forall x \in \mathbb{R}, x \neq 0,$$

and

$$\int_{-\infty}^{+\infty} \delta(x)\, \mathrm{d}x = 1.$$

If $\delta(x)$ is a function, then its value at $x = 0$ must be infinte in order to make the integral different from zero.

The Dirac δ-function is an example of a *distribution.* Distributions play a fundamental role in the mathematical theory of finite elements as well as quantum mechanics. The rigorous definition is beyond the scope of this text, see Ref. [16] for an introduction. Roughly speaking they represent a way to take a weighted average of a function.

From the "definition" of the Dirac distribution we must have

$$\int \delta(x) f(x) = f(0),$$

and so

$$\delta : L^2 \to \mathbb{C}, \quad \delta(f) = (\delta, f) = f(0).$$

Here, we must assume that $f(0)$ is well-defined, but it is not! Any element f in L^2 is considered identical to elements g whose function values differ at a set of (Lebesgue) measure zero. Thus, the δ-distribution is not well-defined on the whole space L^2.

A.2 A Note on Infinite Dimensional Spaces in Quantum Mechanics

Quantum mechanics use linear algebra in Hilbert spaces as framework for the theory. More specifically, the Hilbert spaces employed are infinite-dimensional and separable.

Separable means that one may always find a countable orthonormal basis. Every finite-dimensional vector space is separable.

As stated in section 1.3, the Hilbert space for one particle is

$$\mathcal{H} = L^2(\mathbb{R}^3) \otimes \mathbb{C}^{2s+1},$$

where s is the intrisic spin of the particle. The space L^2 is the square-integrable functions on $\mathbb{R}^3$ into the complex numbers, viz.,

$$L^2(\mathbb{R}^3) := \{f : \mathbb{R}^3 \to \mathbb{C} \quad : \quad \int |f|^2 < \infty\}.$$

Usually, physicists work with $\mathcal{H}$ as if it is a finite-dimensional space, in the sense that Hermitian operators always may be diagonalised (see below) and that operators are well defined for every vector in $\mathcal{H}$. This is a rough simplification of the real situation. Physicists also work with the elements as if they have well-defined values at each point in space. Because of this, quantum mechanical calculations are actually often merely *formal* calculations, or "juggling with symbols." For example, consider the point-interaction Hamiltonian in one-dimension, viz.,

$$H = -\frac{\partial^2}{\partial x^2} + \mu\delta(x_0),$$

where μ is some coupling constant. This is not an operator on L^2! First, the kinetic energy term is not defined on the whole space, and δ is not an even an operator, much less a function.

Consider the momentum operator in one dimension. We write this as

$$P = -i\hbar\frac{\partial}{\partial x},$$

but strictly speaking this is an abuse of notation and not correct. There are many vectors (i.e., functions) in $\mathcal{H}$ that cannot be differentiated, i.e., the operator is not defined for all vectors. (It is not even defined on a closed subspace of $\mathcal{H}$, since a sequence of differentiable functions not necessarily converges to something differentiable.)

If we on the other hand consider the *weak derivative* of an element f in L^2, the operator is well-defined. If f happens to be differentiable in the classical sense, the weak derivative is the same as the classical derivative. This is why physics calculations with the momentum operator usually pose no difficulties.

A.3 Diagonalization of Hermitian Operators

As discussed in chapter 1, quantum mechanical observables are represented as linear operators A on the Hilbert space $\mathcal{H}$. All observables are Hermitian operators, that is $A^\dagger = A$. This is equivalent to the requirement:

$$(\Psi, A\Phi) = (A\Psi, \Phi),$$

for all $\Psi, \Phi \in \mathcal{H}$.

Note that we like to consider operators not defined for *all* vectors, such as the momentum operator or the position operator. Many of these (including the momentum operator) do not have an orthonormal basis of eigenfunctions in L^2.

Diagonalization of an operator A means finding an orthonormal basis of vectors ϕ_n and a set of scalars a_n such that

$$A\phi_n = a_n\phi_n, \quad n \in I,$$

where I is some set. Usually it is a countable set, i.e. a subset of the natural numbers, or it is uncountable as a subset of the real numbers.

All finite-dimensional (complex) Hilbert spaces are isomorphic to $\mathbb{C}^n$ where $n = \dim(\mathcal{H})$. Thus linear operators become $n \times n$ matrices, and Hermitian operators in finite-dimensional spaces may in this way always be diagonalized, since finite-dimensional matrices may always be diagonalized.

For inifite-dimensional Hilbert spaces such as L^2, the situation is not so simple. Many operators may be diagonalized, but others can not. Take for example the operator

$$P = -i\frac{\partial}{\partial x},$$

acting on $L^2(\mathbb{R})$-vectors. Writing out the eigenvalue equation (using the classical derivative!) yields

$$-i\frac{\partial \phi}{\partial x} = p\phi(x).$$

This differential equation has the solutions

$$\phi_p(x) = Ae^{ipx}.$$

These solutions are however not vectors of $L^2(\mathbb{R})$, since

$$\int_{-\infty}^{\infty} e^{ipx}\ \mathrm{d}x$$

does not exist. As another example, consider the operator X; multiplication by x. The eigenvalue equation becomes

$$x\phi(x) = x'\phi(x),$$

whose only sulution is

$$\phi(x) = B\delta(x - x').$$

This is not a function!

Both operators considered above are fundamental in quantum mechanics. Even though their "eigenvectors" are not proper vectors of $\mathcal{H}$, it is very useful to consider them as such. Physicists use the name eigenvectors for the above solutions, even though it is mathematically incorrect. Actually, they are *distributions*.

Do the "eigendistributions" form a basis for $\mathcal{H}$ in some way? In the cases of P and X they do, since any square integrable function $\psi(x)$ obeys

$$\begin{aligned} \psi(x) &= \int \delta(x - x_0)\psi(x_0)\ \mathrm{d}x_0, \\ \text{and}\quad \psi(x) &= \int e^{ipx}\phi(p)\ \mathrm{d}p, \end{aligned}$$

so that there exist a superposition (i.e., an integral) of the eigendistributions that equals $\psi(x)$. The above equations are nothing more than the formal definitions of the Fourier transform and the Dirac δ-function, respectively. In most cases physicists encounter the eigendistributions yields a basis for $\mathcal{H}$ in this sense.

A.4 The Fourier Transform

Given a sufficiently nice function $f : \mathbb{R}^\aleph \to \mathbb{R}$ we may define the *Fourier transform* as:

$$g(\boldsymbol{k}) = \mathcal{F}[f] := \frac{1}{(2\pi)^{n/2}} \int f(\boldsymbol{x})e^{-i\boldsymbol{k}\cdot\boldsymbol{x}}\ \mathrm{d}^n x. \tag{A.1}$$

This transforms the function $f(\boldsymbol{x})$ into another function $g(\boldsymbol{k})$, and the transformation may be inverted, viz.,

$$f(\boldsymbol{x}) = \mathcal{F}^{-1}[g] := \frac{1}{(2\pi)^{n/2}} \int g(\boldsymbol{k}) e^{i\boldsymbol{k}\cdot\boldsymbol{x}} \, \mathrm{d}^n k. \tag{A.2}$$

In other words: A function $f(\boldsymbol{x})$ may be viewed as equivalent to $g(\boldsymbol{k})$ and conversely. Furthermore, $f(\boldsymbol{x})$ is the Fourier transform of $g(-\boldsymbol{k})$, as may easily be seen.

There is a strong connection between the Fourier transforms and quantum mechanics. The inverse Fourier transform may be interpreted in the following way. The function $f(\boldsymbol{x})$, a function of space coordinates, is represented as a superposition of plane waves $\exp(i\boldsymbol{k}\cdot\boldsymbol{x})$ of wavenumber $\boldsymbol{k}$ with coefficients $g(\boldsymbol{k})$. It is the description of the spatial wave function in the basis of plane waves.

A.5 Time Evolution for Time Dependent Hamiltonians

Consider the time dependent Schrödinger equation, viz.,

$$i\frac{\partial \Psi}{\partial t} = H(t)\Psi(t),$$

where the Hamiltonian H (which is linear and Hermitian) may depend explicitly on time. We should really make some assumptions on what function space Ψ belongs to, but let us instead consider the calculations as formal.

Assume that the solution $\Psi(t)$ exists in an interval $t \in [0, T]$. Let $U(t)$ be the operator that takes the initial condition $\Psi(0)$ into $\Psi(t)$. As the Schrödinger equation is linear, U is a linear operator. This yields

$$i\frac{\partial U}{\partial t}\Psi(0) = H(t)U(t)\Psi(0).$$

This holds for all $\Psi(0)$ so that

$$i\frac{\partial U}{\partial t} = H(t)U(t)$$

with $U(0) = \mathbf{1}$ is the differential equation governing the solution operator (or the propagator.) Integration yields

$$U(t) = \mathbf{1} - i\int_0^t H(t_1)U(t_1) \, \mathrm{d}t_1. \tag{A.3}$$

This suggests an iteration process called *Picard iteration*. The first iteration is

$$\begin{aligned} U(t) &= \mathbf{1} - i\int_0^t H(t_1)[\mathbf{1} - i\int_0^{t_1} H(t_2)U(t_2) \, \mathrm{d}t_2] \, \mathrm{d}t_1 \\ &= \mathbf{1} - i\int_0^t H(t_1) \, \mathrm{d}t_1 + (-i)^2 \int_0^t \int_0^{t_1} H(t_1)H(t_2)U(t_2) \, \mathrm{d}t_1 t_2. \end{aligned}$$

The succesive iterations is given by replacing $U(t_n)$ with Eqn. (A.3). Performing infinitely many iterations then gives

$$U(t) = \sum_{n=0}^{\infty} (-i)^n \int_0^t \mathrm{d}t_1 \int_0^{t_1} \mathrm{d}t_2 \ \ldots \int_0^{t_{n-1}} \mathrm{d}t_n \, H(t_1)H(t_2)\ldots H(t_n).$$

If $[H(t_1), H(t_2)] = 0$ we can rewrite this as

$$U(t) = \sum_{n=0}^{\infty} \frac{(-i)^n}{n!} \int_0^t \mathrm{d}t_1 \int_0^t \mathrm{d}t_2 \ \ldots \int_0^t \mathrm{d}t_n \ H(t_1)H(t_2)\ldots H(t_n).$$

Unfortunately, we cannot. The Hamiltonians at different time levels may be arbitrarly different from each other. We therefore introduce the *time ordering operator* $\mathcal{T}$. For $H(t_1)$ and $H(t_2)$ it is defined by

$$\mathcal{T}[H(t_1)H(t_2)] = H(\max\{t_1, t_2\})H(\min\{t_1, t_2\}).$$

For an set of n time levels $S = \{t_i, i = 1, \ldots, n\}$ it is defined by

$$\mathcal{T} \prod_{t\in S} H(t) = H(\max\{S\}) \left(\mathcal{T} \prod_{t\in S'} H(t) \right),$$

where $S' = S - \max\{S\}$. We also define $\mathcal{T}[A + B] = \mathcal{T}A + \mathcal{T}B$. With this definition, we have

$$\begin{aligned} U(t) &= \sum_{n=0}^{\infty} \frac{(-i)^n}{n!} \mathcal{T} \int_0^t \mathrm{d}t_1 \int_0^t \mathrm{d}t_2 \ \ldots \int_0^t \mathrm{d}t_n \ H(t_1)H(t_2)\ldots H(t_n) \\ &= \mathcal{T} \exp\left[-i \int_0^t H(t) \, \mathrm{d}t\right]. \end{aligned}$$

This is the formal expression for the propagator in the case of a time dependent Hamiltonian. As we see, the expression reduces to

$$U(t) = \exp\left[-itH\right]$$

in the case of a time independent Hamiltonian.

Appendix B

Program Listings

B.1 DFT Solver For One-Dimensional Problem

This is the Matlab source code for the simulations in chapter 4. The code is very simple and easy to adapt to new problems.

B.1.1 fft_schroed.m

```
%
% Simple program for solving the time dependent Schroedinger eqn. in
% one dimension with the spectral method. The PDE is given by
%         i psi_t = H psi,  psi(t0) = phi.
% The hamiltonian H is
%         H = T + V = -del^2 + V(x)
% The exact solution is given by
%           psi(t) = U(t,t0) phi,
% where the propagator U is given as
%           U(t,t0) = exp( -i (t-t0) H )
% We employ a split-operator method in time, in which U(t0+tau,t0) is
% approximated by
%           U = DED, D = exp( -i tau/2 T ) and E = exp( -i tau V ).
% Note that T (and hence also D) is diagonal in the Fourier
% transformed representation, viz.,
%           T g(k) = -k^2 g(k),   g(k) = FT(psi(x)).
% Thus A becomes a diagonal op. with diagonal elements
%           D(k,k) = exp( -ik*k*tau/2 ).
% V is diagonal in position-rep., so
%           E(x,x) = exp( -iV(x)*tau ).
%
%

% solve equation on interval [xmin,xmax].
% Use N points with spacing h.
xmin = -30;
xmax = 30;
L = xmax - xmin;
N = 1024;
h = L/(N-1);
b = 2.5;  % half width of barrier
a = 28;  % height of barrier


% choose time step and end-of-simulation time.
tau = 0.00025;
t_final = 2.0;

plot_at_time = [0, 0.5, 1, 1.5, 2, t_final];
plot_colors = ['g', 'r', 'g', 'r', 'g', 'r'];


% make a gaussian initial cond. centered at x0 with initial momentum k0.
% note: kinetic energy <T> = k0^2.
% width of gaussian: sqrt(sigma2)
x0 = -10.0;
sigma2 = 4;
k0 = 5;
X = linspace(xmin, xmax, N);  % x-coordinates of grid points.
psi0 = exp( -(X-x0).*(X-x0)/sigma2).*exp(j*k0*X);
% also create a potential V.
V = zeros(1,N);
for i=1:N
    if (X(i)>=-b) & (X(i)<=b)
        V(i) = a;
    end;
end;

rel_energy = k0*k0/a;   % relative energy of initial wave packet


% set ICs.
% psi is the solution with DFT, psi_fd with FDM
```

```
psi = psi0;
psi_fd = psi0';


% define frequencies for A operator.
freqs = [linspace(0,N-1,N)] * 2*pi/L;
freqs(N/2+1:N) = freqs(N/2+1:N) - N*2*pi/L;
% define diagonal operators for split-operator scheme
D = exp(-j * tau * freqs.*freqs/2);
E = exp(-j * tau * V);

% build the finite difference operator as a sparse matrix.
e = ones(N,1);
FD = spdiags([e -2*e e], -1:1, N, N);
FD(N,1)=1;
FD(1,N)=1;
FD = -FD/(h*h);

hold off
newplot

plot(X,V/a,'b');

plot_indices = floor(plot_at_time/tau);

% propagate ...
for step = 0:round(t_final/tau)

    n = find(plot_indices==step);
    if ~isempty(n)
        hold on
        plot(X,rel_energy+real(conj(psi).*psi),plot_colors(n));
        plot(X,rel_energy+real(conj(psi_fd).*psi_fd),strcat(':',plot_colors(n)));
    end;

    % propagate the spectral method wave funcion psi.
    phi = fft(psi);
    psi = ifft( D.*phi );
    psi = E.*psi;
    phi = fft(psi);
    psi = ifft( D.*phi );

    % propagate the finite difference method wave function psi_fd.
    tempFD1 = j*tau/2*FD*psi_fd;
    tempFD2 = - tau*tau/4*FD*(FD*psi_fd);
    temp = psi_fd + tempFD1 + tempFD2;
    psi_fd = (E.*(temp'))';
    tempFD1 = j*tau/2*FD*psi_fd;
    tempFD2 = - tau*tau/4*FD*(FD*psi_fd);
    psi_fd = psi_fd + tempFD1 + tempFD2;




end;

xlabel('x');
ylabel('|\psi(x,t)|^2');
```

B.2 The *HydroEigen* class

This is the class definition of `HydroEigen`. It is not used directly in the main program but rather through the subclass `TimeSolver`

B.2.1 HydroEigen.h

```
#ifndef HydroEigen_h_IS_INCLUDED
#define HydroEigen_h_IS_INCLUDED
#include <FEM.h>                // includes FieldFE.h, GridFE.h
#include <TimePrm.h>
#include <DegFreeFE.h>
#include <Arrays_real.h>        // for MatDiag
#include <SaveSimRes.h>
#include <FieldFormat.h>
#include <FieldSummary.h>
#include <LinEqAdmFE.h>
#include <MatBand_Complex.h>
#include <IsOs.h>



//
// simple inner product integrand. inlined.
//

class InnerProdIntegrandCalc : public IntegrandCalc {
protected:
  Handle(FieldFE) u, v;
  NUMT result;

public:
  InnerProdIntegrandCalc() { result = 0; }
  virtual void setFields(FieldFE& u_, FieldFE& v_) { u.rebind(u_); v.rebind(v_); }
  virtual NUMT getResult() { return result; }
  virtual void integrandsG(const FiniteElement& fe) {
    real detJxW = fe.detJxW();
    NUMT uval = u->valueFEM(fe);
    NUMT vval = v->valueFEM(fe);
```

```
    result += conjugate(uval)*vval*detJxW;  // integrate and store in result.
  }

};

//
// simple integrand for <r>. inlined.
//
class IntegrandOfExpectation_r : public InnerProdIntegrandCalc {
public:
  IntegrandOfExpectation_r() { }
  virtual void integrandsG(const FiniteElement& fe) {
    real detJxW = fe.detJxW();
    NUMT uval = u->valueFEM(fe);
    NUMT vval = v->valueFEM(fe);
    Ptv(real) x  = fe.getGlobalEvalPt();
    real r = sqrt(x(1)*x(1) + x(2)*x(2));

    result += conjugate(uval)*vval*r*detJxW;  // integrate and store in result.
  }

};



// ************************************************************************
//
// Class declaration of HydroEigen -- a class that solves the time indep.
// öSchrdinger equation for the two-dimensional hydrogen atom with an
// applied magnetic field in two different gauges.
//
// ************************************************************************

class HydroEigen : public FEM
{
protected:
  // Properties that concerns geometry of the system
  // and the structure of the linear equations.
  Handle(GridFE)    grid;      // hnadle of the grid
  Handle(FieldFE)   u;         // a field over the grid
  Handle(DegFreeFE) dof;       // mapping: field <-> equation system

  // Properties that concerns matrices, vectors and fields
  // used.

  // Note: the LenEqAdmFE obj is not actually used for eqns in this class.
  // However it is a convenient means for building the element
  // matrix with makeSystem(*dof, *lineq);
  Vec(NUMT)          linsol;   // a vector that is used for linear eqns
  Handle(LinEqAdmFE) lineq;    // keeps linear equations Ax=b
  Handle(Matrix(NUMT))   M;    // mass matrix
  Handle(Matrix(NUMT))   K;    // element matrix
  Handle(SaveSimRes) database; // for dumping fields to disk

  // Parameters that are read from the menusystem and initialized
  // in scan().
  real gamma;   // strength of magnetic field
  bool nucleus; // coulomb on/off
  bool angmom;  // ang-mom op. on/off
  bool lump;    // lump mass matrix or not.
  real epsilon; // singularity tolerance

  String mat_type;  // chosen storage scheme for matrices.
  String gridfile;  // gridfile from menu system
  String gauge;     // indicates gauge. "symmetric" or "non-symmetric"

  int nev;                  // number ov eigenvalues/vectors to compute.
  bool store_evals;         // store the eigenvalues or not.
  int store_evecs;          // number of eigenvectors to store.
  bool savemat;             // save matrices or not.
  bool store_prob;          // store prob. density or not.
  bool arpack_solve;        // use arpack to solve system.
  bool *erase_log;          // array that indicates unknowns to be erased.
                            // when incorporating BCs.

  void *the_solver;         // a void pointer used to store the EigenSolver object.

  Os logfile;               // writes info to casename.log.
  String real_format;       // contains e.g. "%10.10g"

public:
  // Constructors, destructors and "compulsory" methods
  // for a FEM class.
  HydroEigen ();
  virtual ~HydroEigen ();
  virtual void adm    (MenuSystem& menu);
  virtual void define (MenuSystem& menu, int level = MAIN);
  virtual void scan ();
  virtual void solveProblem ();
  virtual void report ();
  virtual void resultReport ();

  // Helps keeping track of time spent on computations.
  time_t getElapsedTime();
  void reportElapsedTime() ;

  // Add a string to casename.log and stderr.
  void addToLogFile(const String& s);

  // Methods that erase rows and columns from matrices.
  void eraseRowAndCol(MatSparse(NUMT)& A, int k);
  void eraseRowAndCol(MatDiag(NUMT)& A, int k);
  void eraseRowAndCols(MatBand(NUMT)& New, MatBand(NUMT)& Old);

  // fillEraseLog() marks the degrees of freedom that corresponds to
  // boundary nodes, uses bool *erase_log declared above.
```

```
  void fillEraseLog();
  // enforceHomogenousBCs erases the rows and columns from the matrices
  // indicated by bool *erase_log.
  void enforceHomogenousBCs(Handle(Matrix(NUMT))& K, Handle(Matrix(NUMT))& M); // erase dofs stored in erase_log (
        shrink matrices)

  // Saves a matrix in Matlab compatible format. (Diffpack's own is not reliable...)
  void saveMatrix(MatSparse(NUMT)& A, const String& Aname, const String& fname);

  //Creates the mass matrix, lumped or not, in appropriate storage format.
  void makeMassMatrix2(Handle(Matrix(NUMT))& Dest, const Handle(Matrix(NUMT))& WithPattern);
  //Multiplies A with inverse of diagonal matrix D.
  void multInvMatDiag(Handle(Matrix(NUMT)) D, Handle(Matrix(NUMT)) A);

  // Method that calculated expectation values.
  NUMT calcExpectation_r(FieldFE& u, FieldFE& v);
  NUMT calcInnerProd(FieldFE& u, FieldFE& v);

  // Caluclate the (approximate) probability density. (u_j --> |u_j|^2)
  void calcProbabilityDensity(const FieldFE& u, FieldFE& prob, bool redim=false);

protected:
  // Computes the integrand in the FEM matrix formulation. Called in FEM::makeSystem().
  virtual void integrands (ElmMatVec& elmat, const FiniteElement& fe);
};


#endif
```

B.2.2 HydroEigen.cpp

```
#include <HydroEigen.h>
#include <ElmMatVec.h>
#include <FiniteElement.h>
#include <readOrMakeGrid.h>
#include <SparseDS.h>
#include <IsOs.h>
#include <time.h>
#include <MatSparse_Complex.h>

#include <RenumUnknowns.h>
#include <AMD.h>
#include <GibbPooleStockm.h>
#include <Puttonen.h>

#include <IntegrateOverGridFE.h>


#include "EigenSolver.h"


// ********************************************************
//
//  Class definition of EigenSolver. See also EigenSolver.h
//
// ********************************************************

// minimal constructor
HydroEigen:: HydroEigen () { erase_log = NULL; }

// destructor
HydroEigen::~HydroEigen () { if (erase_log) delete[] erase_log;  }

// adm
void HydroEigen:: adm (MenuSystem& menu) {
  SimCase::attach (menu);
  define (menu);

  // let user come up with some choices.
  menu.prompt();
  // initialize solver.
  scan();
}

//
// Set up menu system extensions.
//
void HydroEigen:: define (MenuSystem& menu, int level) {

  menu.addItem(level, "gridfile", "file or prepro command",
         "P=PreproStdGeom | DISK r=40 degrees=360 | e=ElmT3n2D nel=1000 resol=100");

  menu.addItem(level, "nev", "no of eigenvalues", "100");
  menu.addItem(level, "gamma", "magnetic field strength", "0.0");
  menu.addItem(level, "nucleus", "turn on/off coulomb term", "true");
  menu.addItem(level, "epsilon", "singularity tolerance", "-1.0");
  menu.addItem(level, "angmom", "turn on/off angmom term", "false");
  menu.addItem(level, "nitg", "number of itegration points (0 uses GAUSS_POINTS)", "0");
  menu.addItem(level, "lump", "lump mass matrix or not", "false");
  menu.addItem(level, "warp", "warp factor for grid", "1.0");
  menu.addItem(level, "scale", "scale factor for grid", "1.0");
  menu.addItem(level, "renum", "try to renumber nodes?", "true");
  menu.addItem(level, "savemat", "save matrices as Matlab files", "true");
  menu.addItem(level, "store prob density", "store probability density fields", "false");
  menu.addItem(level, "use arpack", "diagonalize with arpack or not", "true");
  menu.addItem(level, "gauge", "gauge", "symmetric"); // or "non-symmetric"

  menu.addItem(level, "store_evecs",
         "number of evecs to store", "-1"); // -1 means all...

  menu.addItem(level, "real format", "real format for streams used", "%10.10g");

  // These commands makes the different classes
```

```
  // fetch default settings from the menu system.
  // For example, the default solver type for linear
  // systems of equations handled by LinEqAdm(FE)
  // can be set in the menu, and these choices are
  // passed on when constructing solver objects.
  SaveSimRes:: defineStatic (menu, level+1);
  //LinEqAdmFE::defineStatic(menu, level+1);
  LinEqAdm::defineStatic(menu, level+1);
  FEM:: defineStatic (menu, level+1);
}


//
// Initialize simulator
//
void HydroEigen:: scan() {

  MenuSystem& menu = SimCase::getMenuSystem();

  // Get the matrix type.
  mat_type = menu.get("matrix type");
  // If savemat==true, the matrices of the ev-problem will be
  // saved in Matlab compatible files.
  savemat = menu.get("savemat").getBool();
  // If store_prob == true, then the probability density
  // for each eigenvector that is stored is also stored.
  store_prob = menu.get("store prob density").getBool();
  // If arpack_solve == true, then ARPACK will be called
  // to actually diagonalize the problem. Usually this is done.
  // but to compare the diagonalization times with e.g. Matlab or
  // LAPACK one might want to instead save the matrices and
  // skip ARPACK diagonalization.
  arpack_solve = menu.get("use arpack").getBool();
  // gauge indicates what gauge to use for the potentials.
  // Usually set to "symmetric".
  gauge = menu.get("gauge");
  if (!((gauge == "symmetric") || (gauge == "non-symmetric"))) {
    // unknown gauge
    s_e << "Unknown gauge! must be 'symmetric' or 'non-symmetric'. Now set to 'symmetric'." << endl;
    gauge = "symmetric";
  }


  //
  // Set up the logfile.
  //
  logfile.rebind( *(new Os(aform("%s.log", casename.c_str()), NEWFILE)) );

  //
  // Get the grid from the menu system.
  //
  gridfile = menu.get ("gridfile");
  grid.rebind (new GridFE());
  readOrMakeGrid (*grid, gridfile);

  //
  // If the grid is a disk, the first node is always
  // at the edge. We should really loop through the nodes
  // to be sure ... :)
  // r is only used when warping (grading) the grid.
  //
  real r = grid->getCoor(1).norm(); // get the radius of the disk, if it is a disk.

  // If renum == true, we renumber the nodes in the
  // grid to optimize the bandwidth of the matrix.
  // This is important (!) if we use banded matrices,
  // but not that important if MatSparse is being used.
  if (menu.get("renum").getBool()) {
    // try to renumber the nodes to optimize ...
    s_e << "Optimizing grid ..." << endl;
    Handle(RenumUnknowns) renum_thing;
    // not stable?? crashes for some disk grids. :(
    //renum_thing.rebind(new GibbPooleStockm());
    renum_thing.rebind(new Puttonen());
    renum_thing->renumberNodes(grid());
  }

  // Get grid modification parameters.
  // See thesis text for details on the
  // grading (warping) process.
  real warp = menu.get("warp").getReal();
  real scale = menu.get("scale").getReal();

  if (warp != 1) {
    s_e << "Warping grid with factor " << warp << " ..." << endl;

    for (int i=1; i<=grid->getNoNodes(); i++) {
      real d = pow(grid->getCoor(i).norm(), warp-1);
      grid->putCoor(grid->getCoor(i)*d*pow(r,1-warp), i);
    }
  }
  if (scale != 1) {
    s_e << "Scaling grid with factor " << scale << " ..." << endl;
    for (int i=1; i<=grid->getNoNodes(); i++) {
      grid->putCoor(grid->getCoor(i)*scale, i);
    }
  }


  // Savce the grid to .temp.grid.
  // Usually not necessary as dumping fields
  // will create casename.grid as well...
  Os gridf(".temp.grid", NEWFILE);
  grid->print( gridf ); // write warped/optimized grid to file
  s_e << "Saving the warped/optimized grid to .temp.grid ..." << endl;
```

```
// Get number of eigenvectors to find.
String nev_string = menu.get("nev");
nev = atoi(nev_string.c_str());

// nucleus turns on/off the Coulomb interaction
// in the Hamiltonian. gamma is the magnetic field.
// If lump==true, we lump the mass matrix.
// angmom turns on/off the term prop. to d/d phi in H.
nucleus = menu.get("nucleus").getBool();
gamma = menu.get("gamma").getReal();
lump = menu.get("lump").getBool();
angmom = menu.get("angmom").getBool();

// Get number of eigenvectors to store.
// A negative value automatically saves all vectors.
store_evecs = atoi(menu.get("store_evecs").c_str());
if ((store_evecs < 0) || (store_evecs > nev))
  store_evecs = nev;


// Init the simres database.
// We store fields (i.e., eigenvectors) with
// database->dump( field );
FEM::scan (menu);
database.rebind (new SaveSimRes());
database->scan (menu, grid->getNoSpaceDim());

// Init the field u and the dof.
// dof is actually not used in the simulator,
// needed in the DegFreeFE object.
u.rebind (new FieldFE (*grid,"u"));
dof.rebind (new DegFreeFE (*grid, 1));  // 1 for 1 unknown per node

// Init the linear system. (Holds matrix in eigenvalue formulation)
// FEM systems are not solved in HydroEigen, but the structure is needed.
lineq.rebind(new LinEqAdmFE());
lineq->scan(menu);
linsol.redim(grid->getNoNodes());
linsol.fill(0.0);
lineq->attach(linsol);

// Set up integration rules.
// This is somewhat tentative: A trapezoidal rule
// is set up if nitg>=2. This rule should ONLY
// be used with triangular elements.

s_e << "Setting up integration rules..." << endl;

String n_string = menu.get("nitg");
int n = atoi(n_string.c_str());
int N = n*(n+1)/2;

if (n>=2) {

  // set up trapezoidal rule.
  // NB: ONLY WORKS FOR TRIANGULAR ELEMENTS!

  s_e << "Using " << N << " integration points w/weights." << endl;
  s_e << "(Are you sure that your elements are triangular?)" << endl;
  VecSimple(Ptv(real)) itg_x(N);
  VecSimple(real) itg_w(N);

  int i = 1, j = 1;
  real dx = 1.0/(real)(n-1);
  real dx2 = dx*dx;
  for (int c=1; c<=N; c++) {
    // set up coordinates
    real xi1 = dx*(i-1);
    real xi2 = dx*(j-1);
    itg_x(c).redim(2);
    itg_x(c)(1) = xi1;
    itg_x(c)(2) = xi2;

    // set up weight
    if ( ((i==1) && (j==1)) || ((i==1) && (j==n)) || ((i==n) && (j==1)) ) {
if ( ((i==1) && (j==n)) || ((i==n) && (j==1)) )
  itg_w(c) = 0.125*dx2;  // sharp corners (1/8 square)
else
  itg_w(c) = 0.25*dx2;   // (1/4 square)
    } else
if ( (i==1)  || (j==1) || (i==(n-j+1)) )
  itg_w(c) = 0.5*dx2; // edges (1/2 square)
else
  itg_w(c) = dx2; // interior (1/1 square)

    // advance to next point
    if (i == (n-j+1)) {
j++;
i = 1;
    } else i++;


  } // end of c=1..N loop.

  // Save integration rule into ElmItgRules FEM::itg_rules.
  itg_rules.ignoreRefill();
  itg_rules.setPointsAndWeights(itg_x, itg_w);

} else {
  // use gaussian quad if n<2.
  s_e << "Using quadrature from menu system. (Phew. I don't like experimental setups.)" << endl;

}

// Output format from menu.
real_format = menu.get("real format");
s_o->setRealFormat(real_format);
```

```
  s_e->setRealFormat(real_format);
  logfile->setRealFormat(real_format);
}


//
// Function for logging both to s_e and logfile.
//
void HydroEigen:: addToLogFile(const String& s) {
  // I could write a joke here; did you get it?
  logfile << s;
  s_e << s;
}



//
// This function erases rows/cols a BANDED matrix according to erase_log[]
// Some assumptions are made on the matrix storage format that I really
// have no reason to believe in. However, I have never experienced a
// counterexample.
//
void HydroEigen::eraseRowAndCols(MatBand(NUMT)& New, MatBand(NUMT)& Old) {
  int rows = Old.getNoRows();
  int bw = Old.getBandwidth();
  bool symm_stor = false;
  bool pivot_allow = false;

  int cols2 = 2*bw - 1;

  int i, j, k;

  int ndiag = cols2 >> 1; // number of sub/super diagonals.
  int middlecol = ndiag + 1;


  for (k=rows; k>=1; k--) {

    // if row/col k is to be deleted, fix matrix data in place!

    if (erase_log[k-1]) {

      // "move up" rows >k by one.
      for (i=k; i<=rows-1; i++) {
  for (j=1; j<=cols2; j++) {
    Old(i,j) = Old(i+1,j);
  }
      }

      // fix upper part of matrix

      for (i=k-1; i>=k-ndiag; i--) {  // totals nd rows
  if (i>=1) {
    for (j=middlecol+(k-1)-i+1; j<=cols2-1; j++)
      Old(i,j) = Old(i,j+1);
    Old(i,cols2)=0;
  }
      }

      // fix lower part of matrix

      for (i=k; i<=k+ndiag-1; i++) { // totals nd rows
  if (i<=rows) {
    for (j=middlecol - 1 + (k-i); j>=2; j--)
      Old(i, j) = Old(i, j-1);
    Old(i, 1) = 0;
  }
      }

      rows--; //we have one less row now.

    }  // end of if
  } // end of k-loop

  // update other matrix parameters --> New.
  New.redim(rows,rows,bw,symm_stor,pivot_allow);

  for (i=1; i<=rows; i++)
    for (j=1; j<=cols2; j++)
      New(i,j)=Old(i,j);

}

//
// Erase row and col number k from sparse matrix _en place_.
//
void HydroEigen::eraseRowAndCol(MatSparse(NUMT)& A, int k) {

  SparseDS& pat = A.pattern();
  int n = pat.getNoRows();
  int m = pat.getNoColumns();

  // NOTE: could not use getNoNonzeroes() because of
  // not correct number of entries. Instead use definition of irow(n+1).

  //int nnz = pat.getNoNonzeroes(); // length of entries array.
  int nnz = pat.irow(n+1)-1;


  //
  // *** first, erase row k. ***
  //

  // must erase entries(s) from s=irow(k) to
  // s=irow(k+1)-1, and also same indices in jcol.
  // then, we must correct the entries in irow,
  // by subtracting delta from irow(i),
```

```
  // i>k and remove irow(k) altogether.


  // delete entries(s) and jcol(s)
  int first = pat.irow(k);
  int last = pat.irow(k+1)-1;
  int delta = last - first + 1; // number of elements to remove from j

  for (int s=first; s<=nnz-delta; s++) {
    A(s) = A(s+delta);
    pat.jcol(s) = pat.jcol(s+delta);
  }

  // fix irow(i)

  // take a copy.
  VecSimple(int) irow_copy(n-1);

  for (int i=1; i<k; i++)
    irow_copy(i) = pat.irow(i);
  for (int i=k; i<=n-1; i++)
    irow_copy(i) = pat.irow(i+1) - delta;

  // redim pat.irow. and fill with modified copy.
  // cols are set to m-1, anticipating removal of column...
  pat.redimIrow(n-1, m-1);

  for (int i=1; i<=n-1; i++)
    pat.irow(i) = irow_copy(i);

  // last entry of irow must be set propery.
  pat.irow(n+1-1) = nnz-delta+1;



  //
  // *** next, erase column k
  //

  // run backwards, s=nnz downto 1.
  // for each s such that jcol(s) = k, jcol(s)
  // and entries(s) must be deleted.
  // furthermore, jcol(s') must be lessened
  // by one for jcol(s')>k. we have erased an
  // element, so irows must be fixed. we subtract
  // 1 from every irow(i)>=s.

  // update variables
  n = pat.getNoRows();
  nnz = pat.irow(n+1)-1;

  // loop backwards ...
  for (int s=nnz; s>=1; s--) {
    if (pat.jcol(s) > k)
      pat.jcol(s)--;
    else if (pat.jcol(s) == k) {
      // remove entry if at column k.
      // (slight special case if s=nnz. nothing
      // needs to be done with entries/jcol then.)
      for (int t=s; t<=nnz-1; t++) {
  pat.jcol(t) = pat.jcol(t+1);
  A(t) = A(t+1);
      }
      // update irows..
      for (int i=1; i<=n+1; i++)
  if (pat.irow(i)>s) pat.irow(i)--;

    }

  }

}


//
// Wrase row/col no. k in a MatDiag(NUMT) object. very simple indeed...
//
void HydroEigen:: eraseRowAndCol(MatDiag(NUMT)& A, int k) {
  Vec(NUMT) temp(A.getNoRows()-1);

  for (int i=1; i<k; i++)
    temp(i) = A(i);
  for (int i=k; i<A.getNoRows(); i++)
    temp(i) = A(i+1);

  A.redim(A.getNoRows()-1);

  for (int i=1; i<=A.getNoRows(); i++)
    A(i) = temp(i);

}


//
// Simple stop-watch functions.
// This should be improved; only seconds are counted.
//
time_t HydroEigen::getElapsedTime() {
  static time_t start = time(NULL);
  return time(NULL)-start;
}
void HydroEigen::reportElapsedTime() {
  addToLogFile(aform("--- elapsed time so far: %d s.\n", getElapsedTime()));
}


//
```

```
// erase_log is an array of bools; one per each unknown in linear system.
// erase_log[k] is set to true if unknown k corresponds with a boundary node.
//
void HydroEigen:: fillEraseLog() {
  int no_nodes = grid->getNoNodes();
  int no_elms = grid->getNoElms();

  erase_log = new bool[no_nodes];
  for (int k=0; k<no_nodes; k++) erase_log[k] = false;

  // fill erase_log with true for boundary nodes.
  for (int e=1; e<=no_elms; e++) {
    int no_dof_per_elm = dof->getNoDofInElm(e);
    for (int e_dof=1; e_dof<=no_dof_per_elm; e_dof++) {
      int n = grid->loc2glob(e, e_dof);
      if (grid->boNode(n)) {
  int k = dof->loc2glob(e, e_dof) - 1;
  erase_log[k] = true;
      }
    }
  }
}

//
// this function enforces the homogenous BCs by removing all rows/cols in the
// matrices K and M that corresponds to boundary nodes. uses fillEraseLog().
//
void HydroEigen:: enforceHomogenousBCs(Handle(Matrix(NUMT))& K, Handle(Matrix(NUMT))& M) {

  addToLogFile("Erasing rows and columns...\n");

  // find the rows/cols that should be removed from the system.
  // loops through the nodes in the grid.
  fillEraseLog();

  addToLogFile(aform("Total number of nonzeroes before == %d\n", K->getNoNonzeroes()));

  //
  // different treatment of MatBand and MatSparse...
  //

  if (mat_type == "MatBand") {
    Handle(MatBand(NUMT)) K_new, M_new;

    K_new.rebind( new MatBand(NUMT)( 1,1 ) );
    M_new.rebind( new MatBand(NUMT)( 1,1 ) );

    eraseRowAndCols(K_new(), (MatBand(NUMT)&)K()) ;
    if (!lump)
      eraseRowAndCols(M_new(), (MatBand(NUMT)&)M()) ;

    addToLogFile("MatBand(NUMT) object K after BC incorporation:\n");
    addToLogFile(aform("   bandwidth == %d\n   dimension == %d\n   nonzeroes  ==%d\n",
           K_new->getBandwidth(), K_new->getNoRows(),
           K_new->getNoNonzeroes()));

    // copy back ...
    if (!lump)
      M.rebind ( M_new() );
    K.rebind ( K_new() );

  } else if (mat_type == "MatSparse") {

    for (int k=K->getNoRows(); k>=1; k--)
      if (erase_log[k-1]) {
  s_e << ".";
  eraseRowAndCol((MatSparse(NUMT)&)(*K), k);
  if (!lump)
    eraseRowAndCol((MatSparse(NUMT)&)(*M), k);
      }

    s_e << endl;


    addToLogFile("MatSparse(NUMT) object K after BC incorporation:\n");
    int nz = ((MatSparse(NUMT)&)K() ).pattern().irow(K->getNoRows()+1)-1;
    int rows = K->getNoRows();
    addToLogFile(aform("   dimension == %d\n   nonzeroes == %d  (%g percent)\n",
           rows,  nz, 100.0*nz/(real)(rows*rows) ));

  }

  // BCs for lumped mass matrix...
  if (lump) {
    addToLogFile("Mass matrix M is lumped. Enforcing BCs...\n");
    for (int k=K->getNoRows(); k>=1; k--)
      if (erase_log[k-1])
  eraseRowAndCol((MatDiag(NUMT)&)(*M), k);

  }

}


//
// This function is the main function of the solver.
// Calculates matrices, solves eigenvalue problem and reports.
//
void HydroEigen:: solveProblem () {
  int n, m;   // matrix dimensions.

  // Init timer.
  getElapsedTime();
  reportElapsedTime();

  // Make the matrix K, i.e., the left matrix.
  addToLogFile("Creating the K matrix ...\n");
```

```
  makeSystem (*dof, *lineq);

  // fetch size.
  lineq->A().size(n, m);

  // Make the mass matrix M
  addToLogFile("Creating the M matrix ... ");

  makeMassMatrix2(M, lineq->A());

  // Fetch a copy of K.
  // For huge systems this may be problematic.
  // I should investigate the possibility of
  // simply making a reference instead.
  lineq->A().makeItSimilar(K);
  *K = lineq->A();

  reportElapsedTime();

  // Tell user what's the deal with the grid.
  addToLogFile("Grid properties:\n");
  addToLogFile(aform("   gridfile == %s\n",  SimCase::getMenuSystem().get("gridfile").c_str()));
  addToLogFile(aform("   number of nodes == %d\n   number of elements == %d\n", grid->getNoNodes(), grid->getNoElms()
      ));

  // Enforce BCs.
  addToLogFile("Enforcing homogenous Dirichlet BCs...\n");
  enforceHomogenousBCs(K, M);
  reportElapsedTime();

  //
  // Here we start the real problem solving part.
  //

  addToLogFile("Instantiating the EigenSolver object ...\n");

  EigenSolver *solver;

  if (!lump) {
    // If M is full then the problem is a generalized problem.
    solver = new EigenSolver (K(), M(), nev, SimCase::getMenuSystem());
  } else {
    // If M is lumped we eliminate the right hand matrix
    // by multiplying with its inverse, which is really
    // simple to to...
    multInvMatDiag(M, K);

//     if (mat_type == "MatBand") {
//       for (int p=1; p<=K->getNoRows(); p++)
//  for (int q=1; q<=K->getNoColumns(); q++)
//    if ( ((MatBand(NUMT)&) K()).insideBand(p,q))
//      K->elm(p,q) /= M->elm(p,p);
//     }
//     else if (mat_type == "MatSparse") {
//       MatSparse(NUMT)& X = (MatSparse(NUMT)&)(*K);
//       SparseDS& pat = X.pattern();
//       for (int p=1; p<=pat.getNoRows(); p++) {
//  int first = pat.irow(p);
//  int last = pat.irow(p+1)-1;
//  for (int q=first; q<=last; q++)
//    X(q) /= M->elm(p,p);
//       }
//     }
    solver = new EigenSolver (K(), nev, SimCase::getMenuSystem());
  }



  // Save matrices if desired.
  // No need to save M if lumped; inv(M) is already mutiplied into K.
  if (savemat && (mat_type=="MatSparse")) {
    addToLogFile("Saving matrices in Matlab format...\n");
    saveMatrix( (MatSparse(NUMT&))(*K), "K", aform("%s_K.m",casename.c_str()) );
    if (!lump)
      saveMatrix( (MatSparse(NUMT&))(*M), "M", aform("%s_M.m",casename.c_str()) );
  }


  // Skip diagonalization?
  if (!arpack_solve) {

    addToLogFile("Skipping diagonalization with ARPACK!\n");

  } else {

    // set up computational mode. Note that EigenSolver
    // in fact supports finding, e.g., the _largest_ eigenvalues
    // instead of the lowest.

    // The reason why we use a void* is that
    // due to not-so-good templating used
    // in ARPACK++ we cannot include the class
    // definitions in HydroEigen.h ...
    the_solver = (void *)solver;
    solver->setCompMode(MODE_REGULAR);
    solver->setMatrixKind(MATRIX_COMPLEX);
    solver->setSpectrumPart(SPECTRUM_SMALLEST_REAL);


    // Go!

    addToLogFile("Finding eigenvalues and eigenvectors ...\n");
    solver->solveProblem();
    addToLogFile("Done!\n");
    reportElapsedTime();

    // Report on the results and findings.
```

```
    report();

  }
}


//
// To calculate K matrix...
// evaluates the integrands in the finite element formulation.
//
void HydroEigen:: integrands (ElmMatVec& elmat, const FiniteElement& fe) {

  /*
    The Hamiltonian for this problem is:

    H = -nabla^2 + gamma^2 r^2 / 4 + i gamma (-y, x) . nabla - 2/r

  */


  int i,j,k;
  const int nsd = fe.getNoSpaceDim();
  const int nbf = fe.getNoBasisFunc();
  const real detJxW = fe.detJxW();
  const real gamma2 = gamma*gamma;


  Ptv(real) coords = fe.getGlobalEvalPt();
  real X = coords(1);
  real Y = coords(2);

  static int gauge_int = (gauge == "symmetric" ? 1 : 2);

  real nabla2;
  real r2 = X*X + Y*Y;
  real r2c = sqrt(r2);
  if (r2c <= epsilon) r2c = epsilon;

  //Complex Im(0,1);
  Complex Im(0,1);

  for (i = 1; i <= nbf; i++)
    for (j = 1; j <= nbf; j++)  {
      nabla2 = 0;
      for (k = 1; k <= nsd; k++)
        nabla2 += fe.dN(i,k)*fe.dN(j,k);
      // add contrib. from nabla^2 (stiffness matrix K)
      elmat.A(i,j) += nabla2*detJxW;


      if (gauge_int == 1) {
  // *** symmetric gauge ***
  // add contrib. from harmonic oscillator term
  elmat.A(i,j) += fe.N(i)*fe.N(j)*r2*0.25*gamma2*detJxW;


  // add contrib. from <A,nabla>.
  if (angmom)
    elmat.A(i,j) += -Im*fe.N(i)*gamma*(
                -Y*fe.dN(j,1) + X*fe.dN(j,2)
              )*detJxW;
      } else {
  // *** non-symmetric gauge ***

  elmat.A(i,j) += 2*Im*gamma*fe.N(i)*Y*fe.dN(j,1)*detJxW;
  elmat.A(i,j) += gamma2*fe.N(i)*fe.N(j)*Y*Y*detJxW;

      }


      // add contrib from coulomb term
      if (nucleus)
  elmat.A(i,j) += -fe.N(i)*fe.N(j)*( 2.0/sqrt(r2) )*detJxW;

    }
}


//
// Simple report function. Writes eigenvectors
// as fields over grid. Also writes eigenvalues.
//
void HydroEigen:: report() {
  // Fetch the EigenSolver object pointer and cast it.
  EigenSolver &solver = *((EigenSolver *)the_solver);

  // In evec_field we store a field corresponding to the eigenvector
  // found.
  Handle(FieldFE) evec_field;
  evec_field.rebind( new FieldFE() );

  // In this we sture the nodal values of the eigenvector field.
  Handle(ArrayGen(NUMT)) evec_values;
  evec_values.rebind( new ArrayGen(NUMT)(grid->getNoNodes()) );

  // Get the eigenvectors and eigenvalues.
  Mat(NUMT) &eigenvectors = solver.getEigenvectors();
  Vec(real) eigenvals(solver.getEigenvalues().getNoEntries());

  // Copy eigenvalues. (We will modify this array, so we take a
  // copy instead of a reference.)
  for (int k=1; k<=eigenvals.getNoEntries(); k++)
    eigenvals(k) = solver.getEigenvalues()(k).Re();


  // index -- permutation of indices that sort the eigenvalues.
  // inv_index -- the inverse permutation.
```

```
  VecSimple(int) index(eigenvals.size()), inv_index(eigenvals.size());
  // Make the sorting indices and the inverse.
  eigenvals.makeIndex(index);
  for (int i=1; i<=index.size(); i++)
    inv_index(index(i)) = i; // Fancy huh! Thanks, HPL!

  // Sort according to index.
  eigenvals.sortAccording2index(index);


  addToLogFile("Expectation values of <r>:\n");

  for (int i = 1; i <= store_evecs; i++) {

    //
    // Create the field corresponding to eigenvector.
    //

    int j2 = 1; // index into eigenvector components.
    // loop through each grid point.
    for (int j=1; j<=grid->getNoNodes(); j++) {
      if (!erase_log[j-1]) {
  evec_values()(j) = eigenvectors(inv_index(i), j2);
  j2++;  // go to next component...
      } else {
  //do nothing with j2, but store BC.
  evec_values()(j) = 0;
      }
    }

    // Redim field and put the field values into it...
    evec_field->redim(grid(), evec_values(), aform("%d", i, eigenvals(i)).c_str());
    // ... and off you go!
    database->dump(evec_field());

    // Write the expectation value <r>.
    addToLogFile(   aform(real_format.c_str(), calcExpectation_r(evec_field(), evec_field()).Re())   );
    if (i<store_evecs)
      addToLogFile(", ");
    else
      addToLogFile("\n");

    // Store probability density if needed.
    if (store_prob) {
//       // calculate probability density
//       for (int j=1; j<=grid->getNoNodes(); j++)
//  evec_values()(j) = pow(evec_values()(j).Re(),2) + pow(evec_values()(j).Im(),2) ;

//       // Dump it.


      evec_field->redim(grid(), aform("prob_%d", i, eigenvals(i)).c_str());
      calcProbabilityDensity(*evec_field, *evec_field, false);
      database->dump(evec_field());

    }
  }

  // Get eigenvalues, print them to stdout...

  // a python-evalable file.
  Os ev_file = Os(aform("%s.eigenvalues", casename.c_str()), NEWFILE);

  addToLogFile("Eigenvalues (real parts):\n");
  for (int k=1; k<=eigenvals.getNoEntries(); k++) {
    addToLogFile(aform("%10.10g", eigenvals(k)));
    ev_file << aform("%10.10g", eigenvals(k));
    if (k<eigenvals.getNoEntries()) {
      addToLogFile(", ");
      ev_file << ", ";
    }
  }
  addToLogFile("\n");
  ev_file << endl;

}

//
// This function is not implemented.
//
void HydroEigen:: resultReport() {}



//
// calculate expectation value of r.
//
NUMT HydroEigen:: calcExpectation_r(FieldFE& u, FieldFE& v) {
  IntegrateOverGridFE integrator;
  IntegrateOverGridFE integrator2;
  IntegrandOfExpectation_r integrand;
  InnerProdIntegrandCalc integrand2;
  integrand.setFields(u, v);
  integrand2.setFields(u, v);

  integrator.volumeIntegral(integrand, *grid);
  integrator2.volumeIntegral(integrand2, *grid);

  return integrand.getResult() / integrand2.getResult();

}

//
// calculate inner product of u and v.
//
NUMT HydroEigen:: calcInnerProd(FieldFE& u, FieldFE& v) {
```

```
  IntegrateOverGridFE integrator;
  InnerProdIntegrandCalc integrand;
  integrand.setFields(u, v);

  integrator.volumeIntegral(integrand, *grid);

  return integrand.getResult();

}



//
// Create (approximate) probability density field.
//
void HydroEigen::calcProbabilityDensity(const FieldFE& u, FieldFE& prob, bool redim) {
  real R, I;
  int nno = grid->getNoNodes();

  if (redim)
    prob.redim(*grid, "prob");


  for (int i=1; i<=nno; i++) {
    R = u.values()(i).Re();
    I = u.values()(i).Im();
    prob.values()(i) =  R*R + I*I;
  }
}




//
// save a sparse matrix in a Matlab-readable format.
// Matrix(NUMT)::save is not entirely reliable...
//
void HydroEigen:: saveMatrix(MatSparse(NUMT)& A, const String& Aname, const String& fname) {
  Os f(fname, NEWFILE);
  SparseDS& pat = A.pattern();
  int rows = pat.getNoRows();
  int cols = pat.getNoColumns();
  NUMT dummy;
  bool reals = (sizeof(dummy) == sizeof(real));



  f << "% This is a matrix saved by HydroEigen::saveMatrix()" << endl;
  f << "%    gridfile == " << gridfile << "." << endl;
  f << endl;

  f << Aname << " = sparse(" << rows << ", " << cols << ");" << endl << endl;

  for (int i=1; i<=rows; i++) {
    for (int s=pat.irow(i); s<=pat.irow(i+1)-1; s++) {
      f << Aname << "(" << i << ", " << pat.jcol(s) << ") = ";
      if (reals)
  f << A(s) << ";" << endl;
      else
  f << "complex" << A(s) << ";" << endl;  // complex(re, im);
    }
  }

  f->close();
}


//
// makeMassMatrix2. First argument is ref. to
// matrix that shall hold mass matrix, second
// argument has the correct pattern.
//
void HydroEigen::makeMassMatrix2(Handle(Matrix(NUMT))& Dest, const Handle(Matrix(NUMT)) &WithPattern) {

  // Make mass matrix according to menu choices et.c.
  // Lumped matrices are always MatDiag(NUMT) objects.
  // Not lumped are same format as right hand side matrix.

  int n, m; // matrix size.

  WithPattern->size(n, m);

  if (!lump) {
    // Make a full mass matrix.
    addToLogFile("(not lumped)\n");

    if (mat_type == "MatSparse") {
      // must COPY the sparsity pattern. makeItSimilar only
      // copies a _reference_ to the SparseDS object. Hence,
      // modifying one matrix will destroy the structure of the other.
      MatSparse(NUMT)& orig = (MatSparse(NUMT)&)(WithPattern());
      int nnz = orig.getNoNonzeroes();

      Dest.rebind( new MatSparse(NUMT)(n,m,nnz) );
      SparseDS& pattern = ((MatSparse(NUMT)&)Dest()).pattern();
      for (int i=1; i<=m+1; i++)
  pattern.irow(i) = orig.pattern().irow(i);
      for (int i=1; i<=nnz; i++)
  pattern.jcol(i) = orig.pattern().jcol(i);

    } else {
      // The structure of MatBand is simpler to copy...
      WithPattern->makeItSimilar(Dest);
    }

  } else {
```

```
    // Create a lumped mass matrix, i.e., a MatDiad.
    addToLogFile("(lumped)\n");
    M.rebind (new MatDiag(NUMT)(n) );
  }
  makeMassMatrix (*grid, *Dest, lump);


}


//
// Multiply A with inverse of D, which is diagonal.
//
void HydroEigen::multInvMatDiag(Handle(Matrix(NUMT)) D, Handle(Matrix(NUMT)) A) {

    if (mat_type == "MatBand") {
      for (int p=1; p<=A->getNoRows(); p++)
  for (int q=1; q<=A->getNoColumns(); q++)
    if ( ((MatBand(NUMT)&) A()).insideBand(p,q))
      A->elm(p,q) /= D->elm(p,p);
    }
    else if (mat_type == "MatSparse") {
      MatSparse(NUMT)& X = (MatSparse(NUMT)&)(*A);
      SparseDS& pat = X.pattern();
      for (int p=1; p<=pat.getNoRows(); p++) {
  int first = pat.irow(p);
  int last = pat.irow(p+1)-1;
  for (int q=first; q<=last; q++)
    X(q) /= D->elm(p,p);
      }
    }

}

//
// End of class definition.
//
```

B.3 The `TimeSolver` class

This is the class definition of `TimeSolver` that solves the time dependent Schrödinger equation. The main program is `main.cpp` and this instatiates the `TimeSolver` class. Notice that the full functionality from the base class `HydroEigen` is retained.

B.3.1 `TimeSolver.h`

```
#ifndef TimeSolver_h
#define TimeSolver_h

#include "HydroEigen.h"



//
// simple integrand for <x>. inlined.
//
class IntegrandOfExpectation_x : public InnerProdIntegrandCalc {
public:
  IntegrandOfExpectation_x() { }
  virtual void integrandsG(const FiniteElement& fe) {
    real detJxW = fe.detJxW();
    NUMT uval = u->valueFEM(fe);
    NUMT vval = v->valueFEM(fe);
    Ptv(real) x  = fe.getGlobalEvalPt();

    result += conjugate(uval)*vval*x(1)*detJxW;  // integrate and store in result.
  }

};

//
// simple integrand for <y>. inlined.
//
class IntegrandOfExpectation_y : public InnerProdIntegrandCalc {
public:
  IntegrandOfExpectation_y() { }
  virtual void integrandsG(const FiniteElement& fe) {
    real detJxW = fe.detJxW();
    NUMT uval = u->valueFEM(fe);
    NUMT vval = v->valueFEM(fe);
    Ptv(real) x  = fe.getGlobalEvalPt();

    result += conjugate(uval)*vval*x(2)*detJxW;  // integrate and store in result.
  }

};



// just some handy defs
#define THETA_RULE 1
#define LEAP_FROG 2
#define IC_GAUSSIAN 1
#define IC_FIELD 2


// *******************************************************
//
```

```
// Class declaration of TimeSolver --- solving the time
// dependent öSchrdinger equation. This is derived from
// HydroEigen because the two problems share many important
// properties, such as the Hamiltonian.
//
// ********************************************************
class TimeSolver : public HydroEigen {
 private:

  Handle(Matrix(NUMT)) A;   // usually holds matrix from MakeSystem. M is defined in HydroEigen.

  // Time parameters.
  int time_method;          // indicates what method to use, THETA_RULE or LEAP_FROG
  Handle(TimePrm) tip;      // numerical clock...
  real dt;                  // time step
  real t_final;             // final time

  real omega;               // parameters for
  real delta;               // varying magnetic field
  real gamma0;              // gamma = gamma0 * pow(sin(t/(pi*T)), 2) * cos(omega*t+delta)

  real theta;               // for theta rule, \in [0,1]., 0.5 == Crank-Nicholson
  bool do_time_simulation;  // indicates whether og not a time dep. simulation should be made.

  // Report variables.
  real dt_rep;              // time interval between reports.
  int n_rep;                // number of reports during simulation.

  // Initial condition parameters.
  int ic_type;              // IC_GAUSSIAN or IC_FIELD
  String field_database;    // name of the database that holds fields to load.
  int field_no;             // number of field to use as IC.
  String gaussian_parameters;  // parameters for gaussFunc IC.
  Handle(FieldFE) *fields;  // array of field handles. holds fields loaded from database.
  int no_of_fields;         // number of fields to load, i.e., length of fields array.

  // fields et.c.; recall that the grid is defined in HydroEigen.

  //Handle(FieldFE) u;      // u is defined in HydroEigen!
  Handle(FieldFE) u_prev;   // previous solution.
  Handle(Vec(NUMT)) scratch;
  Handle(Vec(NUMT)) scratch2; // scratch vectors.

  // for the integrands() function.
  // defining these variables here saves some time
  // when assembling the linear system at each time step.
  MassMatIntg *mass_integrand; // Handle(...) not impl. ???
  ElmMatVec elmat2;
  ElmMatVec elmat3;

 public:
  // constructors and destructors.
  TimeSolver() { mass_integrand = NULL; };
  ~TimeSolver() { if (mass_integrand) delete mass_integrand; };

  //
  // "compulsory" methods.
  //
  virtual void define (MenuSystem& menu, int level = MAIN);
  virtual void scan ();
  virtual void solveProblem ();
  //virtual void report ();
  virtual void resultReport () { };
  virtual void fillEssBC();  // fill essential BCs.
  virtual void setIC();    // set initial condition.
  virtual void integrands(ElmMatVec& elmat, const FiniteElement& fe); // overloaded integrands.

  //
  // Provide magnetic field as function of time.
  //
  virtual real gammaFunc(real t);

  // Calculate expectation value when given a matrix.
  NUMT calcExpectationFromMatrix(FieldFE& u, FieldFE& v, Matrix(NUMT)& A);

  //
  // Report methods. Used during simulation.
  //
  virtual void initialReport();
  virtual void finalReport();
  virtual void reportAtThisTimeStep(int t_index, int r_index);

  Ptv(real) calcExpectation_pos(FieldFE& u, FieldFE& v);

 protected:
  //
  // Load fields (and grid!) from database
  //
  void loadFields(String& db_name);
};



#endif
```

B.3.2 TimeSolver.cpp

```
#include "TimeSolver.h"
#include <IntegrateOverGridFE.h>


//********************************************************
//
```

```
// Class definition of TimeSolver.
//
//******************************************************


//
// Main problem solving method.
//
void TimeSolver::solveProblem() {

  if (!do_time_simulation) {
    s_e << "Skipping time dependent simulation. Diagonalizing instead!" << endl;
    HydroEigen::solveProblem();
    return;
  }

  //Handle(Matrix(NUMT)) A, M;  // holds the linear system and mass matrix.
  A.rebind(NULL);
  M.rebind(NULL);


  //bool matrix_has_changed = true;
  bool first_step_lf = (time_method == LEAP_FROG);

  //real last_report_time = 0;  // used for report book-keeping.
  int last_report_index = 0;  // used for report book-keeping.
  int report_index = 0;
  int t_index = 0;   // holds number of steps taken.

  tip->initTimeLoop();   // initialize timeloop.

  // make M and A.
  gammaFunc(tip->time()); // update magnetic field.
  s_e << "Make M and A ..." << endl;

  do_time_simulation = false;
  makeSystem(*dof, *lineq); // make H^0.
  do_time_simulation = true;

  A.rebind(lineq->A());  // fetch pointer to the element matrix.
  s_e << "Make mass matrix..." << endl;
  A->makeItSimilar(M);  // copy structure of A into M.
  makeMassMatrix(*grid, M(), lump);

  setIC(); // set initial condition.
  linsol = u->values();  // start vector for first time step.

  initialReport();  // report various stuff
  reportAtThisTimeStep(0, 0);  // report initial condition.

  while (!tip->finished()) { // loop until t = t_final
    tip->increaseTime(); // t -> t + dt
    addToLogFile("% ");
    addToLogFile(aform("Solving for t == %g\n", tip->time()));

    // build Hamiltonian and mass matrix at this time step.
    s_e << "Essential BCs..." << endl;

    fillEssBC(); // ess. BCs.

    // first step i leap-frog should be a theta-rule step.
    // create mass matrix also.
    if (first_step_lf) {
      time_method = THETA_RULE;
    }

    //
    //  --- solve for leap frog method. ---
    //
    if (time_method == LEAP_FROG) {
      s_e << "(Using LEAP_FROG)" << endl;
      s_e << "(";
      s_e << "making system ... " ;
      // make coefficient matrix M and right hand side -2*i*dt*H*u
      makeSystem(*dof, *lineq);

      // solving a lumped system in the leap-frog method
      // is very easy! :-)
      // therefore we separate the two cases.
      if (!lump) {
        s_e << "solving linear system ... ";
        lineq->solve(true);
      } else {
        s_e << "solving linear system (by inspection) ... ";
        Vec(NUMT)& rhs = (Vec(NUMT)&)lineq->b(); // a little bit ugly but...
        for (int p=1; p<=linsol.size(); p++)
          linsol(p) = pow(lineq->A().elm(p,p), -1) * rhs(p);
      }

      // add u_prev to solution of linear system.
      // linsol now then the new u's field values.
      linsol.add(linsol, u_prev->values());

      *u_prev = *u; // update u_prev for next time step.
      dof->vec2field(linsol, *u); // store new solution in field object.

      s_e << ")" << endl;
    }

    //
    // --- solve for theta rule method. ---
    //
    if (time_method == THETA_RULE) {
      s_e << "(Using THETA_RULE)" << endl;
      s_e << "(";

      // make the system: rhs = [M - i*dt*(1-theta)*H(t-dt)]*u,
```

```
      // A = M + i*theta*dt*H(t)
      s_e << "making system ... " ;
      makeSystem(*dof, *lineq);

      s_e << "solving linear system ... ";
      lineq->solve(true);   // linsol holds solution

      *u_prev = *u; // update u_prev for next time step
      dof->vec2field(linsol, *u); // convert to field object.

      s_e << ")" << endl;
    }


    // if we did a theta-rule step in the first leap-frog step...
    if (first_step_lf)
      time_method = LEAP_FROG;

    first_step_lf = false;


    // solved a new time step we did!
    t_index++;

    report_index = (int)floor(tip->time()/dt_rep);

    // report
    if (tip->finished() ||  (report_index > last_report_index)) {
      // update last_report_time
      last_report_index = report_index;

      reportAtThisTimeStep(t_index, report_index);
    }

  }  // end of time loop

  finalReport();  // report various stuff.

}


//
// Report in time-loop
//
void TimeSolver::reportAtThisTimeStep(int t_index, int r_index) {
  Handle(FieldFE) prob;
  prob.rebind( new FieldFE );

  String prefix = aform("%s_data.", casename.c_str());

  s_e << "Reporting ... ";

  // Store current solution of desired.
  if (store_evecs>0) {
    database->dump(*u, &(tip()));
    s_e << "(u saved)";
  }
  // Store probability density if desired.
  if (store_prob) {
    // Calc. prob. density from current solution.
    calcProbabilityDensity(*u, *prob, true);
    database->dump(*prob, &(tip()));
    s_e << "(u*u saved)";
  }

  Ptv(real) pos = calcExpectation_pos(*u,*u);

  addToLogFile("\n% ");
  addToLogFile(aform("reporting at t(%d) == %10.10g;\n", t_index, tip->time()));
  addToLogFile(prefix);
  addToLogFile(aform("t(%d)           = %10.10g;\n", r_index+1, tip->time()));
  addToLogFile(prefix);
  addToLogFile(aform("u_norm(%d)      = %10.10g;\n", r_index+1, calcInnerProd(*u,*u).Re()));
//  addToLogFile(aform("u_energy(%d)    = %10.10g;\n", r_index+1, calcExpectationFromMatrix(*u, *u, *A).Re()));
  addToLogFile(prefix);
  addToLogFile(aform("gamma(%d)       = %10.10g;\n", r_index+1, gammaFunc(tip->time())));
  addToLogFile(prefix);
  addToLogFile(aform("u_pos(%d,:)     = [%10.10g, %10.10g];\n",  r_index+1, pos(1), pos(2)));

  if ((t_index > 0) && !(lump && (time_method == LEAP_FROG))  ) {
    // retrieve performance info.
    LinEqStatBlk &perf = lineq->getPerformance();
    SolverStatistics &stats = perf.solver_info;
    addToLogFile(prefix);
    addToLogFile(aform("solve_time(%d)  = %10.10g;\n", r_index+1, stats.cputime));
    addToLogFile(prefix);
    addToLogFile(aform("solve_niter(%d) = %d;\n", r_index+1, stats.niterations));
  }
  addToLogFile(prefix);
  addToLogFile(aform("system_time(%d) = %10.10g;\n", r_index+1, cpu_time_makeSystem));

  addToLogFile(prefix);

  do_time_simulation = false;
  s_e << "Making Hamiltonian..." << endl;
  gammaFunc(tip->time());
  makeSystem(*dof, *lineq); // make H^\ell.
  do_time_simulation = true;
  addToLogFile(aform("u_energy(%d)    = %10.10g;\n", r_index+1, calcExpectationFromMatrix(*u, *u, lineq->A()).Re()));


}



//
// Extend functionality of solver menu.
```

```
//
void TimeSolver::define(MenuSystem& menu, int level) {

  menu.addItem(level, "simulation in time", "set to false to solve eigenvalue problem instead", "true");

  int level1 = level+1;
  MenuSystem::makeSubMenuHeader (menu, "Params for time dependent solve", "time", level1, 'T');
  menu.addItem(level1, "time method", "time integration method", "theta-rule");  // or "leap-frog"
  menu.addItem(level1, "T", "final time", "1.0");
  menu.addItem(level1, "dt", "time step", "0.1");
  menu.addItem(level1, "omega", "omega", "0");
  menu.addItem(level1, "delta", "delta", "0");
  menu.addItem(level1, "gamma0", "gamma0", "0");

  menu.addItem(level1, "n_rep", "number of reports", "10");
  menu.addItem(level1, "theta", "theta", "0.5");

  menu.addItem(level1, "ic type", "initial condition type", "gaussian");  // or "field"
  menu.addItem(level1, "field database", "field database", "none");  // database that should hold eigenvectors.
  menu.addItem(level1, "ic field no", "number of field to use as ic", "1");   // 1 should be ground state...

  menu.addItem(level1, "gaussian", "gaussian parameters", "-x -10 -y 0 -sx 2 -sy 2 -kx 0 -ky 1"); // used to
        initialize gaussFunc.

  // Set up previous functionality as well.
  HydroEigen::define(menu, level);

}


//
// Extend functionality of scan().
//
void TimeSolver::scan() {
  MenuSystem &menu = SimCase::getMenuSystem();

  int ok = false;

  do_time_simulation = menu.get("simulation in time").getBool();

  //
  // If user sets 'simulation in time' to false, the HydroEigen solver is used instead.
  //
  if (!do_time_simulation) {
    // we do not want to do a time dependent simulation but rather a time independent simulation.

    s_e << "Calling HydroEigen::scan()." << endl;
    // call the old scan().
    HydroEigen::scan();
    s_e << "Back from HydroEigen::scan()." << endl;

    s_e << "Skipping initialization of time dependent solver." << endl;

    return;
  }


  // Get the time method from menu.
  // Abort if not "theta-rule" or "leap-frog"
  String s = menu.get("time method");
  ok = false;
  if (s == "theta-rule") {
    time_method = THETA_RULE;
    ok = true;
  } else if (s == "leap-frog") {
    time_method = LEAP_FROG;
    ok = true;
  }
  if (!ok)
    fatalerrorFP("TimeSolver::scan()", "Illegal time method!");

  // Get time parameters, set up tip.
  t_final = menu.get("T").getReal();
  dt = menu.get("dt").getReal();
  theta = menu.get("theta").getReal();
  gamma0 = menu.get("gamma0").getReal();
  omega = menu.get("omega").getReal();
  delta = menu.get("delta").getReal();

  tip.rebind( new TimePrm() );
  tip->scan(aform("dt=%g, t in [%g,%g]", dt, 0.0, t_final));

  // Set up report parameters.
  n_rep = (int)menu.get("n_rep").getReal();
  if (n_rep < 1)
    n_rep = 1; // at least 1 report (at the end of sim)
  dt_rep = t_final/(real)n_rep;
  if (dt_rep < dt) {
    dt_rep = dt; // not too many reports...
    n_rep = (int)(t_final/dt);
  }


  // Set up IC parameters.
  field_database = menu.get("field database");
  field_no = (int)menu.get("ic field no").getReal();
  s = menu.get("ic type");
  gaussian_parameters = menu.get("gaussian");
  ok = false;
  if (s == "gaussian") { ic_type = IC_GAUSSIAN; ok = true; }
  if (s == "field") { ic_type = IC_FIELD; ok = true; }
  if (!ok)
    fatalerrorFP("TimeSolver::scan()", "Illegal IC type!");



  // Call the old scan(). Set up grid et.c.
```

```
  s_e << "Calling HydroEigen::scan()." << endl;
  HydroEigen::scan();
  s_e << "Back from HydroEigen::scan()." << endl;


  // Load fields from database.
  // This overwrites the grid read by HydroEigen::scan() if fields are found/read.
  if (ic_type == IC_FIELD)
    loadFields(field_database);


  // set up u and u_prev.
  u_prev.rebind(new FieldFE);
  u_prev->redim(*grid, "u_prev");
  u.rebind(new FieldFE);
  u->redim(*grid, "u");

  // set up scratch vectors
  scratch.rebind(new Vec(NUMT));
  scratch->redim(grid->getNoNodes());
  scratch->fill(0.0);
  scratch2.rebind(new Vec(NUMT));
  scratch2->redim(grid->getNoNodes());
  scratch2->fill(0.0);

  // for integrands().
  mass_integrand = new MassMatIntg(lump); // deleted in desctructor...
  elmat2.attach(*dof);
  elmat3.attach(*dof);

}

//
// Gamma as function of t.
//
real TimeSolver::gammaFunc(real t) {
  //gamma = 1;
  if (gamma0==0)
    gamma=0;
  else
    gamma = gamma0*pow(sin(M_PI*t/t_final), 2)*cos( (omega*(t-0.5*t_final)+delta)*2*M_PI  );

  return gamma;
}



//
// Set homogenous BCs.
//
void TimeSolver::fillEssBC() {
  dof->initEssBC();
  const int nno = grid->getNoNodes();

  for (int i=1; i<=nno; i++)
    if (grid->boNode(i))
      dof->fillEssBC(i, 0.0);

}


//
// Simple gaussian functor.
//
class GaussFunc : public FieldFunc {
public:
  Ptv(real) r0;     // centre
  Ptv(real) sigma;  // width
  Ptv(real) x;      // work vector
  Ptv(real) k0;     // momentum

  GaussFunc() { // default constructor: exp(-r^2 / 2)
    r0.redim(2); sigma.redim(2); x.redim(2); k0.redim(2);
    r0.fill(0.0);
    sigma.fill(0.5);
    x.fill(0.0);
    k0.fill(0.0);
  }
  virtual void init(real x0, real y0, real s1, real s2, real k1, real k2) {  // exp(-(x-x0)^2/sigma1^2 - (y-y0)^2/
       sigma2^2)
    r0.redim(2); sigma.redim(2); x.redim(2); k0.redim(2);
    k0(1) = k1; k0(2) = k2;
    r0(1) = x0; r0(2) = y0;
    sigma(1) = s1; sigma(2) = s2;
    sigma(1) = 0.5/(sigma(1)*sigma(1));
    sigma(2) = 0.5/(sigma(2)*sigma(2));
    x.fill(0.0);
  };
  virtual NUMT valuePt(const Ptv(real)& r, real t=DUMMY) { // evaluate gaussian
    NUMT temp;
    x(1) = r(1)-r0(1);
    x(2) = r(2)-r0(2);
    temp = exp( Complex(0, 1)*x.inner(k0) );
    x(1)=x(1)*x(1);
    x(2)=x(2)*x(2);
    return exp( -x.inner(sigma) ) *  temp ;
  };
  virtual void scan(String& s);
};

void GaussFunc::scan(String& s) {
  real x = 0, y = 0, sx = 1, sy = 1, kx = 0, ky = 0;
  int ntok = s.getNoTokens(" ");
  for (int i=1; i<=ntok; i++) {
    String token, token2;
    s.getToken(token, i, " ");
```

```
    if (token == "-x") { s.getToken(token2, i+1, " "); x = token2.getReal(); }
    if (token == "-y") { s.getToken(token2, i+1, " "); y = token2.getReal(); }
    if (token == "-sx") { s.getToken(token2, i+1, " "); sx = token2.getReal(); }
    if (token == "-sy") { s.getToken(token2, i+1, " "); sy = token2.getReal(); }
    if (token == "-kx") { s.getToken(token2, i+1, " "); kx = token2.getReal(); }
    if (token == "-ky") { s.getToken(token2, i+1, " "); ky = token2.getReal(); }

  }
  init(x, y, sx, sy, kx, ky);
}


//
// Set initial conditions.
//
void TimeSolver::setIC() {

  // If ic_type == IC_GAUSSIAN, set up a gaussian initial
  // condition. If ic_type == IC_FIELD, we choose a loaded
  // eigenvector as IC.

  if (ic_type == IC_GAUSSIAN) {
    GaussFunc gaussian;
    gaussian.scan(gaussian_parameters);
    u_prev->fill(gaussian, 0.0);
  } else {
    if ((field_no>no_of_fields) || (field_no<1))
      field_no = 1;

    // copy field from database.
    *u_prev = *(fields[field_no - 1]);

  }

  // ensure unit norm.
  u_prev->mult( 1.0/sqrt(calcInnerProd(*u_prev, *u_prev).Re()) );
  // set u = u_prev.
  *u = *u_prev;

}


//
// calculate (u, Av). uses scratch assumed to be of same length as u and v.
//
NUMT TimeSolver:: calcExpectationFromMatrix(FieldFE& u, FieldFE& v, Matrix(NUMT)& A) {

  Vec(NUMT)& u_values = u.valuesVec();
  Vec(NUMT)& v_values = v.valuesVec();
  A.prod(v_values, *scratch);
  return u_values.inner(*scratch);

}


//
// function that loads the fields stored in database db_name.
// they are typically created with HyddroEigen solvers over the
// same grid and Hamiltonian.
//
void TimeSolver::loadFields(String& db_name) {
  real t = 0.0;
  int nsd = 0;
  String field_name;
  String field_type;
  int component=0, max_component=0;
  int test = 0;

  if (db_name == "none") {
    no_of_fields = 0;
    return;
  }

  // open the simres file, e.g., SIMULATION.
  SimResFile s(db_name);

  s_e << "Dataset name == " << s.getDatasetName() << endl;

  int n = s.getNoFields(); // get number of fields stored.
  bool load_it[n]; // indicates what fields should be loaded.

  s_e << "I check " << n << " fields from " << db_name << "." << endl;

  no_of_fields = 0;
  for (int i=0; i<n; i++) {
    s.locateField(i+1, field_name, t, nsd, field_type, component, max_component);
    // eigenvectors are stored with their number as field name, the number >0.
    test = atoi(field_name.c_str());
    if (test>0) {
      load_it[i] = true;
      no_of_fields++;  // increase total number of fields.
    }
    else {
      load_it[i] = false;
    }
  }

  s_e << "I found " << no_of_fields << " appropriate fields to load ";

  fields = new Handle(FieldFE)[no_of_fields];  // allocate handles.
  int j = 0;
  for (int i=0; i<n; i++) {
    if (load_it[i]) {

      fields[j].rebind( new FieldFE );  // allocate a new field.
      SimResFile::readField(fields[j](), s, aform("%d", j+1), t);  // read it.
```

```
      // ensure unit norm.
      fields[j]->mult( 1.0/sqrt(calcInnerProd(*fields[j], *fields[j]).Re()) );
      j++;
      s_e << ".";
    }
  }
  s_e << " ok "<< endl;

  s_e << "Fetching the grid." << endl;

  grid.rebind( fields[0]->grid() );

  // because we have a new grid, we must update some stuff.

  // init the unknown u and the dof. actually not used in the simulator.
  // needed in the DegFreeFE object.
  //u.rebind (new FieldFE (*grid,"u"));
  dof.rebind (new DegFreeFE (*grid, 1));  // 1 for 1 unknown per node

  // init the linear system. (holds matrix in eigenvalue formulation)
  // FEM systems are not solved in HydroEigen, but the structure is needed.
  lineq.rebind(new LinEqAdmFE());
  lineq->scan(SimCase::getMenuSystem());

  linsol.redim(grid->getNoNodes());
  linsol.fill(0.0);
  lineq->attach(linsol);


}



//
// Do an initial report.
//
void TimeSolver::initialReport() {
  String prefix = aform("%s_data.", casename.c_str());

  addToLogFile("%\n");
  addToLogFile(aform("%s simulation log from case %s\n", "%", casename.c_str()));

  addToLogFile("\n% erase variables we use.\n");
  addToLogFile(aform("clear %s_data;\n",casename.c_str()));

  addToLogFile(aform("%scasename = '%s';\n", prefix.c_str(), casename.c_str()));
  addToLogFile("%\n\n");
  MenuSystem& menu = SimCase::getMenuSystem();

  addToLogFile("% simulation parameters: \n");
  addToLogFile(prefix);
  addToLogFile(aform("gridfile = '%s';\n", menu.get("gridfile").c_str()));
  if (time_method == THETA_RULE) {
    addToLogFile(aform("\n%s using the theta-rule.\n", "%"));
    addToLogFile(prefix);
    addToLogFile(aform("theta = %10.10g;\n", theta));
  }   else {
    addToLogFile("% using the leap-frog method.\n");
  }

  addToLogFile("% is mass matrix lumped?\n");
  addToLogFile(prefix);
  addToLogFile("lump = ");
  if (lump)
    addToLogFile("1;\n");
  else
    addToLogFile("0;\n");

  addToLogFile("\n");
  addToLogFile(prefix);
  addToLogFile(aform("dt = %10.10g;\n", dt));
  addToLogFile(prefix);
  addToLogFile(aform("t_final = %10.10g;\n", t_final));
  addToLogFile(prefix);
  addToLogFile(aform("gamma0 = %10.10g;\n", gamma0));
  addToLogFile(prefix);
  addToLogFile(aform("omega = %10.10g;\n", omega));
  addToLogFile(prefix);
  addToLogFile(aform("delta = %10.10g;\n", delta));

  if (ic_type == IC_GAUSSIAN) {
    addToLogFile("\n% using a Gaussian for IC.\n");
    addToLogFile(prefix);
    addToLogFile("gaussian_params = '");
    addToLogFile(menu.get("gaussian"));
    addToLogFile("';\n");
  } else {
    addToLogFile("% using a field from database for IC.\nfield_database = '");
    addToLogFile(prefix);
    addToLogFile(menu.get("field database"));
    addToLogFile("';\n");
    addToLogFile(prefix);
    addToLogFile("field_number = ");
    addToLogFile(menu.get("field no"));
    addToLogFile(";\n");
  }




}

//
// Do a final report.
//
void TimeSolver::finalReport() {
```

```
  String prefix = aform("%s_data.", casename.c_str());

  addToLogFile("\n% finally ...\n");
  addToLogFile(prefix);
  addToLogFile(aform("n_rep = length(%st);\n",prefix.c_str()));
  addToLogFile(prefix);
  addToLogFile(aform("solve_total = sum(%ssolve_time);\n",prefix.c_str()));
  addToLogFile(prefix);
  addToLogFile(aform("system_total = sum(%ssystem_time);\n",prefix.c_str()));



}


//
// new integrands for time dependent problems.
// reuses  HydroEigen::integrands() as well.
//
void TimeSolver::integrands(ElmMatVec& elmat, const FiniteElement& fe) {

  // if we solve a eigenvalue problem, use HydroEigen::integrands()  instead.
  if (!do_time_simulation) {
    HydroEigen::integrands(elmat, fe);
    return;
  }


  int nbf = fe.getNoBasisFunc();

  // initialize scratch objects.
  MassMatIntg mass_integrand(lump);
  elmat2.attach(*dof);
  elmat3.attach(*dof);

  elmat2.refill(elmat.elm_no);
  elmat3.refill(elmat.elm_no);


  if (time_method == LEAP_FROG) {
    // goal: A = M,  b = -2*i*dt*H(t)*u(t)

    // create the leap-frog element matrix and vector.

    // element matrix = mass matrix.
    mass_integrand.integrands(elmat, fe);  // add contribution from mass matrix.

    // element vector = -2*i*dt*H(t)*u
    // create a new ElmMatVec to hold the Hamiltonian contributions.

    gammaFunc(tip->time()-dt); // update gamma.
    HydroEigen::integrands(elmat2, fe);

    // fill elmat2.b with u's values.
    for (int i=1; i<=nbf; i++)
      elmat2.b(i) = u->values()(elmat.loc2glob_u(i));

    elmat2.A.prod(elmat2.b, elmat3.b);  // elmat2.b=elmat2.A*elmat2.b
    elmat3.b.mult(Complex(0, -2*dt));  // multiply Hu by -2i*dt
    elmat.b.add(elmat.b, elmat3.b);  // update elmat.b


  } else
  if (time_method == THETA_RULE) {
    // create the theta-rule element matrix and vector

    // b = [M - i*dt*(1-theta)*H(t-dt)]*u

    gammaFunc(tip->time() - dt);

    mass_integrand.integrands(elmat2, fe);  // add contribution from mass matrix.

    // add contrib to M*u in rhs.
    for (int i=1; i<=nbf; i++)
      elmat2.b(i) = u->values()(elmat.loc2glob_u(i));
    elmat2.A.prod(elmat2.b, elmat3.b);
    elmat.b.add(elmat.b, elmat3.b);  // elmat.b += M*u

    elmat2.A.fill(0.0);
    elmat2.b.fill(0.0);
    elmat3.b.fill(0.0);
    HydroEigen::integrands(elmat2, fe); // Hamiltonian.
    elmat2.A.mult(Complex(0, -dt*(1.0-theta)));  // ... times -i*dt*(1-theta)
    for (int i=1; i<=nbf; i++)
      elmat2.b(i) = u->values()(elmat.loc2glob_u(i));
    elmat2.A.prod(elmat2.b, elmat3.b);

    elmat.b.add(elmat.b, elmat3.b);  // elmat.b += -i*dt*(1-theta)*H*u

    // A = M + i*dt*theta*H(t)

    gammaFunc(tip->time()); // update gamma.
    elmat2.A.fill(0.0);  // erase values.
    HydroEigen::integrands(elmat2, fe);  // fill it
    elmat2.A.mult(Complex(0, dt*theta));
    elmat.A.add(elmat.A, elmat2.A);  // elmat.A = - i*dt*theta*H(t);
    mass_integrand.integrands(elmat, fe);  // add M.



  }
```

```
}


//
// calculate expectation value of position.
//
Ptv(real) TimeSolver:: calcExpectation_pos(FieldFE& u, FieldFE& v) {
  IntegrateOverGridFE integrator;
  IntegrateOverGridFE integrator2;
  IntegrandOfExpectation_x integrand_x;
  IntegrandOfExpectation_y integrand_y;
  InnerProdIntegrandCalc integrand2;
  integrand_x.setFields(u, v);
  integrand_y.setFields(u, v);
  integrand2.setFields(u, v);
  Ptv(real) result; result.redim(2);

  integrator.volumeIntegral(integrand_x , *grid);
  integrator.volumeIntegral(integrand_y , *grid);
  integrator2.volumeIntegral(integrand2 , *grid);

  result(1) = (integrand_x.getResult() / integrand2.getResult()).Re();
  result(2) = (integrand_y.getResult() / integrand2.getResult()).Re();

  return result;

}
```

B.3.3 main.cpp

```
#include <TimeSolver.h>

int main (int argc, const char* argv[])
{
  initDiffpack (argc, argv);
  global_menu.init ("Time solver test", "TimeSolver");
  TimeSolver sim;
  global_menu.multipleLoop (sim);
  return 0;
}
```

B.4 The EigenSolver class

This class extends the class definition found in Ref. [5] to include standard and generalized complex eigenvalue problems.

B.4.1 EigenSolver.h

```
#ifndef _EIGENSOLVER_H_

#define _EIGENSOLVER_H_

#include <FEM.h>
#include <Matrix_Complex.h>
#include <LinEqSolver.h>
#include <LinEqAdm.h>



// display warning messages
#define WARNING(s) { s_e << " >>> EigenSolver warning: " << s << endl; }


// constants used in the class definition
#define PROBLEM_UNDEFINED 0
#define PROBLEM_STANDARD 1
#define PROBLEM_GENERALIZED 2
#define MATRIX_SYMMETRIC 1
#define MATRIX_NONSYMMETRIC 2
#define MATRIX_COMPLEX 3
#define MODE_REGULAR 1
#define MODE_SHIFT_INVERT 2
#define MODE_BUCKLING 3
#define MODE_CAYLEY 4
#define MODE_COMPLEX_SHIFT 5
#define SPECTRUM_SMALLEST_MAGNITUDE 0
#define SPECTRUM_SMALLEST_ALGEBRAIC 1
#define SPECTRUM_SMALLEST_REAL 2
#define SPECTRUM_SMALLEST_IMAG 3
#define SPECTRUM_LARGEST_MAGNITUDE 4
#define SPECTRUM_LARGEST_ALGEBRAIC 5
#define SPECTRUM_LARGEST_REAL 6
#define SPECTRUM_LARGEST_IMAG 7

//
// class definition of EigenSolver.
//
class EigenSolver {
 private:
  int n, nev;                    // dimension of problem, number of eigenvalues to seek
  Handle(Matrix(NUMT)) A, B;     // problem defining matrices
  int problem_kind;              // standard or generalized
  int matrix_kind;               // symm, non-symm or complex
  int comp_mode;                 // regular, shift-and-invert et.c.
  int spectrum_part;             // indicates what part of spectrum to find
  Mat(NUMT) eigenvectors;        // store the eigenvectors here
```

```
  Vec(NUMT) eigenvalues;         // store the eigenvalues here
  Handle(Vec(NUMT)) V, W;        // aux vectors for matrix-vector operations
  real sigma, sigma_im;          // real (and imaginary) part of shift

  // linear equations objects.
  Handle(Matrix(NUMT)) C, D;
  Handle(LinEqSolver) linear_solver;
  Handle(LinEqSolver_prm) linear_solver_prm;
  Handle(LinEqAdm) lineq;
  bool c_has_changed;  // indicates whether we do the first linear solve or not


 public:
  // constructors...
  EigenSolver(Handle(MenuSystem) menu_handle = NULL);
  EigenSolver(Matrix(NUMT) &the_A, int the_nev = 1, Handle(MenuSystem) menu_handle = NULL);
  EigenSolver(Matrix(NUMT) &the_A, Matrix(NUMT) &the_B, int the_nev = 1, Handle(MenuSystem) menu_handle = NULL);
  // methods for setting the matrices
  bool setA(Matrix(NUMT)& the_A);
  bool setB(Matrix(NUMT)& the_A);

  // methods for setting/getting number of eigenvalues to seek
  bool setNev(int the_nev);
  int getNev();
  // set the matrix characteristics
  bool setMatrixKind(int kind);
  // set the real and imaginary part of the spectrum shift
  void setSigma(real the_sigma);
  void setSigmaIm(real the_sigma_im);
  // set the computational mode
  bool setCompMode(int mode);
  bool setSpectrumPart(int part);

  // solve the problem, silly report and silly printing of matrices.
  void solveProblem();
  void report();
  void sillyPrint(Matrix(NUMT)& matrisen);

  // retrieve references to the internal eigenthings storage
  Mat(NUMT)& getEigenvectors();
  Vec(NUMT)& getEigenvalues();

 protected:
  void resetAllMembers();     // clears everything, incl. pointers.
  // initialize the chain of objects constit. linear solver.
  void initLinearSolver(Handle(MenuSystem) menu_handle = NULL);
  void removeA();             // three methods that kill matrices...
  void removeB();
  void removeMatrix(Handle(Matrix(NUMT)) h);

  // matrix-vector operations methods. passed to ARPACK++ objects.
  void multAx(NUMT *, NUMT *);
  void multBx(NUMT *, NUMT *);
  void multInvCDx(NUMT *, NUMT *);
  void multInvCx(NUMT *, NUMT *);

}; // EigenSolver




#endif
```

B.4.2 EigenSolver.cpp

```
#include "EigenSolver.h"

// include the ARSymGenEig class template
#include "argsym.h"
#include "argcomp.h"

//
// Implementation of EigenSolver class.
//
// Author:
// Simen Kvaal.
//
// Currently only symmetric problems are supported.
// The rest of the problems may easily be
// added in the SolveProblem method.
//
// Last update:
// Aug. 12, 2003
//



//
// reset all members. helps keeping things clean.
//
void EigenSolver:: resetAllMembers() {
  removeA();
  removeB();
  n = 0;
  nev = 0;
  problem_kind = PROBLEM_UNDEFINED;
  matrix_kind = MATRIX_SYMMETRIC;
  comp_mode = MODE_REGULAR;
  spectrum_part = SPECTRUM_SMALLEST_MAGNITUDE;
  eigenvalues.redim(0);
  eigenvectors.redim(0,0);

  removeMatrix(C);
```

```
  removeMatrix(D);

  sigma = 0;
  sigma_im = 0;

}

//
// initialize chain of objecs constituting
// the linear solver.
//
void EigenSolver:: initLinearSolver(Handle(MenuSystem) menu_handle) {

  // create the LinEqSolver object
  linear_solver_prm.rebind( LinEqSolver_prm::construct() );
  if (menu_handle != NULL)
    linear_solver_prm->scan(*menu_handle);
  //initFromCommandLineArg("-s", linear_solver_prm->basic_method, "GaussElim");  /// commented out; i think we
        actually may use the settings in the menu system instead!
  linear_solver.rebind( linear_solver_prm->create() );


  // create LinEqAdm object for solving linear systems.
  lineq.rebind( new LinEqAdm(EXTERNAL_STORAGE) );
  lineq->attach( linear_solver() );

  c_has_changed = true;
}

//
// default constructor
//
EigenSolver:: EigenSolver(Handle(MenuSystem) menu_handle) {
  // make it clean...
  resetAllMembers();
  initLinearSolver(menu_handle);
  V.rebind( new Vec(NUMT)(0) );
  W.rebind( new Vec(NUMT)(0) );
}

//
// constructor for standard problems.
//
EigenSolver:: EigenSolver(Matrix(NUMT) &the_A, int the_nev, Handle(MenuSystem) menu_handle) {

  resetAllMembers();
  initLinearSolver(menu_handle);

  V.rebind( new Vec(NUMT)(0) );
  W.rebind( new Vec(NUMT)(0) );

  // attempt to set A.
  if (!setA(the_A)) {
    WARNING("Could not set A. Bailing out of constructor.");
    return;
  }

  nev = the_nev;

}

//
// constructor for generalized problems.
//
EigenSolver:: EigenSolver(Matrix(NUMT) &the_A, Matrix(NUMT) &the_B, int the_nev, Handle(MenuSystem) menu_handle) {

  resetAllMembers();
  initLinearSolver(menu_handle);

  V.rebind( new Vec(NUMT)(0) );
  W.rebind( new Vec(NUMT)(0) );

  // attempt to set A.
  if (!setA(the_A)) {
    WARNING("Could not set A. Bailing out of constructor.");
    return;
  }

  // attempt to set B.
  if (!setB(the_B)) {
    WARNING("Could not set B. Solver is now a standard solver.");

    nev = the_nev;

    return;

  } else {
    nev = the_nev;
  }

}

//
// remove matrices...
//
void EigenSolver:: removeA() {
  removeMatrix(A);
}
void EigenSolver:: removeB() {
  removeMatrix(B);
}
void EigenSolver:: removeMatrix(Handle(Matrix(NUMT)) h) {
  h.detach(); h.rebind(NULL);
}

//
// set matrices...
```

```
// also sets problem_kind and n, making a complete
// problem specification.
//
bool EigenSolver:: setA(Matrix(NUMT)& the_A) {
  int N, M;

  // get the matrix size.
  the_A.size(N, M);

  // the matrix must be square...
  if (N==M) {
    A.rebind( the_A );
    n = N;
  }
  else {
    return false;
  }

  // set problem kind.
  if (B.getPtr())
    problem_kind = PROBLEM_GENERALIZED;
  else
    problem_kind = PROBLEM_STANDARD;

  return true;

}

//
// set B. A must be already set.
//
bool EigenSolver:: setB(Matrix(NUMT)& the_B) {
  int N, M;

  // get the matrix size.
  the_B.size(N, M);

  // the matrix must be square and of same size as A...
  if ((N==M)&& (N==n)) {
    B.rebind( the_B );
    problem_kind = PROBLEM_GENERALIZED;
    return true;
  }
  else {
    return false;
  }

}

// set nev and return true if success.
// nev must be in range 1 .. n-1
bool EigenSolver:: setNev(int the_nev) {
  if ((the_nev >= 1) && (the_nev <= n-1)) {
    nev = the_nev;
    return true;
  } else {
    WARNING("nev is out of range.");
    return false;
  }
}

// get nev...
int EigenSolver:: getNev() { return nev; }

// set spectrum part...
bool EigenSolver:: setSpectrumPart(int part) {
  if ((part == SPECTRUM_SMALLEST_MAGNITUDE) ||
      (part == SPECTRUM_SMALLEST_ALGEBRAIC) ||
      (part == SPECTRUM_SMALLEST_REAL) ||
      (part == SPECTRUM_SMALLEST_IMAG) ||
      (part == SPECTRUM_LARGEST_MAGNITUDE) ||
      (part == SPECTRUM_LARGEST_ALGEBRAIC) ||
      (part == SPECTRUM_LARGEST_REAL) ||
      (part == SPECTRUM_LARGEST_IMAG)) {
    spectrum_part = part;
    return true;
  }
  WARNING("Invalid spectrum part.");
  return false;
}

// set matrix kind...
bool EigenSolver:: setMatrixKind(int kind) {
  if ((kind == MATRIX_SYMMETRIC) || (kind == MATRIX_NONSYMMETRIC) ||
      (kind == MATRIX_COMPLEX)) {

    matrix_kind = kind;

  } else {

    WARNING("Illegal matrix kind.");
    return false;

  }

  return true;

}

//
// solve the problem!
// note that only symmetric problems
// are implemented at this point,
// but adding more matrix kinds is
// really easy.
//
void EigenSolver:: solveProblem() {
```

```
bool supported_mode = false;

char *descriptive_array[] = { "SM", "SA", "SR", "SI", "LM", "LA", "LR", "LI" } ;
char *descriptive = descriptive_array[spectrum_part];

// create a pointer to the base class in the
// ARPACK++ hierarchy.
ARrcStdEig<real, real> *solver = NULL;
ARrcStdEig<real, arcomplex<real> > *solverComplex = NULL;
typedef void (EigenSolver::*real_multfunc)(real *, real *) ;
typedef void (EigenSolver::*Complex_multfunc)(arcomplex<real> *, arcomplex<real> *) ;


// create instance of proper
// class based on matrix kind and problem kind
// and regular mode.

// attempt at complex matrix implementation. seems to work!
if (matrix_kind == MATRIX_COMPLEX) {
  if (problem_kind == PROBLEM_STANDARD) {
    if (comp_mode == MODE_REGULAR) {
      ARCompStdEig<real, EigenSolver> *temp =
        new ARCompStdEig<real, EigenSolver>(n, nev, this, (Complex_multfunc)&EigenSolver:: multAx, descriptive);

      solverComplex = temp;
      supported_mode = true;
      WARNING("ARCompStdEig in regular mode created.");

    }

  }

  if (problem_kind == PROBLEM_GENERALIZED) {
    if (comp_mode == MODE_REGULAR) {
      // use the regular mode constructor.
      ARCompGenEig<real, EigenSolver, EigenSolver> *temp =
        new ARCompGenEig<real, EigenSolver, EigenSolver>(n, nev, this, (Complex_multfunc)&EigenSolver:: multInvCDx,
                                                         this, (Complex_multfunc)&EigenSolver:: multBx, descriptive)
                                                               ;

      solverComplex = temp;
      // set up helper matrices.
      D.rebind( A() );
      B->makeItSimilar( C );
      C() = B();  // note different linear system in the regular case.
      supported_mode = true;
      WARNING("ARCompGenEig in regular mode created.");

      supported_mode = true;
     }
   }


}

// symmetric problems. --> requires real matrices!

if (matrix_kind == MATRIX_SYMMETRIC) {

  // standard problems.
  if (problem_kind == PROBLEM_STANDARD) {
    if (comp_mode == MODE_REGULAR) {
      // use the regular mode constructor
      // must cast pointer due to different NUMT ...
      ARSymStdEig<real, EigenSolver> *temp =
        new ARSymStdEig<real, EigenSolver>(n, nev, this, (real_multfunc)&EigenSolver:: multAx, descriptive);
      solver = temp;
      supported_mode = true;
      WARNING("ARSymStdEig in regular mode created.");
    }
    if (comp_mode == MODE_SHIFT_INVERT) {
      // use the shift-and-invert mode constructor
      ARSymStdEig<real, EigenSolver> *temp =
        new ARSymStdEig<real, EigenSolver>(n, nev, this, (real_multfunc)&EigenSolver:: multInvCx, sigma,
              descriptive);
      solver = temp;
      A->makeItSimilar(C);
      Handle(Matrix(NUMT)) Eye; A->makeItSimilar(Eye); Eye() = 0.0;
      for (int i=1; i<=n; i++) Eye->elm(i,i) = 1.0;
      add(C(), A(), -sigma, Eye());  // C = A - sigma*I
      supported_mode = true;
      WARNING("ARSymStdEig in shift-and-invert mode created.");
    }
  }

  // generalized problems.
  if (problem_kind == PROBLEM_GENERALIZED)  {
    if (comp_mode == MODE_REGULAR) {
      // use the regular mode constructor.
      ARSymGenEig<real, EigenSolver, EigenSolver> *temp =
        new ARSymGenEig<real, EigenSolver, EigenSolver>(n, nev, this, (real_multfunc)&EigenSolver:: multInvCDx,
                                                        this, (real_multfunc)&EigenSolver:: multBx, descriptive);

      solver = temp;
      // set up helper matrices.
      D.rebind( A() );
      B->makeItSimilar( C );
      C() = B();  // note different linear system in the regular case.
      supported_mode = true;
      WARNING("ARSymGenEig in regular mode created.");
    }

    if (comp_mode == MODE_SHIFT_INVERT) {
      // use shift-and-invert-mode constructor.
      ARSymGenEig<real, EigenSolver, EigenSolver> *temp =
        new ARSymGenEig<real, EigenSolver, EigenSolver>('S', n, nev, this, (real_multfunc)&EigenSolver::multInvCx,
```

```
                                                    this, (real_multfunc)&EigenSolver::multBx, sigma,
                                                          descriptive);
        solver = temp;
        // set up linear system.
        B->makeItSimilar( C );
        add(C(), A(), -sigma, B());
        supported_mode = true;
        WARNING("ARSymGenEig in shift-and-invert mode created.");
      }
      if (comp_mode == MODE_BUCKLING) {
        ARSymGenEig<real, EigenSolver, EigenSolver> *temp =
          new ARSymGenEig<real, EigenSolver, EigenSolver>('B', n, nev, this, (real_multfunc)&EigenSolver::multInvCx,
                                                    this, (real_multfunc)&EigenSolver::multAx, sigma,
                                                          descriptive);
        solver = temp;
        // set up linear system.
        B->makeItSimilar( C );
        add(C(), A(), -sigma, B());
        supported_mode = true;
        WARNING("ARSymGenEig in buckling mode created.");
      }
      if (comp_mode == MODE_CAYLEY) {
        // use buckling mode constructor.
        ARSymGenEig<real, EigenSolver, EigenSolver> *temp =
          new ARSymGenEig<real, EigenSolver, EigenSolver>(n, nev, this, (real_multfunc)&EigenSolver::multInvCx,
                                                    this, (real_multfunc)&EigenSolver::multAx,
                                                    this, (real_multfunc)&EigenSolver::multBx, sigma,
                                                          descriptive);
        solver = temp;
        // set up linear system.
        B->makeItSimilar( C );
        add(C(), A(), -sigma, B());
        supported_mode = true;
        WARNING("ARSymGenEig in Cayley mode created.");
      }
    }
  }

  if (!supported_mode) {
    WARNING("Your computational mode is not supported! Sorry.");
    return;
  }


  // find the eigenvalues and eigenvectors.
  // copy them to internal arrays inside EigenSolver.

  void *eval_ptr;
  void *evec_ptr;

  int converged;

  if (solver) {
    solver->FindEigenvalues();
    solver->FindEigenvectors();
    eval_ptr = (void *)solver->RawEigenvalues();
    evec_ptr = (void *)solver->RawEigenvectors();
    converged = solver->ConvergedEigenvalues();
  }

  if (solverComplex) {
    solverComplex->FindEigenvalues();
    solverComplex->FindEigenvectors();
    eval_ptr = (void *)solverComplex->RawEigenvalues();
    evec_ptr = (void *)solverComplex->RawEigenvectors();
    converged = solverComplex->ConvergedEigenvalues();
  }



  eigenvalues.redim(converged);
  eigenvectors.redim(converged, n);

  for (int i=0; i<converged; i++)
    eigenvalues(i+1) = ((NUMT *)eval_ptr)[i];

  for (int i=0; i<converged; i++)
    for (int j=0; j<n; j++)
      eigenvectors(i+1, j+1) = ((NUMT *)evec_ptr)[n*i + j];



  // clean up memory.
  if (solver) delete solver;
  if (solverComplex) delete solverComplex;

}


//
// matrix-vector multiplication: w <- Av
//
void EigenSolver:: multAx(NUMT *v, NUMT *w) {

  s_e << ".";

   // matrix-vector operations methods. passed to ARPACK++ objects.
  void multAx(real *, real *);
  void multBx(real *, real *);
  void multInvCDx(real *, real *);
  void multInvCx(real *, real *);
 // let V use v as underlying pointer, and W use w.
  V->redim(v, n);
  W->redim(w, n);

  // multiply: y = Ax
  A->prod(V(), W());
```

```
}


//
// matrix-vector multiplication: w <- Bv
//
void EigenSolver:: multBx(NUMT *v, NUMT *w) {

  s_e << ".";
  // let V use v as underlying pointer, and W use w.
  V->redim(v, n);
  W->redim(w, n);

  // multiply: y = Ax
  B->prod(V(), W());

  //sillyPrint(V()); s_o << " --- Bx ---> ";
  //sillyPrint(W()); s_o << endl;


}

//
// matrix-vector multipliction: w <- inv(C)Dv,  v <- Av
//
void EigenSolver:: multInvCDx(NUMT *v, NUMT *w) {

  s_e << "*";

  //let V use v and W use w as underlying pointer.
  V->redim(v, n);
  W->redim(w, n);

  // multiply: y = Ax
  D->prod(V(), W());

  // let v <-- w
  V() = W();

  // solve w = B^-1 y

  //             C    x  =   b
  lineq->attach(C(), W(), V());
  lineq->solve( c_has_changed );
  if (c_has_changed) c_has_changed = false;

}

//
// matrix-vector multipliction: w <- inv(C)v
//
void EigenSolver:: multInvCx(NUMT *v, NUMT *w) {

  s_e << "*";

  //let V use v and W use w as underlying pointer.
  V->redim(v, n);
  W->redim(w, n);

  // solve w = B^-1 y

  //             C    x  =  b
  lineq->attach(C(), W(), V());  // C W = V
  lineq->solve( c_has_changed );
  if (c_has_changed) c_has_changed = false;


}



//
// make a simple report, displaying matrices and so on.
//
void EigenSolver:: report() {

  s_o << " n   == " << n << endl;
  s_o << " nev == " << nev << endl << endl;
  s_o << " sigma == " << sigma << endl;

  s_o << " A == " << endl;
  if (A.getPtr())
    sillyPrint(A());
  else
    s_o << "[ undefined ]";

  s_o << " B == " << endl;
  if (B.getPtr())
    sillyPrint(B());
  else
    s_o << "[ undefined ]";

  s_o << endl;
}

//
// method that prints matrix in a silly way
//
void EigenSolver:: sillyPrint(Matrix(NUMT)& matrisen) {
  int m, n;

  matrisen.size(n, m);

  s_o << "[";
```

```
  for (int i=1; i<=n; i++)
    for (int j=1; j<=m; j++) {
      s_o << matrisen.elm(i,j);
      if ((i==n) && (j==m))
        s_o << "]" << endl;
      else
        s_o << ", ";
    }
}


//
// set the shift
//
void EigenSolver:: setSigma(real the_sigma) { sigma = the_sigma; }
void EigenSolver:: setSigmaIm(real the_sigma_im) { sigma = the_sigma_im; }

//
// sets the computational mode.
//
bool EigenSolver:: setCompMode(int mode) {
  bool success = false;

  // regular mode and shift-and-invert-modes are always ok.
  if ((mode == MODE_REGULAR) || (mode == MODE_SHIFT_INVERT))
    success = true;

  // two more modes for generalized, symmetric problems.
  if ((problem_kind == PROBLEM_GENERALIZED) && (matrix_kind == MATRIX_SYMMETRIC)) {
    if ((mode == MODE_CAYLEY) || (mode == MODE_BUCKLING))
      success = true;
  }

  // one more mode for generalized, non-symmetric problems.
  if ((problem_kind == PROBLEM_GENERALIZED) && (matrix_kind == MATRIX_NONSYMMETRIC)) {
    if (mode == MODE_COMPLEX_SHIFT)
      success = true;
  }

  if (success)
    comp_mode = mode;

  return success;
}

//
// get references to eigenthings.
//
Mat(NUMT)& EigenSolver:: getEigenvectors() { return eigenvectors; }
Vec(NUMT)& EigenSolver:: getEigenvalues() { return eigenvalues; }
```

Bibliography

[1] Home page for FemLab. Web address: *http://www.comsol.com/*.

[2] C. Bottcher. Accurate quantal studies of ion-atom collisions using finite-element techniques. *Phys. Rev. Lett.*, 48:85–88, 1982.

[3] T.N. Rescigno and C.W. McCurdy. Numerical grid methods for quantum mechanical scattering problems. *Phys. Rev. A*, 62, 2000.

[4] F.S. Levin and J. Schertzer. Finite-element solution of the schrödinger equation for the helium ground state. *Phys. Rev. A*, 32:3285–3290, 1985.

[5] S. Kvaal. Home page for this *cand. scient.* project. Web address: *http://www.fys.uio.no/~simenkva/hovedfag/*.

[6] H. Goldstein. *Classical Mechanics*. Addison Wesley, 1970.

[7] R. Shankar. *Principles of Quantum Mechanics, 2nd ed.* Plenum, 1994.

[8] Wikipedia – the free encyclopedia. Web address: *http://www.wikipedia.org/*.

[9] M.C. Gutzwiller. *Chaos in Classical and Quantum Mechanics*. Springer, 1990.

[10] J.J. Brehm. *Introduction to the Structure of Matter*. Wiley, 1989.

[11] H. Kragh. *Quantum Generations: A History of Quantum Physics in the Twentieth Century*. Princeton Univ. Press, 1999.

[12] B.E. Lian and O. Øgrim. *Størresler og enheter i fysikk*. Universitetsforlaget, 2000.

[13] G.A Biberman, N. Sushkin, and V. Fabrikant. *Dokl. Akad. Nauk*, 26:185, 1949.

[14] L. de Broglie. *Ann. Phys., Lpz.*, 10:22, 1925.

[15] J.E. Marsden and M.J. Hoffman. *Elementary Classical Analysis*. Freeman, 1993.

[16] S.C. Brenner and L.R. Scott. *The Mathematical Theory of Finite Element Methods*. Springer, 1994.

[17] S. Larsson and V. Thomée. *Partial Differential Equations with Numerical Methods*. Springer, 2003.

[18] A. Galindo and Pascual P. *Mecánica Quántica*. Alhambra, 1978.

[19] J.M. Leinaas and Myrheim J. On the theory of identical particles. *Il Nuovo Cimento*, 37B:1, 1977.

[20] J.-L. Basdevant and Dalibard J. *Quantum Mechanics*. Springer, 2002.

[21] E.D. Jackson. *Classical Electrodynamics*. Wiley, 1962.

[22] P.C. Hemmer. *Kvantemekanikk, 2nd ed.* Tapir, 2000.

[23] J. Frøyland. *Forelesninger i klassisk teoretisk fysikk II.* UNIPUB, 1996.

[24] K. Rottmann. *Matematisk Formelsamling.* Spektrum forlag, 1999.

[25] G. Paz. On the connection between the radial momentum operator and the hamiltonian in n dimensions. *Eur. J. Phys.*, 22:337, 2000.

[26] S. Brandt and H.D. Dahmen. *The Picture Book of Quantum Mechanics.* Springer, 1995.

[27] J.M. Leinaas. *Non-relativistic Quantum Mechanics – Lecture notes in FYS4110.* (Unpublished), 2003.

[28] C. Kittel. *Introduction to Solid State Physics, 6th ed.* Wiley, 1986.

[29] A.H. MacDonald and D.S. Ritchie. *Phys. Rev. B*, 33:8336, 1986.

[30] M. Robnik and V. G. Romanovski. Two-dimensional hydrogen atom in a strong magnetic field. *J. Phys. A: Math. Gen.*, 36:7923–7951, 2003.

[31] L.R. Ram-Mohan. *Finite Element and Boundary Element Applications in Quantum Mechanics.* Oxford, 2002.

[32] R.C. McOwen. *Partial Differential Equations; Methods and Applications.* Prentice-Hall, 2003.

[33] A. Tveito and R. Winther. *Introduction to Partial Differential Equations – a Computational Approach.* Springer, 1998.

[34] H.P. Langtangen. *Computational Partial Differential Equations.* Springer Verlag, 2001.

[35] A. Goldberg, H.M. Schey, and J.L. Schwartz. Computer-generated motion pictures of one-dimensional quantum-mechanical transmission and reflection phenomena. *Am. J. Phys.*, 35:177, 1967.

[36] P.C. Moan and S. Blanes. Splitting methods for the time-dependent schrödinger equation. *Phys. Lett. A*, 265:35–42, 2000.

[37] A. Askar and A.S. Cakmak. Explicit integration method for the time dependent schrödinger equation for collision problems. *J. Chem. Phys.*, 68:7294, 1978.

[38] L.D. Landau and E.M. Lifshitz. *Course of Theoretical Physics Volume 1: Mechanics.* Butterworth, 1976.

[39] J.B. Fraleigh and R.A. Beauregard. *Linear Algebra, 3rd ed.* Addison-Wesley, 1995.

[40] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C++, Second Edition.* Cambridge, 2002.

[41] Y. Saad. *Numerical Methods for Large Eigenvalue Problems.* Wiley, 1992.

[42] Home page for inutech. Web address: *http://www.inutech.de/.*

[43] Home page for numerical objects as. Web address: *http://www.nobjects.com/.*

[44] Home page for simula research laboratory. Web address: *http://www.simula.no/.*

[45] Online Diffpack documentation. Web address:
http://www.nobjects.com/diffpack/refmanuals/current/classes.html.

[46] H.P. Langtangen, N.N.G. Pedersen, K. Samuelsson, and H. Semb. Finite element preprocessors in diffpack. The Numerical Objects Report Series #1999-01, February 1, 2001.

[47] Various Authors. *ARPACK User Guide and ARPACK++ Reference Manual*. Available online from *http://www.ime.unicamp.br/~chico/arpack++/*.

[48] R.L. Liboff. *Introductory Quantum Mechanics, 4th ed.* Addison-Wesley, 2003.

[49] W. Dörfer. A time- and spaceadaptive algorithm for the linear time-dependent schrödinger equation. *Numer. Math.*, 73:419–448, 1996.

[50] L.B. Madsen. Gauge invariance in the interaction between atoms and few-cycle laser pulses. *Phys. Rev. A*, 65:053417, 2002.

[51] D. Leibfried, R. Blatt, C. Monroe, and Wineland. Quantum dynamics of single trapped ions. *Rev. Mod. Phys.*, 75:281–306, 2003.

[52] J. Schertzer. Finite-element analysis of hydrogen in superstrong magnetic fields.

[53] J. Schertzer, L.R. Ram-Mohan, and D. Dossa. Finite-element analysis of low-lying states of hydrogen in superstrong magnetic fields.

[54] H. Møll Nilsen. *Aspects of the Theory of Atoms and Coherent Matter and Their Interaction With Electromagnetic Fields*. PhD thesis, University of Bergen, 2002.

[55] S. Albeverio, J.E. Fenstad, H. Holden, and T. Lindstrøm, editors. *Ideas and Methods in Quantum and Statistical Physics*. Cambridge, 1992.

[56] T.F. Jordan. *Linear Operators for Quantum Mechanics*. Wiley, 1969.