PyProp - A Python Framework for Propagating the Time Dependent Schrödinger Equation

Tore Birkeland

Dissertation for the degree of Philosophiae Doctor (PhD)

Department of Mathematics
University of Bergen

September 2009

Acknowledgements

In contrary to the deceiving title page of this dissertation, the work in here is not due to myself alone. I have received considerable help, support, encouragement and otherwise from a number of people, all deserving my gratitude.

First and foremost I would like to thank my friends and co-workers Raymond Nepstad and Lene Sælen, whose help have been absolutely invaluable. The countless hours we have spent discussing each others problems have indubitably helped our understanding. More importantly, our collaboration has created a wonderful atmosphere in the office, making even the worst days enjoyable. I would also like to thank Raymond for a great semester in Belfast with many adventures, late night programming sessions and Father Ted at The Parlour.

My supervisor, Tor Sørevik deserves a big thanks for his support and encouragement. He has always had time to discuss whatever problems I have had, and has suggested new research opportunities on numerous occasions. I am also very grateful the ability of my co-supervisor Jan Petter Hansen to cut through the chaff and making sure things get done. Everybody in the Bergen Computational Quantum Mechanics Group (BCQPG) deserves a collective thank you for many interesting lunch seminars and discussions. I have had many wonderful discussions with Morten Førre, improving my understanding of atomic physics. Thank you for your patience.

During my PhD I had the pleasure of staying at Queen's University Belfast for six months with Jim McCann as my supervisor. Jim's enthusiasm is unrivaled, and we had a great collaboration on molecular vibration. While staying in Belfast I had the pleasure of meeting a lot wonderful people, and in particular Chris Calvert, who taught me all I know about experimental atomic physics as well as the finer pints of Irish culture.

I am grateful to the Nano UiB project for funding this work. I would also like to thank The Center for the Study of the Sciences and the Humanities (SVT) for financially supporting my stay in Belfast, and in particular Roger Strand for providing a most interesting side track to my regular research activities.

Finally, I thank Kjersti with all my ♡!

List of papers

1. T. Birkeland and T. Sørevik *Parallel Redistribution of Multidimensional Data* Proceedings ParCo 2007 Conference NIC Series Volume **38**, 2007

2. V. Popsueva, R. Nepstad, T. Birkeland, M. Førre, J. P. Hansen, E. Lindroth, and E. Waltersson
Structure of lateral two-electron quantum dot molecules in electromagnetic fields Physical Review B, **76**, 035303 (2007)

3. L. Sælen, I. Sundvor, T. Birkeland, S. Selstø, and M. Førre
Classical and quantum-mechanical investigation of the role of nondipole effects on the binding of a stripped HD^{2+} molecule Physical Review A, **76**, 013415 (2007)

4. T. Sørevik, T. Birkeland and G. Okša
Numerical solution of the 3D time dependent Schrödinger equation in spherical coordinates: Spectral basis and effects of split-operator technique Journal of Computational and Applied Mathematics, **225**, (2007) 56-67

5. C. R. Calvert, T. Birkeland, R. B. King, I. D. Williams and J. F. McCann
Quantum chessboards in the deuterium molecular ion J. Phys. B: At. Mol. Opt. Phys., **41**, 205504 (2008)

6. T. Birkeland and T. Sørevik
Parallel Pseudo-spectral Methods for the Time Dependent Schrödinger Equation Parallel Computing: Numerics, Applications, and Trends R. Trobec, M Vajteršic and P. Zinterhof (ed.) 2009, p261-279

7. T. Birkeland, R. Nepstad and M. Førre
Multiphoton Double Ionization in Helium Driven by Intense XUV Attosecond Pulses, to be submitted

8. L. Sælen, T. Birkeland, N. Sisourat, A. Dubois and J. P. Hansen
Non perturbative treatment of single ionization of H_2 by fast, highly charged ion impact, to be submitted

Contents

Chapter 1

Introduction

Numerical analysis and computations have been an integral part of science from the development of calculus in the 17th century. It was discovered early on that the differential equations of classical mechanics in many cases were too complicated to be solved analytically; even for the three-body problem of celestial mechanics (the sun-earth-moon orbit), analytical solutions can not be found, and approximations must be used.

In 1768, Leonhard Euler published a canonical scheme for solving differential equations, based on the Taylor expansion, which has later become known as "Euler's Method". This method allowed the motion of complicated planetary systems to be studied as initial value problems.

With the convergence proof of the backward Euler method by Augustin Louis Cauchy in 1824, development of more accurate integration methods started, leading to the discovery of multi-step methods and Runge-Kutta methods during the 19th and 20th centuries. Although these methods could be constructed such as to be very accurate every timestep, it was observed that for simulations of extended duration, such as predicting the trajectories of the celestial objects for centuries, the calculated solution could slowly drift away from the exact solution. On the other hand, lower order integration schemes like the Störmer-Verlet[1] method did not experience this energy drift even if their accuracy was seemingly lower than that of higher order methods. The understanding that properties other than order of accuracy and stability are of importance was gradually realized in different fields during the 20th century. In the 1990s, this direction of research developed into a separate field, geometric numerical integration[28].

Along with the development of computational methods emerged the profession of computers. These early computers were not electronic devices like today, but rather humans performing numerical calculations by hand. The limited speed of performing calculations with pen-and-paper restricted the use, and therefore development, of advanced numerical schemes until the development of first mechanical computers, and later electronic computers. These computational devices opened up new possibilities in numerical research, making increasingly more sofisticated methods feasable.

Today, a plethora of schemes for solving differential equations are available, and there is a challenge in finding the scheme most suitable for the problem at hand. Physicists and engineers rely on numerical techniques for everyday problem solving, but

[1] Although originally devised by Newton in Principia, this method has been reinvented many times and is often attributed to Carl Störmer in astronomy and Loup Verlet in molecular dynamics.

do not have the time or resources to investigate the multitude of methods thoroughly. Mathematicians and numerical analysts, on the other hand, have plenty of resources to study the numerical methods, but often lack interesting problems to test the methods on. This leads to a less than optimal amount of technological transfer from numerical analysis to applications in physics and engineering. One possible remedy for this problem is to create software frameworks catering both to developers and users of numerical schemes. ProtoMol[44] is such a framework for molecular dynamics. It serves numerical analysts as a framework for simplifying development and testing of numerical schemes, while at the same time being efficient and scalable enough to support physically interesting problems.

By solving the equations of classical mechanics, scientists had huge success in predicting planetary trajectories, the motion of rigid bodies, etc. Despite this success, it became inceasingly clear towards the end of the 19th century that classical mechanics was not sufficient to describe motion on the atomic level. It was discovered[62] that instead of assigning position and momentum to each particle, one should rather consider all possible values of position and momentum simultaneously. This consideration turns the equations of motion from a set of coupled ordinary differential equations into a partial differential equation (PDE) called the time dependent Schrödinger equation (eqn 1.4). In the Schrödinger equation, the number of particles correspond to degrees of freedom in the PDE, leading to exponential increase with the number of particles. While classical calculations can consider millions of particles simultaneously, the "exponential wall" of quantum mechanics[37] effectively limits the number of particles that can be considered in quantum mechanical calculations to a handful. It is only in the last 10-15 years that advances in computational power have made full two-electron studies of helium, molecular hydrogen and other two electron systems possible. Systems with more than two or three particles are still out of reach on modern computers and approximations such as density functional theory[38] or Hartree-Fock[11, p. 382] must be used.

In understanding the fundamental processes of atomic systems, numerical simulations are invaluable. As physical experiments grow increasingly more complex, it is important to complement the experiments with numerical experiements in order to better understand underlying physics. Numerical and physical experiments are usually not hard in the same manner. Physical experiments are limited in what kind of information can be extracted from a system, as well as the amount of control one have over the experimental parameters. In numerical experiments, on the other hand, the wavefunction is available, and can any kind of observable can in principle be calculated. The difficulty here is to create physically realistic scenarios and extracting physically relevant information, as these often require long simulation times on a large domains, leading to high requirements of memory and computational time.

Simulating large quantum mechanical systems require good methods and efficient implementation. The fantastic development in computers has allowed increasingly larger systems to be considered numerically. The current trend is no longer towards faster processing units, but rather towards *more* processing units. Numerical methods working well on these parallel machines are therefore required if simulations are to be performed on systems of ever larger size. Such numerical methods can become quite complex, and the physicists interested in numerical simulations are not necessarily interested in the details of the numerical methods. It is therefore important to have

software which simplifies the application of numerical methods to interesting physical problems.

Scripting languages, and in particular Python, have become popular for writing scientic software in recent years. An increasing number of scientific codes are written in Python or have wrappers which allow them to be called from Python[32, 48]. There is even an annual conference on scientific programming in Python[63]. Furthermore, an entire issue of the prestigous IEEE Computing in Science and Engineering (May/June 2007) was recently dedicated to the use of Python in scientific and engineering software. Much of this increased popularity is undoubtedly due to the simplicity of making Python extension modules, and the availability of high quality extension modules such as NumPy, SciPy[53] and Matplotlib[34], providing visualizations, numerical arrays, linear algebra and more.

The main part of this thesis is dedicated to explaining some of the ideas behind the software package PyProp. As will be discussed in in the following chapters, PyProp is a software package designed to aid researchers in physics in solving the time dependent Schrödinger equation. It is distributed under the GNU General Public License (GPL)[22], and is publicly available[8].

1.1 Fundamentals of Quantum Mechanics

A textbook[30] on quantum mechancis should be consulted for a proper introduction to quantum mechanics. Nevertheless, we here give a short overview of some of the principles of quantum mechanics which will be useful for the reminder of this dissertation.

In quantum mechanics, the state of a system is described by a complex-valued *wavefunction*. For n particles the wavefunction can be written

$$\psi(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N, t). \tag{1.1}$$

The square of the wavefunction is related to probability density, and the probability for finding particle 1 at position $\mathbf{x}_1$, particle 2 at position $\mathbf{x}_2$, etc. is $|\psi(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n, t)|^2$.

For a full description of a system a Hamiltonian is required in addition to the wavefunction. The Hamiltonian describes the energies of the system, and is related to the classical Hamiltonian by rewriting the canonical position an momentum to the corresponding quantum mechanical operators[2].

$$\begin{aligned} \mathbf{p}(t) &\rightarrow \mathbf{p} = -\mathrm{i}\nabla \\ \mathbf{q}(t) &\rightarrow \mathbf{q} = \mathbf{x}. \end{aligned} \tag{1.2}$$

The Hamiltonian is in general a Hermitian linear differential operator, and can usually be written in terms of the kinetic energy $\mathbf{T} = -\nabla^2/2m$, and potential energy, $\mathbf{V} = V(\mathbf{x}, t)$, operators.

$$\mathbf{H} = \mathbf{T} + \mathbf{V} \tag{1.3}$$

The time dependent Schrödinger equation (TDSE) describes the time evolution of the wavefunction under a given Hamiltonian,

$$\mathrm{i}\frac{\partial}{\partial t}\psi(\mathbf{x}, t) = \mathbf{H}\psi(\mathbf{x}, t). \tag{1.4}$$

Developing methods and software for solving this equation is the main topic of this dissertation.

An important postulate in quantum mechanics is that the only possible observable values corresponding to an operator $\mathbf{O}$, are the eigenvalues of that operator. As the Hamiltonian is the operator corresponding to energy in the system, the eigenvalues and eigenstates (eigenpairs) of the Hamiltonian have special importance. The eigenvalue equation for the Hamiltonian is often called the time *independent* Schrödinger equation (TISE)

$$\mathbf{H}\psi_n(\mathbf{x}) = E_n\psi_n(\mathbf{x}) \tag{1.5}$$

Eigenstates represent the stationary states of the system, such that if the wavefunction is purely in an eigenstate, it will remain so during time evolution, with the phase rotating according to the energy of the state,

$$\psi_n(\mathbf{x}, t) = \psi_n(\mathbf{x}, 0)e^{-\mathrm{i}E_n t}. \tag{1.6}$$

Another important quantity is the the expectation value of an operator,

$$\langle \mathbf{O}(t) \rangle = \int_\Omega \psi^*(\mathbf{x}, t)\mathbf{O}(t)\psi(\mathbf{x}, t)\mathrm{d}\mathbf{x}. \tag{1.7}$$

[2]Here, as everywhere else in this dissertation, atomic units[65] are used.

The expectation value represents the average value of an observable if large number of identical experiments are performed. As quantum mechanical systems approach the limit where a classical description is applicable, the the expectation value of position and momentum should approach their classical counterparts.

Chapter 2

Building a General TDSE Solver

There are numerous packages available for computations in atomic physics. These packages are ranging from large frameworks based on a variety of methods like Density Functional Theory or Hartee-Fock to calculate structures for very general systems, to small programs performing a limited set of calculations on specific systems.

From a physicist's point of view, the specialized programs can be very useful. If their scope fit the research at hand, a small program is easily understood, and can perhaps be tweaked to give the desired result. From a software standpoint, however, specialized programs are often made with only a specific result in mind, and are therefore often not easily extensible to other problems. Furthermore, as more features are added without an overall design goal, the programs can become overly complex and difficult to use.

The larger atomic physics software packages, such as Gaussian, Dalton, etc.[26] have a more thorough design philosophy, allowing broader development without introducing unnecessary complexity. However, these packages are mostly geared towards the time *independent* case, that is, calculating some or all eigenstates of a system.

In creating a general solver for time dependent problems, PyProp could focus exclusively on being a black box delivering wavefunctions after propagating a given initial state through a time dependent problem. We realize, however, that producing a final wavefunction is often only a small part of the problem. In many cases the amount of effort to properly analyze the result can be quite comparable to the actual propagation. Furthermore, the analysis needed in time dependent problems varies greatly between projects. PyProp is therefore designed to simplify the analysis process in addition to delivering robust propagators for a wide variety of systems. This is achieved by exposing most of the functionality through a Python interface, with which the user can create a specialized program for both propagation and analysis.

2.1 Fundamental concepts in PyProp

The main idea behind our software package PyProp has been to develop a research tool for time dependent atomic physics. The software should allow researchers to quickly start calculations on a new system with a minimal amount of work. As the desired quantities from time dependent calculations varies considerably from project to project, it is not a goal for PyProp to be a plug-and-play system, able to automatically calculate all kinds of quantities without any kind of programming from the user. Rather, the goal is

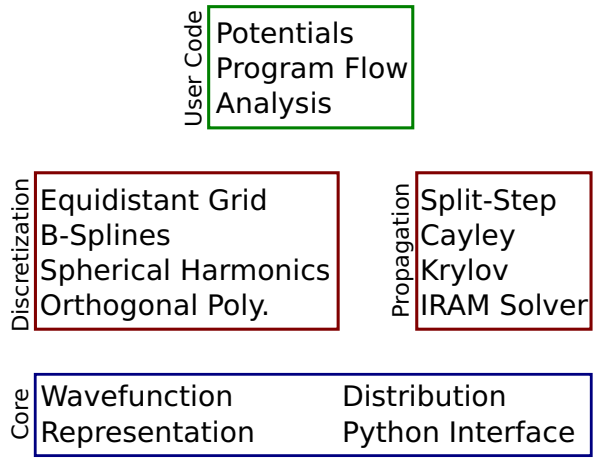

Figure 2.1: High level overview of PyProp. A core library (blue) is the basis on which a number of independent discretization and propagation schemes (red) are built. The user supplies the problem specific code (green) which may be dependent on the chosen schemes as well as the core library.

to provide tools for making it possible for researchers to implement the desired analysis without having detailed knowledge of all the numerical aspects of the propagation.

Problems in atomic physics can easily become very large. It is therefore paramount that the most efficient methods are used. In our case, "method" involves choice of coordinate system, type of discretization as well as propagation algorithm. PyProp must therefore support a number of methods in order to cater to a variety of problems. It is very hard to determine which method will be most suited for a particular problem in advance. PyProp should therefore make it reasonably simple to test different methods before settling on one, instead of relying on a priori knowledge about the system.

In figure 2.1, the high level design of PyProp is shown. A set of core routines contains the basic building blocks: The Wavefunction class contains the discretized wavefunction data, while the Representation class is a common interface to all types of discretizations. The actual implementation of the different discretization and propagation schemes are independent of each other, and depend only on the core routines. On top of these two layers, the user can add the problem dependent routines, deciding the actual flow of the program and performing analysis.

2.1.1 Method of Lines

An important aspect of creating a flexible time dependent PDE solver is realizing the difference between the spatial and the temporal dimensions. While there can be many spatial dimensions to a problem, there is always one temporal dimension. Furthermore, in the temporal dimension, the wavefunction is known initially at a time t_0, and the purpose of the propagation is to determine the wavefunction at a later time $t_1 > t_0$. The spatial dimensions, on the other hand, are surrounded by boundary conditions at both lower and upper boundary. It is therefore natural to use the method of lines [29], and

first discretize the spatial dimensions, reducing the PDE to a large set of coupled ODEs, and then discretize the temporal dimension.

2.1.2 Wavefunction

In PyProp, the wavefunction object plays a central role. It represents a self-contained snapshot of the discretized wavefunction at a point in time. It contains information about what kind of representation and discretization the numerical values represent, as well as how these values are distributed across processors. This makes it possible to write quite general routines for calculating various quantities, as the wavefunction contains the necessary information (grid points, integration weights and distribution information) through the representation and distributed model objects.

The wavefunction is said to have a certain rank, which indicates the number of independent discretizations that are combined in the current system. For example, a Cartesian coordinate system will typically have the same rank as the number of degrees of freedom, while a spherical coordinate system will typically have rank = 2; one rank for the radial dimension, and another for the two angular dimensions. The two angular dimensions (θ, ϕ) will typically not be split into different ranks, because in the spherical harmonic representation (discussed in section 4.2.3), the l and m quantum numbers do not form a tensor product, and as the wavefunction can not change rank during propagation, it is best to keep θ and ϕ compressed into one rank.

2.1.3 Representation

Every wavefunction object is associated with a representation object, representing the discretization details of the wavefunction. The representation has methods for extracting the grid points (for grid based discretizations), integration weights, as well as methods for calculating the inner product of two wavefunctions.

Most commonly, the CombinedRepresentation (a specialization of the Representation class) is used as the representation of the wavefunction. The CombinedRepresentation allows every rank to have independent sub-representations, enabling users to pick suitable discretizations for each rank. This flexibility allows PyProp to support a number of different systems, ranging from simple reduced dimensionality models, to full six degrees of freedom treatment of helium, as described in Paper **VII**.

During the course of propagation, the wavefunction can be transformed between representations. This includes transformations between a Fourier-space and a grid-space representation, between a spherical harmonic and spherical grid representation, or between any other representations for which a transformation has been defined. Under such operations, the actual representation objects attached to a wavefunction will be replaced, and the representation objects can therefore not be considered immutable.

2.1.4 Distributed Model

An important aspect in designing PyProp was to make parallelization work as automatically and transparently as possible. A project using only standard functionality in PyProp, should to a large extent not notice whether it is run in parallel or not. Furthermore, the high performance computer systems currently available are almost

exclusively distributed memory machines. On these machines, the Message Passing Interface (MPI) is by far the most popular parallelization interface available, and the parallelization schemes in PyProp are therefore based on MPI.

In order to facilitate a mostly transparent MPI parallelization, every representation is given a DistributedModel, which describes how the wavefunction is distributed among the processors. The parallelization scheme used is to distribute $d < \text{Rank}$ ranks of the wavefunction as evenly as possible on a d-dimensional grid of processors. The shape of the processor grid is chosen by the MPI implementation, and could possibly exploit structure in the underlying network hardware. By leaving one of the ranks local, operations requiring access to all elements along a rank can be performed on the local rank, and by applying the redistribution scheme described in Paper **I**.

2.2 Implementation

As the goal for PyProp is to be a flexible and yet highly performing TDSE solver, certain care has to be taken when choosing programming languages and libraries. The Fortran languages, for example, have a proven history of very high performance; the language is reasonably simple, it is geared towards efficient loops and array operations, and highly optimizing compilers are available. Fortran is, on the other hand, quite limited in abstraction possibilities, and anything other than array operations are quite cumbersome. It is therefore unsuited as a main language for a larger framework such as PyProp.

At the other end of the scale, dynamic object oriented languages such as Python are very flexible. Object oriented techniques have previously been argued as a very good method for reuse of code[13, 46]. The dynamic nature of Python, and the way it is possible to integrate code from other languages, makes Python an excellent language for writing high level scientific codes[41].

The main argument against Python in scientific computing is performance. Depending on the type of application, computationally intensive programs written in Python are typically 10 to 300 times slower than equivalent programs written in C++ or Fortran[71]. However, for larger programs, the performance is usually confined to a small number of performance sensitive routines. Python makes it easy to integrate other languages, and specifically C/C++[1] or Fortran (using f2py). The performance sensitive routines can then be rewritten in a compiled language to yield the desired performance, while the main body of performance insensitive code is written in a highly expressive language such as Python.

In PyProp, Python is used extensively to deal with configuration files and high level steering tasks, while the numerical calculations are performed mostly in C++, with the exception of a few kernels written in Fortran. Using the interactive Python shell `IPython`[36], the user gets access to a command completion, command history and an interactive debugger, etc. Combined with the plotting library `matplotlib`[34], the user can use PyProp to rapidly explore the results from quantum mechanical calculations to look for interesting results.

2.2.1 C++ Classes

Implementing the concepts introduced in section 2.1 has been done by implementing the `Wavefunction`, `Representation`, and `DistributedModel` as C++ classes templated on the rank of the problem. Knowing the rank at compile-time allows the compiler to perform a number of optimizations, reducing the overhead of writing rank independent algorithms. Furthermore, templating on rank allows PyProp to use the excellent `Blitz++` library[73], which further simplifies the formulation of efficient rank independent algorithms.

2.2.2 Python Interface

All PyProp C++ classes are exposed to Python with the help of the `boost::python`[1] library. Using `pyste` to generate the `boost::python` wrapper, each class is exposed to Python using only a few lines of code. An issue here, is that as C++ templates are compile time structures, it is not possible to expose a templated class directly to Python. Rather, only a specific instance of the templated class may be exposed to Python. Ideally, a scheme where Python instantiates the compiler and compiles the required template instance when needed could be devised. This is similar to the approach the weave project[75] uses to speed up computational kernels. However, as discussed in section 2.2.3, one of the major platforms for which PyProp is available, does not support dynamic libraries. All compiled code must therefore be available before the program starts, which is incompatible with the scheme above. A simpler but more limiting scheme is therefore used, where the rank-templated classes are instatiated for ranks 1 through 4, and compiled into Python extension modules. Most of the C++ classes have Python-native wrappers supplying high level functionality in order to simplify access to the computational routines.

When using PyProp, the user creates an extension module containing the custom potentials and other compiled code necessary for the analysis. The user extension module links to the PyProp modules in order to get access to the required core routines. Most of the user code will usually be in the form of Python code gluing together the necessary routines from the PyProp module.

2.2.3 Dynamic Libraries

In addition to running on "normal" platforms such as GNU/Linux workstations, PyProp must be portable to the high performance computing (HPC) platforms available. In our case, the interesting HPC-platforms are mostly GNU/Linux clusters, with one notable exception: the Cray XT-4 computer at the University of Bergen, known as Hexagon.

The Cray XT-4 platform is built on a custom Linux kernel, optimized for HPC needs. Among the features stripped from the stock Linux kernel is support for dynamic loading of libraries at runtime. This has interesting implications for using Python extension modules, as these are most commonly loaded dynamically at runtime. As PyProp depends on 3rd party extension modules like NumPy, matplotlib and pypar, these modules must also be compiled into one big executable along with the core Python library and the problem specific code.

2.3 Operators

The Hamiltonian in the TDSE can always be partitioned into sum of operators, representing different parts of the system,

$$H = \sum_i H_i. \tag{2.1}$$

As all physically significant quantum mechanical operators are Hermitian (although their discretized representation may not necessarily be), it can safely be assumed that all reasonable operators will have a diagonal representation, i.e. the matrix representation of the operators are far from defect. Specifically, most operators are linear operators of the spatial position $\mathbf{x}$, or the momentum $\mathbf{p} = -\mathrm{i}\hbar\nabla$.

A general discretized linear operator, O, can be multiplied to a discretized wavefunction by

$$\psi'_{\mathbf{i}} = \sum_{i'_0}\sum_{i'_1}\cdots\sum_{i'_N} O_{\mathbf{i},\mathbf{i}'}\psi_{\mathbf{i}'}. \tag{2.2}$$

Here, $\mathbf{i}$ and $\mathbf{i}'$ are vector indices representing an index in an N-dimensional array. If $O_{\mathbf{i},\mathbf{i}'} = 0$ for $\mathbf{i} \neq \mathbf{i}'$, i.e. it does not mix elements in the wavefunction, the operator is said to be a diagonal operator, or a *potential*.

Depending on the kind of propagation scheme, different representations of the operators will be required. For split-step based propagators (discussed in section 3.2), each operator is propagated separately. Most often this is easily performed by having the operator in diagonal form, where the operator exponential can be easily calculated. For the remaining propagators described in chapter 3, only the action of the operator on a wavefunction is required, leaving a greater freedom of choice in the representation of the operators.

2.3.1 Potential evaluation

The goal is to enable users to express operators in an intuitive fashion. Most user supplied operators are potentials. PyProp therefore has a special framework for setting up and evaluating potentials. On the high level, any class implementing a simple Potential-interface supporting can be used as an operator.

For diagonal operators, the user need only create a class implementing a `GetPotentialValue(...)` method mapping a grid coordinate to a potential value. The `PotentialEvaluator` class has a `PotentialClass` template parameter, and uses this to evaluate the potential at the grid points given by the representation, as shown in Listing 2.1. As a result of the template metaprogramming trick of making the potential available to the PotentialEvaluator at compile time, the compiler is able to optimize the call to `GetPotentialValue` inside the loop. The alternative would be to pass the potential as an overloaded function, but then each call to `GetPotentialValue` would have to be looked up at run time, making optimizations impossible.

2.3.2 Tensor Potentials

Some propagation schemes do not require the application of the exponential of an operator, and only the application of the operator on the wavefunction is required. For

Listing 2.1: Excerpt of the PotentialEvaluator class

```
template<int Rank, class PotentialClass>
class PotentialEvaluator
{
public:
  void Apply(typename Wavefunction<Rank>::Ptr psi)
  {
    for (/* iterate over all grid points */)
    {
      Vector pos = getPos(psi->GetRepresentation(), idx);
      cplx V = Pot.GetPotentialValue(pos);
      psi->GetData()(idx) *= exp(- i*dt*V);
    }
  }

private:
  PotentialClass Pot;
};
```

such cases, it might not always be optimal to represent the operators in diagonal form, but rather find an "intermediate" representation, where all operators have more or less the same sparsity structure. In the B-spline representation (see section 4.2.4), for example, both $\hat{\mathbf{x}}$ and $\hat{\mathbf{p}}$ operators are banded, making it natural to represent both kinetic and potential energy in the B-spline basis. In order to create a framework for expanding operators in a product of bases, we will assume that the operators are a combination of grid functions and derivatives, and can be written on the form

$$H_i = g(t) f^{(0)}(x_0, x_1, \cdots, x_N) \frac{\partial^{p_1}}{\partial {x_1}^{p_1}} \frac{\partial^{p_0}}{\partial {x_0}^{p_0}} \cdots \frac{\partial^{p_N}}{\partial {x_N}^{p_N}}. \tag{2.3}$$

Here, the potential values in the initial potential $f^{(0)}$ can be evaluated using the method described in the previous section. The time dependence is quite limited in this form of operators, but well suited for interactions between atomic systems and laser light.

This operator is to be represented in a Kronecker product of one-dimensional bases,

$$B(x_0, x_1, \cdots, x_N) = B^0(x_0) \otimes B^1(x_1) \otimes \cdots \otimes B^N(x_N). \tag{2.4}$$

Here, $B^r(x)$ are the basis functions $\{B^r_j(x)\}$ defining the desired basis for the rth rank.

Finding the matrix elements of H_i in the basis B, can be achieved by expanding f in the basis functions of each rank, r, given the differentiated basis functions $\{\frac{\partial^{p_r} B^r_j(x)}{\partial x^{p_r}}\}$ are known,

$$f^{(1)}_{j',j}(x_0, \cdots, x_{r-1}, x_{r+1}, \cdots x_N) = \int B_{j'}(x_r) f^{(0)}(\mathbf{x}) \frac{\partial^p B_j(x_r)}{\partial {x_r}^p} \mathrm{dx_r}. \tag{2.5}$$

By successively performing the expansion in each rank, the operator will be represented completely in B, known in PyProp as a *Tensor Potential*,

$$\widehat{H}_i = \{f^{(N)}_{\mathbf{i},\mathbf{i}'}\}. \tag{2.6}$$

Listing 2.2: Example of tensor potential application for a rank 3 problem, `op` is the tensor potential data represented in a rank 3 array. `indexPairs` is a list of index pairs for each rank. `src` and `dst` are the source and destination wavefunction arrays respectively.

```
def TensorPotentialMultiply(op, indexPairs, src, dst):
  for i0, (r0,c0) in enumerate(indexPairs[0]):
    for i1, (r1,c1) in enumerate(indexPairs[1]):
      for i2, (r2,c2) in enumerate(indexPairs[2):
        dst[r0,r1,r2] += op[i0,i1,i2] * src[c0,c1,c2]
```

Here, $\mathbf{i} = (i_0, i_1, \cdots, i_N)$ and $\mathbf{i}'$ are vector indices into the N-dimensional wavefunction array.

This formalism can easily be adapted to exploit the sparsity structures in each rank. In the B-spline basis, for example, operators always form a $2k-1$ banded matrix, and the dipole laser field, $f = r\cos\theta$, forms a bi-diagonal matrix in the spherical harmonic representation. Exploiting sparsity is essential in order to have a well performing code, as the difference in size between the full and the sparse matrix can easily become very large (e.g. a factor 500 for a reasonable rank 2 B-spline problem). Here, we will assume that the sparsity structures are known in advance, and that the sparsity structures are independent in each rank.

The integrator for each rank is supplied with the index pairs $\{i_r, i'_r\}$ that give non-zero matrix elements, and calculates the matrix elements for those indices. Application of the tensor potential can be done in N nested loops iterating over the index pairs for each rank, as shown in Listing 2.2. For performance reasons, it is desirable to optimize each of the loops according to the sparsity structure. E.g., if the innermost rank (rank 2 in the above example) is banded, it would be desirable to replace the innermost loop with a call to the BLAS function `zgbmv`.

In order to support a wide variety of sparsity structures without sacrificing performance, a tensor potential multiplication generator is included in PyProp. The generator is given a list of sparsity structures (dense, banded, Hermitian, etc.), and generates a Fortran function capable of performing a tensor potential multiplication with a wavefunction, as described in Listing 2.2. The generator is written in Python, and consists of a recursively chained list of code generators for each rank. Each code generator decides on the best way to iterate over the index pairs according to whether it is the innermost rank, etc.

Chapter 3

Propagation Schemes

Assuming for a moment that the TDSE can be successful discretized (Which we will return to in chapter 4), the TDSE can be written in matrix form,

$$-\mathrm{i}\mathbf{S}\dot{\mathbf{c}}(t) = \mathbf{H}(t)\mathbf{c}(t), \tag{3.1}$$

where $\mathbf{c}$ is the discretized wavefunction, $\mathbf{H}(t)$ is the Hamiltonian represented in the corresponding basis, and $\mathbf{S}$ is the overlap matrix originating from non-orthogonal basis functions such as B-splines (section 4.2.4). For orthogonal bases, the overlap matrix is the identity matrix, $\mathbf{S} = \mathbf{I}$, simplifying the system considerably.

Solving equation 3.1 exactly is only possible for special cases of $H(t)$. In general, numerical approximation schemes must be used. A vast literature is dedicated to this canonical problem, see for example Iserles and Nørsett[35], Zanna[76] and others[28, p.122]. Here, we will only consider schemes assuming the time step h is short enough for the Hamiltonian to be considered time independent on $[t, t+h]$. This corresponds to ignoring the error term in the following expression,

$$\mathbf{c}(t+h) = \exp\left[-\mathrm{i}\mathbf{S}^{-1}\mathbf{H}(t)h\right]\mathbf{c}(t) + \mathcal{O}(h^2). \tag{3.2}$$

Propagating the TDSE is then reduced to repeatedly applying the matrix exponential of the Hamiltonian to the wavefunction. Moler and van Loan[47] have written a review of methods for calculating the matrix exponential. Some of these methods will be described here, tailored to solving the TDSE. For a more mathematical analysis of most of the methods described here, see [42]

For simplified notation, let $\phi(h) = \exp\left[-\mathrm{i}\mathbf{S}^{-1}\mathbf{H}(t)h\right]$ be the propagation matrix, advancing the wavefunction a timestep h.

3.1 Preserving Structure

In the following, we will assume that the Hamiltonian is Hermitian, $\mathbf{H} = \mathbf{H}^*$, and the overlap matrix is Hermitian and positive definite. This is not a severe restriction, as the observable physical quantities are assumed to correspond to Hermitian operators. In this case, by using the adjoint of the propagation matrix, $\phi^*(h) = \exp\left[\mathrm{i}\mathbf{H}(t)\mathbf{S}^{-1}h\right]$, it can be shown that the propagation operator $\phi(h)$ is $\mathbf{S}$-unitary,

$$\phi^*(h)\mathbf{S}\phi(h) = \mathbf{S}. \tag{3.3}$$

The unitarity of the propagation operator leads to a conservation of probability,

$$P(t+h) = \mathbf{c}^*(t+h)\mathbf{S}\mathbf{c}(t+h) = \mathbf{c}^*(t)\phi^*(h)\mathbf{S}\phi(h)\mathbf{c}(t) = \mathbf{c}^*(t)\mathbf{S}\mathbf{c}(t) = P(t) \tag{3.4}$$

Another important physical property is the expectation value of energy,

$$\langle E(t)\rangle = \mathbf{c}^*(t)\mathbf{H}(t)\mathbf{c}(t). \tag{3.5}$$

If $\mathbf{H}$ is not explicitly time dependent, $\langle E(t)\rangle = \langle E\rangle$ will be constant throughout the propagation. This can readily be seen from expanding $\mathbf{c}(t)$ in the eigenstates of $\mathbf{H}$.

Having numerical propagation schemes with these properties built in can have tremendous benefits on the efficiency of the schemes. The total probability of the system is a physically relevant quantity, and it does not make sense to have a total probability different from 1. Preserving probability in the calculations can therefore help get more qualitatively correct results. E.g. if the goal is to calculate the probability of ionization from a system, the ionization can be calculated from the probability of remaining in a bound state, $P_I = 1 - P_b$. The propagation scheme need not give accurate results for the population or phase on individual states, but can still give a good estimate for the total population in the bound states. Using a probability conserving method, we will automatically get a good estimate for the ionization probability from the bound probability.

Preserving the intrinsic physical properties of the PDE is closely related to the field of Geometric Numerical Integration, reviewed in detail in [28].

3.2 Split Step

Assume the Hamiltonian can be split into a sum of p sub-hamiltonians $\mathbf{H}_i$ (equation 2.1), where the exponential of the sub-hamiltonians, $\exp[-\mathrm{i}\mathbf{S}^{-1}\mathbf{H}_i(t)h]$, be can easily calculated. The idea of the split step propagator is to approximate the full operator exponential for the Hamiltonian by a series of operator exponentials for the sub-hamiltonians,

$$\mathbf{c}(t+h) = \phi(h)\mathbf{c}(t) = \phi_0(h)\cdots\phi_p(h)\mathbf{c}(t) + \mathcal{O}(h^2). \tag{3.6}$$

Here, $\phi_i(h)$ is the propagator for the ith sub-hamiltonian. The error in the expression above is due to the non-commutivity between the different sub-hamiltonians (if the sub-hamiltonians commute, the error vanishes), and can be reduced by a more complicated composition of the sub-hamiltonians. It has been shown that compositions yielding accuracy of any even order k can be obtained, see [45] and references therein.

Despite the work on high order splitting schemes, the most common split step propagator is using the third-order Strang composition scheme[70], where a symmetric product of the terms in equation 3.6 is formed,

$$\phi(h) = \phi_0(h/2)\cdots\phi_{p-1}(h/2)\phi_p(h)\phi_{p-1}(h/2)\cdots\phi_0(h/2) + \mathcal{O}(h^3). \tag{3.7}$$

Combined with spherical harmonics (section 4.2.3), and the fast Fourier transform, this method was made popular for the TDSE by Feit, Hermann and Fleck in the 1980s[20, 31].

3.2.1 Sub-Propagators

In PyProp, the built in split step propagator, `CombinedPropagator`, is based on a specific partitioning of the Hamiltonian in order to simplify user interaction. Every rank of the wavefunction is assigned a sub-propagator, responsible for propagating that rank one time step. The sub-propagators usually propagate the kinetic energy operator for that rank, but are in general free to solve any propagator as long as they implement the sub-propagator interface. The B-spline sub-propagator, for example, can include a one-dimensional time independent potential in its propagator.

The most prominent part of the sub-propagator interface is the `AdvanceStep` method which, in addition to applying the kinetic energy operator, is responsible for leaving the wavefunction in a representation where the potentials can be propagated, i.e. a grid representation. As an example, the spherical harmonic sub-propagator propagates the angular kinetic energy term (which is diagonal in the spherical harmonic representation), and transforms the given rank to the angular grid representation. Similarly, the B-spline representation transforms the wavefunction to the B-spline grid representation after propagating the kinetic energy term. The sub-propagator interface also supplies an `AdvanceStepConjugate` method, transforming the given rank *back* from the grid representation, and then propagates the kinetic energy operator.

3.2.2 Potentials

In addition to the sub-propagators, the user can specify a number of additional potentials which will be exponentiated and propagated. Again, these potentials do not necessarily have to be potentials in the sense defined in section 2.3. Any object implementing the potential interface described in Listing 3.1 can be used as a potential. The potential interface allows propagation, i.e. applying the exponential of the operator, as well as multiplying the wavefunction by the operator.

Listing 3.1: Excerpt of the PotentialEvaluator interface

```
class Potential:
  def AdvanceStep(psi, t, dt):
    # Apply exp(- i dt op) to psi, where op is
    # the operator modelled by this potential,
    # t is the current time and dt is the time step
    ...

  def MultiplyPotential(srcPsi, dstPsi, t, dt):
    # Right multiply srcPsi with the operator,
    # and put the result in dstPsi
    ...
```

3.2.3 CombinedPropagator

`CombinedPropagator` uses the symmetric splitting scheme (equation 3.7). This makes it easy to support transforming sub-propagators, as well as automatic paralleliza-

tion. Propagating one time step is achieved by first propagating a half time step with the sub-propagators, then propagating a full time step with the user supplied potentials before finally propagating a half timestep with the conjugate sub-propagators.

To support parallelization, each sub-propagator can signal whether it can be applied while the given rank is distributed. If it can not be propagated in parallel, the `CombinedPropagator` redistributes the wavefunction as described in Paper **I** before propagating the sub-propagator.

Split step propagation was initially the only propagation method available in PyProp, and the simulations performed in Papers **II**, **III**, **IV**, **V** and **VIII** were all performed with a version of the split step propagator.

3.2.4 Numerical Issues

Despite the $\mathcal{O}(h^3)$ local error predicted for this method, the exact form of the error is dependent on the commutator between the split operators. In particular, splitting differential operators from potentials with singularities, e.g. the Coulomb potential, can lead to errors that are difficult to overcome. This problem is discussed in more detail in Paper **IV**, and is also noted in [42].

3.3 Krylov Subspace Exponentiation

An alternative to splitting the full propagation operator is to project the wavefunction into a smaller basis, perform the matrix exponential there, and then transform back. The matrix exponential can be formally defined from the Taylor series,

$$\exp\left[-\mathrm{i}\mathbf{S}^{-1}\mathbf{H}h\right]\mathbf{c} = \sum_k \frac{(-\mathrm{i}h)^k}{k!}(\mathbf{S}^{-1}\mathbf{H})^k\mathbf{c}. \tag{3.8}$$

One possible propagation scheme is to truncate the series and apply it directly. This method is known in the literature as a Taylor propagator[18], but has the undesirable property of not being explicitly unitary. For this reason, we will not consider the Taylor propagator further here.

The idea in the Krylov subspace exponentiation is to truncate the Taylor series and solve the matrix exponential in a subspace of the full solution spanned by the m first terms. This corresponds to the Krylov space of order m, $\mathcal{K}_m = \mathrm{span}[\mathbf{c}, \mathbf{S}^{-1}\mathbf{H}\mathbf{c}, \cdots, (\mathbf{S}^{-1}\mathbf{H})^{m-1}\mathbf{c}]$. An $\mathbf{S}$-orthogonal basis $\mathbf{Q}_m$ spanning $\mathcal{K}_m$ can be found using the Arnoldi method[4] or, if $\mathbf{H}$ is Hermitian, Lanczos iterations[40]. The matrix exponential can then be performed in the $\mathbf{Q}_m$ basis,

$$\exp\left[-\mathrm{i}\mathbf{S}^{-1}\mathbf{H}h\right]\mathbf{c} \approx \mathbf{Q}_m \exp\left[-\mathrm{i}h\mathbf{Q}_m^*\mathbf{H}\mathbf{Q}_m\right]\mathbf{Q}_m^*\mathbf{S}\mathbf{c} = \mathbf{Q}_m \exp\left[-\mathrm{i}\mathbf{H}_m h\right]\mathbf{e}_m. \tag{3.9}$$

Here, $\mathbf{e}_m = (1,0,0,\cdots)$ is the unit vector of length m. The matrix exponential of the projected Hamiltonian $\mathbf{H}_m$, can be calculated using one of the methods described in [47]. The choice of method is not important, as the propagator usually spends much more time performing the Arnoldi or Lanczos iterations compared to exponentiation $\mathbf{H}_m$.

This method was used in quantum simulations by Nauts and Wyatt[49], and later extended for non-orthogonal bases by Park and Light[54]. Hochbruck[33] analyzed the convergence of the method, showing that approximating the exponential directly by Krylov subspace exponentiation converges faster than solving the corresponding (unpreconditioned) linear system of equations. A freely available software package, Expokit[66], enables the user to calculate matrix exponential using the method described above.

Although the Krylov propagator is not unitary, it is explicitly probability conserving. As $\mathbf{H}$ is Hermitian, $\mathbf{H}_m$ will also be Hermitian, and it follows that the exact exponentiation of $\mathbf{H}_m$ is unitary, conserving probability in the Krylov basis. The Arnoldi basis is orthonormal and the initial condition is fully represented in that basis. Correspondingly, the final wavefunction will also be fully represented in the Arnoldi basis, having the same norm as the initial wavefunction. This feature makes the Krylov propagator stand out among the explicit propagators.

3.3.1 Implementation in PyProp

Implementing the Arnoldi or Lanczos iterations can be performed in matrix free mode, i.e. the iterator does not need to know the Hamiltonian matrix directly. The only requirement is to be able to perform the matrix vector product $\tilde{\mathbf{c}} \leftarrow \mathbf{Hc}$, i.e. left multiply the wavefunction by the Hamiltonian.

Using this matrix free design, the Krylov propagator (known in PyProp as the parallel Arnoldi method propagator, `pAMP`) can be implemented on top of another propagator, as long as the base propagator implements a `MultiplyHamiltonian` method, performing the matrix vector product.

The `CombinedPropagator` from section 3.2.3 can be modified to support matrix vector multiplication by adding a `MultiplyHamiltonian` method to the sub-propagators. `MultiplyHamiltonian` is similar to `AdvanceStep`, transforming the wavefunction to a suitable representation. Instead of multiplying the exponentiated sub-hamiltonians, `MultiplyHamiltonian` performs matrix vector products between the sub-hamiltonians and the wavefunction, adding the results to an output-wavefunction.

Although the `CombinedPropagator` can be used to perform matrix vector multiplications, and it is very easy to switch between the split step and Krylov propagators, some overhead is induced by transforming the wavefunction to the desired representations instead of representing all operators in one basis. The `TensorPotential` discussed in section 2.3.2 is intended to resolve this issue by being able to expand operators in a basis at the start of the propagation and performing matrix vector multiplications without transformations.

3.3.2 Parallelization

We use the standard way of parallelizing Krylov subspace methods, assuming a parallel matrix vector multiplication. Calculation and exponentiation of $\mathbf{H}_m$ is performed locally on every processor, and the only parallel consideration the Arnold or Lanczos iterations must do, is to perform a global sum across processors when orthogonalizing $\mathbf{Q}_m$. This is not performed in expokit, and is the main reason why a custom Krylov propagator has been implemented in PyProp.

3.4 Cayley Form Propagator

The Cayley form propagator is perhaps the most well-known propagator for the TDSE and is known in quantum mechanics under many names, e.g. Cayley form propagator[74], or Crank-Nicholson propagator[14]. In the numerical ODEs literature, it is usually called implicit midpoint rule[42]. The method can be derived by forming the average of the backward and forward Euler propagators, and can also be seen as a $p = q = 1$ Padé approximation to equation 3.2 [25].

$$\mathbf{c}(t+h) = \left(\mathbf{S} + \frac{\mathrm{i}h}{2}\mathbf{H}\right)^{-1} \left(\mathbf{S} - \frac{\mathrm{i}h}{2}\mathbf{H}\right) \mathbf{c}(t) + \mathcal{O}(h^3). \tag{3.10}$$

Higher order versions of this propagator have been known for a long time[15], but the Cayley form propagator has remained popular due to its simplicity and numerical features such as unitarity and time reversibility.

Applying the Cayley form is performed in two steps. First, apply the rightmost term of equation 3.10. This is simply a matrix vector multiplication similar to the ones performed in the previous section. The second part of the Cayley form, on the other hand, involves solving a linear system of equations involving the full Hamiltonian. This is the main computational difficulty in using the Cayley form, as the system can easily become *very* large.

For small problems, Cayley propagator equation can be solved by *direct methods*, as provided by LAPACK[3], SuperLU[17] and others. This works well for rank 1 problems, as some one-dimensional discretizations (B-splines and finite differences) yield banded matrices with bands close to the main diagonal. These systems can be efficiently solved by direct methods, exploiting the bandedness in the matrices. For systems of rank > 1, the bands of the discretizations are no longer close to the diagonal, and substantial fill-in would occur. As the computational work of solving a linear system scales cubically with the number of unknowns, the interesting systems quickly become too large for direct methods, and *iterative methods* must be used instead. Furthermore, the direct methods require direct access to the Hamiltonian matrix, which does not correspond well with the black box design of propagators and potentials in PyProp.

In PyProp, the Cayley form propagator is used in two different ways, as a partial propagator in the `CombinedPropagator`, or as a complete propagator for the entire system using iterative methods to solve the linear system of equations. The Cayley propagator is used in the partial propagator for B-splines and finite differences, as the bandedness of the corresponding matrices makes solving the linear system much faster than than calculating the full matrix exponential.

3.4.1 Solving the Linear Equations using GMRES

Writing $\mathbf{A} = \mathbf{S} - \frac{\mathrm{i}h}{2}\mathbf{H}$ and $\mathbf{b} = (\mathbf{S} - \frac{\mathrm{i}h}{2}\mathbf{H})\mathbf{c}(t)$, the implicit part of the Cayley form of the propagator can be cast in a more common linear algebra notation,

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{3.11}$$

Here, $\mathbf{x} = \mathbf{c}(t+h)$ is the desired wavefunction.

When the Cayley propagator is used as a partial propagator the matrix dimension of the above equation is usually reasonably small, and the equation can therefore be solved using direct solvers. On the other hand, when used as a complete propagator, we can not expect direct methods to be efficient. Furthermore, $\mathbf{A}$ is not Hermitian, limiting the iterative solver alternatives. PyProp has an implementation of the Generalized Minimal Residual (GMRES) method[59] built in. The GMRES method is similar to the Krylov exponentiation described in section 3.3 as it projects the system into a Krylov subspace, leading to a lower dimensional problem. In the case of GMRES, the solution is chosen such as to create the minimal error (residual) of any possible solution in the space spanned by the Krylov basis.

When discretizing the Hamiltonian, the range of eigenvalues of the resulting matrices is dependent on the discretization parameters. For many of the problems we are interested in here, the lowest eigenvalue (in absolute value) decreases as the size of the grid on which the TDSE is solved increases. Furthermore, the largest eigenvalues increases with increasing grid resolution. This is examplified by considering the Hamiltonian of a free particle, i.e. a Hamiltonian with only the Laplacian. When discretizing with a Fourier basis as described in section 4.2.1, the eigenvalues are quadratically spaced between $E_0 = 0$ and $E_N = 2(\pi N/L)^2$. Although the Hamiltonians considered here have a more complicated structure, many of the systems arising in atomic physics have a continuum component where the Laplacian dominates. Iterative methods, such as GMRES, are known to converge slowly for systems with these properties, and generally work best for systems with clustered eigenvalues. However, the situation can be improved through preconditioning. Choosing a preconditioning matrix $\mathbf{M}$, equation 3.11 can be transformed,

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b} = \tilde{\mathbf{b}}. \tag{3.12}$$

$\mathbf{M}$ should be chosen such as to make $\mathbf{M}^{-1}\mathbf{A}$ close to the identity matrix while at the same time ensuring that solving $\mathbf{M}^{-1}\mathbf{b}$ is easily performed. Preconditioning can then substantially decrease the time spent solving the linear system. It is quite non-trivial to find an optimal preconditioner, and there are no reasons to believe a single preconditioner will be optimal for most cases. In PyProp, the user is therefore expected to supply a suitable preconditioner to the GMRES solver, implementing a `Preconditioner` interface.

An approach similar to this was used in [27], where the Cayley propagator was used in conjunction with iterative solvers to propagate the Gross-Pitaevskii equations.

3.5 Finding Eigenstates

In quantum mechanics, the eigenvalues of the Hamiltonian have a special meaning. The eigenstates are the stationary states of the system, and the corresponding eigenvalues are the possible energies the system can have.

$$E_n\mathbf{S}\mathbf{v}_n = \mathbf{H}\mathbf{v}_n \tag{3.13}$$

For time dependent Hamiltonians, eigenstates does not make sense in an ordinary manner[64]. It is rather the eigenstates of the time independent Hamiltonian, i.e. the system after or before the interaction has taken place, that is of special interest.

Eigenstates and values are often used for analysis, as energy is often a measurable and interesting quantity in physical experiments. In Paper **V**, for example, the analysis involves projecting the wavefunction on the vibrational eigenstates of the system. In Paper **VII**, the energy distribution of the ionized electrons are found by a projection is on the ionizing eigenstates of a simpler system.

Eigenstates can also be used for propagation, as they diagonalize the Hamiltonian, and makes propagating with equation 3.2 trivial. Some of the sub-propagators discussed in section 3.2.1 uses this fact to propagate one sub-hamiltonians.

In this section we will focus on some methods for calculating eigenstates and eigenvalues used in PyProp. For the same reasons as discussed for direct solution of linear systems in the previous section, direct diagonalization will not be considered here.

3.5.1 Imaginary Time Propagation

One way of obtaining the lowest energy state is imaginary time propagation [58]. Consider the wavefunction expanded in eigenstates, $\mathbf{c}(t) = \sum_i a_i(t)\mathbf{v}_i$. The solution of the TDSE for a time independent Hamiltonian is in this case

$$a_i(t) = a_i(0)e^{-\mathrm{i}E_i t} \tag{3.14}$$

The idea of imaginary time propagation, is to transform t,

$$t \to -\mathrm{i}\tau. \tag{3.15}$$

This is equivalent of turning the Schrödinger equation into the heat equation where the time evolution of each eigenstate amplitudes turn into

$$a_i(\tau) = a_i(0)e^{-E_i\tau}. \tag{3.16}$$

As all the eigenvalues of the Hamiltonian are real, the corresponding amplitudes will be damped exponentially with exponential factors corresponding to the eigenvalues. I.e. the highest eigenvalues will be damped the most. By propagating τ, and keeping the wavefunction normalized to unity, all components of the wavefunction with eigenvalues larger than the ground state, will be quenched.

The groundstate can be found by starting in a random state, and propagate until some convergence criterion (typically change in expectation value of energy) is met. Excited states can also be found using this technique by restarting the propagation and making sure the new state is orthogonal to the already converged states at all times. Because spurious populations on the already converged states may be introduced by the propagator, it becomes increasingly hard to find higher excited states. Only a few of the lowest excited states can therefore be found with this method.

3.5.2 Implicitly Restarted Arnoldi Method

If only a few eigenpairs are desired, it is possible to use the Krylov subspace methods to approximate the eigenvalues and eigenvectors of the system. This was the initial application of the Lanczos and Arnoldi iterations[4, 40]. The idea is similar to the Krylov subspace exponentiation in that the Hamiltonian is projected in an **S**-orthogonal

basis $\mathbf{Q}_m$, and the Ritz-values, i.e. the eigenvalues of the projected Hamiltonian is used as estimates for some of the eigenvalues of the full Hamiltonian,

$$\mathbf{H} = \mathbf{V}\mathbf{E}\mathbf{V}^{-1} \approx \mathbf{Q}_m\mathbf{V}_m\mathbf{E}_m\mathbf{V}_m^{-1}\mathbf{Q}_m^*. \tag{3.17}$$

$\mathbf{V_m}$ is the eigenbasis of the projected Hamiltonian. Unlike the Krylov subspace exponentiation, the start vector for the Krylov space is now arbitrary, and is usually chosen randomly.

The eigenvalues corresponding to the most bound states, i.e. the eigenvalues in the lower end of the spectrum, are usually of interest. Arnoldi and Lanczos iterations, however, are best at locating the eigenvalues of largest magnitude. One way of directing the iterations to the desired part of the spectrum is to make sure the start vector is orthogonal to the eigenvectors in the undesired part of the spectrum. The eigenvectors are not known at the start of the iterations, but by stopping the iterations after the Krylov space reaches a given size, we can use the least desired Ritz values to improve the start vector and restart the iterations. The most widely applied restarting technique today is the *implicitly restarted Arnoldi method* used in ARPACK[69].

3.5.3 Inverse Iterations

Noticing that the methods in the section above works best for the largest eigenvales, a natural alternative to using the Krylov space of $\mathbf{H}$ is to use the reciprocal of the shifted Hamiltonian $\mathbf{B} = (\mathbf{H} - \sigma\mathbf{S})^{-1}$. In this case, the largest eigenvalues of $\mathbf{B}$, will correspond to the eigenvalues of $\mathbf{H}$ closest to σ,

$$\lambda_n = \frac{1}{E_n - \sigma}. \tag{3.18}$$

The eigenvectors, on the other hand, will remain the same for $\mathbf{H}$ and $\mathbf{B}$.

In forming the Krylov subspace, matrix vector multiplication with $\mathbf{B}$ must be performed repeatedly, i.e. we must solve a linear system for each step in the Arnoldi or Lanczos iterations,

$$(\mathbf{H} - \sigma\mathbf{S})\mathbf{x} = \mathbf{S}\mathbf{b}. \tag{3.19}$$

As argued in section 3.4, this linear system is in general too large to be solved directly, which leads us to using another iterative method for solving the linear system. In PyProp, this problem is currently solved using the same solver as for the Cayley propagator. Both systems of equations can be cast into the form

$$(\sigma_H\mathbf{H} - \sigma_S\mathbf{S})\mathbf{x} = \mathbf{S}\mathbf{b}, \tag{3.20}$$

and be solved with a preconditioned GMRES solver. It should be noted that equation 3.19 is Hermitian, and could be solved with a Hermitian solver like conjugate gradient or similar. However, for the current projects, much more time is spent propagating compared to finding eigenpairs, and the speed and memory requirements of the eigensolver have not been an issue for the systems where inverse iterations have been used.

Chapter 4

Spatial Discretization

Computers can only perform a finite number of operations per second. The only viable option for solving the TDSE using computers is therefore to reduce the infinite-dimensional Hilbert space containing the wavefunction to one of finite, albeit very large, dimension. I.e. we must *discretize* the system. Using a good discretization, the solution of the original, continuous system can be well described from the solution of the discretized system. There is little reason to believe that there is one kind of discretization that will be superior for all systems. A solver aiming for a wide range of systems must therefore support a number of different discretization schemes. In this chapter, we will consider the discretization schemes that are available in PyProp, and discuss some of their strengths and weaknesses.

Discretization is in PyProp controlled by the `Representation` class discussed in section 2.1.3. In almost all cases, the `CombinedRepresentation` is used to combine different degrees of freedom into one representation. The goal of the discretization is being able to represent all important aspects of the wavefunction with as few data points as possible. With this in mind, it is also of great importance to choose the "right" coordinate system before discretizing, as the wavefunction may have some structure in certain coordinate systems, and can thus be represented by fewer data points.

4.1 Coordinate Systems

Choosing a coordinate system is the first step in any discretization scheme, and choosing the optimal coordinate system is not necessarily easy. Some common systems, like atoms, have exact or near spherical symmetry which makes spherical coordinates the obvious choice, as the solution can be expected to be slowly varying in the angular directions. The computational effort can therefore be concentrated on the radial direction. The Hamiltonian of diatomic molecules with one active electron can be exactly separated in the prolate spheroidal coordinates[6], making that the obvious coordinate system. Other systems may similarly be more or less separable in one coordinate system, but it is not only the symmetries of the unperturbed system that dictates coordinate system: the interaction part of the Hamiltonian must also be considered. An example is the interaction between charged particles and linearly polarized laser radiation in the dipole approximation, where a rotational symmetry around the polarization axis of the laser is observed. Choosing a coordinate system which is able to exploit this symmetry, such as spherical or cylindrical coordinates, can reduce the complexity of the problem

by one degree of freedom. For semi-classical collision problems, however, there is no azimuthal symmetry, and the analysis might be sufficiently complicated by a spherical cordinate system to make Cartesian coordinates optimal.

The desired observable quantities must also be taken into consideration when choosing a coordinate system. The analysis can be severely complicated by having the wavefunction represented in the "wrong" coordinate system, especially if differential or integrated quantities in a different coordinate system are desired. For instance, constructing the angular differential ionization probabilities from a large Cartesian coordinate system can be involved enough to warrant propagation in spherical coordinates even if the system otherwise favors a Cartesian coordinate system. Cartesian and spherical coordinate systems are currently in use in PyProp, and an implementation of prolate spheroidal coordinates is under development.

A coordinate system gives a number of independent dimensions and a Jacobian as well as a representation of the Laplacian. In PyProp, the dimensions are described by one-dimensional representations and combined through the `CombinedRepresentation`. The representations know the grid in each dimension, as well as the integration weights necessary to implement inner products between two wavefunctions in the same representation.

4.1.1 Cartesian Coordinates

Cartesian coordinate systems have the simplest possible kind of coordinates. The dimensions are completely independent, the Jacobian is unity and the Laplacian is expandable into equivalent parts for each dimension,

$$\nabla^2 = \sum_i \frac{\partial^2}{\partial {x_i}^2}. \tag{4.1}$$

As each dimension is completely independent, it is natural to assign one rank to each physical dimension.

4.1.2 Spherical Coordinates

Consider a spherical coordinate system. Although the ideas presented here are readily extensible to an arbitrary number of dimensions[61], the focus here will be on the common three-dimensional version,

$$\begin{aligned} x &= r\sin\theta\cos\phi \\ y &= r\sin\theta\sin\phi \\ z &= r\cos. \end{aligned} \tag{4.2}$$

The Jacobian for this coordinate transform is

$$|J| = r^2\sin\theta. \tag{4.3}$$

It is often preferable to use a trick called reduced wavefunction, where the wavefunction is scaled with a radial function before discretizing. The most common scaling function is $f(r) = r$, leading to a new wavefunction,

$$\Psi(\mathbf{r},t) = r\psi(\mathbf{r},t), \tag{4.4}$$

with which the calculations take place. Reducing the wavefunction has several advantages. First, it simplifies the left boundary conditions to $\Psi(r=0)=0$. Second, it removes the r^2 term from the Jacobian. Furthermore, the Laplacian is considerably simplified, removing the first order radial derivative term, turning the spherical Laplacian into

$$\nabla^2 = \frac{\partial^2}{\partial r^2} + \frac{1}{r^2}\left(\frac{\partial^2}{\partial \theta^2} + \cot\theta\frac{\partial}{\partial\theta} + \frac{1}{\sin^2\theta}\frac{\partial^2}{\partial\phi^2}\right) = \frac{\partial^2}{\partial r^2} - \frac{\mathbf{L}^2}{r^2}. \tag{4.5}$$

Here, $\mathbf{L}^2$ is the angular momentum operator. Having the Laplacian on this form, any discretization used for a Cartesian rank can be used as a radial discretization. The discretization of the angular part is discussed further in section 4.2.3.

It is possible to use different scaling functions[39, 43] for the reduced wavefunction. This corresponds to tuning the Laplacian such as to make a certain set of states eigenstates to the radial part of the Laplacian.

4.2 Discretization Schemes

In order to create a flexible discretization system, each rank will be discretized separately. Let $\mathcal{M}_i$ be a map discretizing rank i in the wavefunction. The full discretization is then a tensor product of rank one discretizations,

$$\mathbf{c} = \mathcal{M}_0 \otimes \mathcal{M}_1 \otimes \cdots \otimes \mathcal{M}_N \psi(\mathbf{x}). \tag{4.6}$$

This means that different discretizations can be chosen independently for each rank. In PyProp this is represented by the `CombinedPropagator`'s ability to combine different rank 1 representations. It is therefore sufficient to study the rank 1 representations separately. Most of these representations will be suitable in a Cartesian coordinate system, or as the radial part in a spherical coordinate system.

A general scheme for discretization is to expand the wavefunction in a finite set of basis functions,

$$\psi(x) \approx \sum_{i=0}^{n-1} c_i B_i(x). \tag{4.7}$$

The basis functions $B_i(x)$ should satisfy appropriate boundary conditions and be at least twice differentiable. Let the overlap matrix $\mathbf{S}$ be defined from the integral between two basis functions,

$$S_{i',i} = \int B^*_{i'}(x) B_i(x) \mathrm{d}x. \tag{4.8}$$

The expansion coefficients can be found by solving for the overlap matrix,

$$\mathbf{c} = \mathbf{S}^{-1}\int \mathbf{B}^*(x)\psi(x)\mathrm{d}x = \mathbf{S}^{-1}\begin{pmatrix} \int B^*_0(x)\psi(x)\mathrm{d}x \\ \int B^*_1(x)\psi(x)\mathrm{d}x \\ \cdots \\ \int B^*_{n-1}(x)\psi(x)\mathrm{d}x \end{pmatrix}. \tag{4.9}$$

Here, $\mathbf{B}(x) = (B_0(x), B_1(x), \cdots, B_{n-1}(x))$ is the set of basis functions. In the simplifying case where $\mathbf{S}$ is the identity matrix, the basis is said to be *orthogonal*.

In this chapter we will consider mostly the collocation method where the approximation in equation 4.7 is forced to be exact on a set of collocation points $\{x_i\}$. This leads to two natural representations of the wavefunction, one grid representation and one basis representation. In the grid representation the wavefunction is sampled at the collocation points, while in the basis representation the wavefunction is represented in terms of expansion coefficients for the corresponding basis.

In order to turn this scheme into a linear algebra formulation, $\mathbf{B}(x)$ can be sampled in the collocation points and turned into the basis matrix $\mathbf{B}$,

$$B_{i,j} = B_j(x_i). \tag{4.10}$$

Given a basis, the integrals needed are mostly on the form 4.8. A quadrature being able to accurately evaluate such integrals should therefore be chosen. The quadrature can be written as a diagonal weight matrix $\mathbf{W}$ such that

$$\int f(x)\mathrm{d}x \approx \sum_i W_{i,i} f(x_i) \tag{4.11}$$

approximates an integral on the appropriate domain. Using the weight and the overlap matrices, the sampled wavefunction can be transformed into the basis representation,

$$\mathbf{c} = \mathbf{S}^{-1}\mathbf{B}^*\mathbf{W}\psi, \tag{4.12}$$
$$\psi = \mathbf{B}\mathbf{c}. \tag{4.13}$$

Inner products between two wavefunctions ψ and ψ', can then be calculated in the grid- or basis-representation

$$\left\langle \psi(x) | \psi'(x) \right\rangle \approx \psi^*\mathbf{W}\psi' = \mathbf{c}^*\mathbf{S}\mathbf{c}'. \tag{4.14}$$

In the basis representation, the differential operators can be approximated by differentiating the basis functions,

$$\frac{\partial^k}{\partial x^k}\psi(x) \approx \sum_{i=0}^{n-1} c_i \frac{\partial^k B_i(x)}{\partial x^k} = \sum_{i=0}^{n-1} c_i B_i^{(k)}(x). \tag{4.15}$$

By using the quadrature to project the differentiated basis functions on the basis functions, a differentiation matrix can be formed,

$$\mathbf{D}^k = \mathbf{B}^*\mathbf{W}\mathbf{B}^{(k)}, \tag{4.16}$$

The different discretization schemes can then be seen as a choice of basis functions and corresponding quadrature.

4.2.1 Fourier Representation

For periodic boundary conditions, a natural choice of basis functions are the Fourier functions for a box of size L,

$$B_j = e^{\mathrm{i}2\pi jx/L}, \qquad j = (-n/2, \cdots, 0, \cdots n/2-1). \tag{4.17}$$

For Fourier basis functions, the collocation points are equispaced, $\Delta x = L/n$, and the natural quadrature is trapezoidal integration with constant weights, $w_i = \Delta x$.

The Fourier functions have the desirable property of being eigenstates to the differentiation operators which physically can be associated with a well defined momentum, k,

$$\frac{\partial B_j(x)}{\partial x} = \mathrm{i}2\pi j/L B_j(x) = \mathrm{i}k B_j(x). \tag{4.18}$$

This makes the Fourier basis a good candidate for the split step propagation schemes described in section 3.2[20]. Furthermore, the transformation to and from the Fourier basis can be carried out with the fast Fourier transform (FFT), making it the fastest of the global basis transforms.

The effectiveness of using a Fourier basis is generally determined by the smoothness of the periodic extension of the wavefunction. This can be separated into two conditions: that the wavefunction is smooth in the interior of the interval (x_{min}, x_{max}), and that the periodic extension of the wavefunction does not remove smoothness.

The first condition is dependent on the Hamiltonian, e.g. for a harmonic oscillator in Cartesian coordinates, the exact solutions are Hermite functions. Hermite functions are analytic on the entire real domain and thus smooth. For the Coulomb problem, however, it is well known that in Cartesian coordinates, the ground state wavefunction has a cusp in the origin, limiting the smoothness to C^0. This makes the Fourier basis in Cartesian coordinates unsuitable for calculations requiring high accuracy of the Coulomb ground state.

The second condition can in Cartesian coordinates usually be resolved by increasing the box size. However, when used in a radial coordinate, the wavefunction can not be expected to approach zero smoothly near the left boundary ($x = 0$). The standard trick here[31] is to anti-symmetrically extend the wavefunction to $-x_{max}$, such that $\psi(-x) = -\psi(x)$. As discussed in Paper **IV**, this does not remove all smoothness related problems for the Coulomb potential, as the ground state wavefunction is only C^1 in the origin. Nevertheless, it is a vast improvement over using a periodic grid on $x \in [0, x_{max})$.

4.2.2 Orthogonal Polynomials

Another, although related, option is to use interpolating polynomials as basis functions. We define orthogonal polynomials by choosing a domain on the real axis $\Omega = [x_0, x_1]$, which can be infinite or semi-infinite, as well as a strictly positive weight function $w(x)$. The unique set of normalized orthogonal polynomials of Ω and $w(x)$ are then defined as the set of functions $\{p_j(x)\}$ such that, $p_j(x)$ is a polynomial of degree j and

$$\int_\Omega w(x) p_i(x) p_j(x) \mathrm{d}x = \delta_{i,j}. \tag{4.19}$$

The orthogonal polynomials have many fascinating properties[2, chap. 22], and in particular, it can be shown that the optimal collocation points for an orthogonal polynomial of maximum order $n-1$ are the roots of the orthogonal polynomial of order n.

If orthogonal polynomials are desired on a given interval, it is possible to construct them from the expression above. However, it can often be fruitful to use an already defined set of orthogonal polynomials and perform a change of variables, $x = f(y)$.

This was done in Paper IV, where Chebyshev polynomials were transformed with a change of variables from $\Omega_x = [-1,1]$ to $\Omega_y = [0,\infty)$ in order to be suited to a radial discretization. A similar approach was performed in [10] for $\Omega_y = (-\infty,\infty)$. By using a radial coordinate transform like this, it is possible to position the grid points where they are most needed. For spherical problems, this is usually near the origin. In Paper **IV** we showed how this can be used to get excellent convergence properties for hydrogenic systems.

As is the case with all methods, using orthogonal polynomials are not expected to be the universally best discretization schemes. Except for very special polynomials, the transformation between a grid representation in the collocation points and a orthogonal polynomial representation is very computationally expensive. For a general global orthogonal basis, the transformation between basis and grid representation (equation 4.12) will scale as $\mathcal{O}(n^2)$ operations for a one-dimensional wavefunction. In contrast, using a Fourier or Chebyshev basis, this computational cost can be reduced to $\mathcal{O}(n\log n)$ operations.

There is also a drawback of including $x \to \infty$ in the computational domain. The drawback here lies in that the continuum states of atomic and molecular systems, i.e. the states corresponding to one of the particles becoming dissociated with the system corresponds to states that do not decay exponentially as $x \to \infty$. Such states are not L_2 integrable, and can not be properly represented by discretized functions on that domain. Methods like R-matrix theory [12] deal with this problem in an elegant way, but the most straight-forward solution is to terminate the grid at some x_{max} such that $\psi(x > x_{max}) \approx 0$ for the entire propagation. Setting the boundary condition $\psi(x_{max}) = 0$ can be seen as a way of discretizing the continuum by only accepting continuum functions that have a node in $x = x_{max}$. As the wavefunctions are nonzero only for $x < x_{max}$, the continuum functions are only needed inside that region.

Fourier basis functions can also be seen in light of orthogonal polynomials. Although not regular polynomials, Fourier functions are trigonometric polynomials, $\exp(ikx) = \exp^k(ix)$, which are orthogonal on $x \in [0, 2\pi]$ with weight function $w(x) = 1$ and equidistant collocation points.

4.2.3 Spherical Harmonics

The angular part of spherical coordinates requires some special attention. The reason for choosing spherical coordinates is usually that the problems are nearly spherically symmetric. In practice this means that there is little motion in the angular coordinates, which should be exploited. Generally, when a the state of a system is spanned by a small number of states, it can be preferable to use an eigenstate basis. The eigenstates for the angular momentum operator in eqn 4.5 are the spherical harmonics,

$$Y_l^m(\theta,\phi) = \delta_m \sqrt{\frac{2l+1}{2\pi}\frac{(l-m)!}{(l+m)!}} P_l^{|m|}(\theta) e^{\mathrm{i}m\phi}. \tag{4.20}$$

Here $P_l^{|m|}(\theta)$ are the associated Legendre polynomials, and δ_m is the Condon Shortley phase. These eigenstates are currently used as basis functions for all calculations in PyProp where spherical coordinates are used.

From the definition of the spherical harmonics it would be natural to write the angular part of the wavefunction as two ranks; a tensor product between θ and ϕ. However, it is not possible to write the wavefunction as a tensor product of l and m, as the associated Legendre polynomials are zero for $|m| > l$. For this reason, PyProp deals with the angular part as a single rank, discretizing the solid angle $\Omega = (\theta, \phi)$ instead of the two angles separately.

It has been known since antiquity that it is not possible to create a perfect distribution of points on a sphere unless the number of points match the vertices on one of the Platonic solid. Even if an optimal set of points can not be found, it is still possible to create near-optimal distributions on the sphere[68]. This scheme has been employed in earlier work[7], but is not currently implemented in PyProp due to speed issues. By using n points to discretize the sphere, the cubatures of Sloan and Womersley require $\mathcal{O}(n^2)$ operations to transfer between the grid representation and the basis representation.

If a tensor product of collocation points in θ and ϕ are used instead, it is possible to make the transformation considerably faster. Note that due to the choice of coordinates, the system is 2π periodic in ϕ. Using an equidistant grid in ϕ, the integral in ϕ,

$$\psi_m(\theta) = P_l^{|m|}(\theta) \int_0^{4\pi} e^{\mathrm{i}m\phi} \psi(\theta, \phi), \tag{4.21}$$

is recognized as the Fourier transform of θ, and thus the FFT can be used to perform the transform in $\mathcal{O}(n_\phi \log n_\phi)$ operations. Although there exist fast transforms for the associated Legendre polynomials[57, 72], these methods are only faster for very large basis sets ($n_\phi > 512$). The integral in θ is therefore discretized straight forwardly using a Gauss-Legendre quadrature. Normally we use an equal number of points in θ and ϕ, $n_\theta = \phi = \sqrt{n}$, such that the total number of operations to transform between a grid and basis representation are $\mathcal{O}(n^{3/2} log \sqrt{n})$.

The Hamiltonian can in many situations be written in a form such that it is invariant in ϕ. In such cases, there will be no coupling between different ms in the calculations. The system can then be simplified by removing the m and ϕ dependence from the basis functions altogether.

4.2.4 B-Splines

A common limitation in the global methods described above is that their efficiency being limited by the smoothness of the wavefunction. Additionally, it seems that the global basis functions are either very good for representing continuum states or for representing bound states, but not both at the same time. By using a local method, i.e. a method that only requires access to a small set of grid points (or basis functions) to evaluate operators, we hope to strike a balance between accuracy and computational cost.

B-splines are piecewise polynomials and are thoroughly described in the book by de Boor[16]. In the paper by Bachau et al.[5], a review of B-splines in atomic an molecular physics was given. Each polynomial is non-zero only in a small region, and can be seen as a compact polynomial approximation to a Gaussian function[9]. As the non-zero regions of the basis functons overlap, the overlap matrix $\mathbf{S}$ will be non-trivial.

The regions are formed such that every basis function overlaps with $2k-1$ other basis functions, where k is the polynomial order of the B-splines. This leads to a banded overlap matrix with $2k-1$ bands.

As the basis functions are polynomials, they are easily differentiable, with the differentiated basis functions non-zero only in the same region as the original functions. This results in differentiation matrices $\mathbf{D}$ having the same banded structure as the overlap matrix, and hints that the B-spline basis is a compromise between a space-grid and a momentum-grid.

One of the advantages of the B-spline basis, is the freedom in placement of the basis functions. If highly oscillatory behaviour is expected in a region, such as near the nucleus in an atom, a large number of basis functions can be placed there without complicating the discretization scheme.

The main disadvantage of B-splines is the non-orthogonality of the basis. The overlap matrix must always be carefully considered in order to arrive at correct propagation schemes. Furthermore, as discussed in chapter 3, the overlap matrix must be solved for at a number of occations. This makes parallelization much harder for B-spline based ranks. Currently, the parallelization of B-splines in PyProp is similar to parallelization of global basis functions in that the rank must be local in order to solve for the overlap matrix and perform transforms. This is in contrast to finite differences, which more easily can directly support parallelization.

4.2.5 Finite Difference

Finite differences can be related to interpolating polynomials. We start by defining a set of polynomial basis functions of order k,

$$p_j(x) = \frac{\omega_k(x)}{\omega_k'(x_j)(x-x_j)}, \tag{4.22}$$

with

$$\omega_k(x) = \prod_j^k (x-x_j). \tag{4.23}$$

By letting $\{x_j\}$ coincide with a subset of the the collocation points, the basis functions are seen to be *cardinal basis functions*, i.e. vanishing in all collocation points but one. The Lagrange interpolating polynomial is then

$$P(x) = \sum_j p_j(x)\psi(x_j). \tag{4.24}$$

This can in term be used to approximate the mth derivative of the wavefunction in one of the collocation points,

$$\left.\frac{\partial^m \psi(x)}{\partial x^m}\right|_{x=x_i} \approx \sum_j \left.\frac{\partial^m p_j(x)}{\partial x^m}\right|_{x=x_i} \psi(x_j) = \sum_j \delta_{k,j}^m \psi(x_j). \tag{4.25}$$

A stable method for generating the finite difference coefficients $\delta_{k,j}^m$ was given by Fornberg[21]. Usually, the k collocation points in the region around x_i are used to

approximate the differentiation operator, leading to a banded differentiation matrix D^m with k bands. Combined with the fact that cardinal basis functions make the basis orthogonal in the grid representation, the bandedness makes application of finite differences very fast. Furthermore, the orthogonality and bandedness make it possible to perform highly efficient parallel matrix-vector multiplications with finite difference matrices. For this reason, some of the largest calculations that have been performed on atomic systems have used finite differences [55].

It can be shown[23, 24] that in the limit of $k \to n$, i.e. a finite difference stencil using all grid points, the resulting method is equivalent to the corresponding global basis method. For equidistant collocation points and periodic boundary conditions we get the Fourier method, while other collocation points give different global bases. Thus, finite difference methods can be seen as truncated global basis methods and can be very efficient if the wavefunction is not smooth enough to warrant a global method.

Chapter 5

Applications of PyProp

We will now consider two scientifically relevant but quite different physical examples, and examine how PyProp has been used to simplify the research. The scientific results from these cases are discussed in detail in the corresponding papers, and will not be discussed here. The focus here will instead be on how PyProp is used to perform the calculations in these projects. In addition to the examples listed here, PyProp was used in all the scientific papers included in this dissertation as well as in [52].

5.1 Ion-Molecule Collision

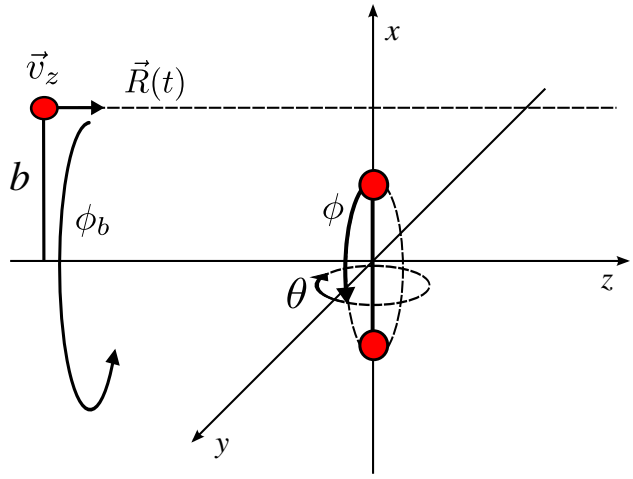

Figure 5.1: Expermiental setup of an ion molecule collision. The ions (described classically) approach the molecular target with a velocity v_z.

Here we will consider the collision of a highly charged ion (Kr^{34+}) with a small diatomic molecular target (H_2). The scientific results from this work is available in Paper **VIII**. The ion is accelerated to a kinetic energy of $\approx 63\,\text{MeV}$. This corresponds to a velocity of $v \approx 50\,\text{a.u.}$, and is sufficient to consider the motion of the ion classically while considering the molecule quantum mechanically. Due to the high velocity of the

ion, the reaction time is very small compared to the vibrational motion of the molecule. The two nuclei in the molecule can be therefore be considered stationary for the duration of the collision. Finally, earlier work on lower dimensional models[67] indicated that a single active electron (SAE) model should be sufficient for this problem. We will therefore only consider one electron in a static molecule, giving rise to the following Hamiltonian,

$$\begin{aligned} H_e &= -\frac{\nabla^2}{2} - \frac{Z_p}{\sqrt{|\mathbf{R}(t)-\mathbf{r}|^2+\beta^2}} \\ &- \frac{Z_H}{\sqrt{|\mathbf{R}_a-\mathbf{r}|^2+\alpha^2}} - \frac{Z_H}{\sqrt{|\mathbf{R}_b-\mathbf{r}|^2+\alpha^2}}. \end{aligned} \tag{5.1}$$

This system is visualized in figure 5.1, with $\mathbf{R}(t)$ being the time-dependent position of the projectile. Z_p and Z_H are the electric charges of the projectile and the target nuclei respectively. α and β are softing parameters to avoid infinite potentials on the grid, and finally $\mathbf{r}$ is the only quantum-mechanical degree of freedom in the system; the electron position. In order to compare results with experiments, we must consider the possible orientations (θ,ϕ) of the molecule, as well as the possible impact parameters (b,ϕ_b) of the ion.

Under these parameters there are no apparent symmetries in the system. We therefore choose the simplest possible coordinate system, i.e. Cartesian coordinates. Furthermore, a Fourier-split step method with a time step $h = 0.006$ is used to propagate the system. Through extensive testing in lower dimensions, it has been found that a grid defined on $x \in [-75,75]$a.u., distributed over $N = 1024$ grid points in each rank is sufficient for a rank 3 problem. This leads to a wavefunction consuming 16GB memory.

A part of the analysis in this project was to remove the bound eigenstates states of the system from the final wavefunction in order to analyze the ionized part of the wavefunction separately. Using the PyProp implementation of the implicitly restarted Arnoldi method discussed in section 3.5.2, the lowest bound states of the system could be found with minimal work. However, due to the size of the problem, the bound state eigenproblem had to be performed on a smaller grid with $x \in [-37.5,37.5]$ and the same grid spacing as the full problem.

It is clear that this problem must be heavily parallelized in order for the calculations to be completed within reasonable time (hours). Using the parallelization scheme described in section 3.2.3, it has been found that using a rank 2 processor grid, the propagation can be scaled past 1024 processing units on the Cray XT4 system we have available. However, it is optimal to use $N_p = 512$ processors for the main propagation as a compromise between fast propagation, and long waiting time in the job queue.

The main reason for using PyProp for this project was initially the availability of a parallelized high performance propagator. The only specialization of PyProp required for propagation and finding eigenstates in this project, was to write the the time dependent and time independent potentials as separate `PotentialClasses` (as discussed in section 2.3.2), and setting up a configuration file. With these two classes, PyProp is able to set up an IRAM solver for finding eigenvalues, as well as a split step propagator for propagating the TDSE.

During the course of the project, we have discovered the advantages of having all

the propagation routines available as a Python library, as it significantly helps making the analysis tools. Due to memory and time constraints, it is currently not possible to work with a 16 GB wavefunction on a single workstation and consequently almost all of the analysis must be performed in parallel. Using the PyProp library, we can parallelize the analysis by using the already parallelized wavefunction loader, Fourier transform, etc. Furthermore, the analysis involves a lot of book-keeping in integrating over impact parameters and averaging over molecular orientations. The expressiveness of the Python language simplifies this book-keeping, allowing us to focus on discovering the physical insights.

5.2 Laser Ionization of Helium

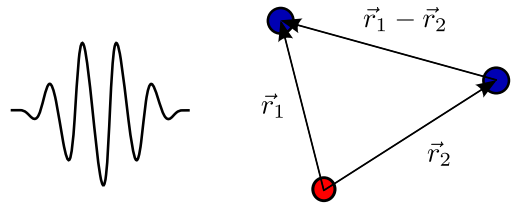

Figure 5.2: Schematic setup for laser ionization of helium. Two electrons (blue) are initially attached to a nucleus (red) of approximately inifite mass. Laser radiation interacts with the two electrons allowing for the possibility of one or two electrons leaving the nucleus. The two electrons each set up a three-dimensional system connected through the electron-electron interaction $\frac{1}{|\mathbf{r}_1-\mathbf{r}_2|}$, making the system six-dimensional.

A very different problem is that of laser ionization of Helium. As sketched in figure 5.2, the Helium atom has two electrons bound to a single nucleus of charge $Z = 2$. The nucleus can here safely be assumed to have infinite mass compared to the electrons. For the time independent problem, i.e. the helium atom without laser interaction, the total Hamiltonian, is a sum of two independent particle Hamiltonians H_0^i in addition to the electron interaction term, i.e. the Coulombic repulsion between two electrons,

$$H_0^i = -\frac{\nabla_{\mathbf{r}_i}^2}{2} - \frac{Z}{r_i} \tag{5.2}$$

$$H_0 = H_0^1 + H_0^2 + \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \tag{5.3}$$

Here, $\mathbf{r}_i$ is the position of electron i with r_i the radial part. $Z = 2$ is the helium charge and $\nabla_{\mathbf{r}_i}$ is the Laplacian corresponding to the ith coordinate. The interaction between the electrons and the electromagnetic field can be described in the velocity gauge. The radiation in the dipole approximation and linearly polarized along the z axis, can be written in terms of the electromagnetic vector field $\mathbf{A}(t)$,

$$H_f(t) = \sum_{i=\{1,2\}} A_z(t) \left(\frac{\partial}{\partial z_i} - \frac{\cos\theta_i}{r_i} \right). \tag{5.4}$$

5.2.1 Discretization

Disregarding the electron interaction term, the system is two three-dimensional problems with perfect spherical symmetry. It is therefore seems like a good idea to choose spherical coordinates for both electrons. Including the electron interaction term, the spherical symmetry is broken, but by mixing two spherical harmonics bases into a *coupled spherical harmonic* basis,

$$\mathcal{Y}^{L,M}_{l_1,l_2}(\Omega_1,\Omega_2) = \sum_m \langle l_1\ l_2\ m\ M-m|L\ M\rangle\, Y^m_{l_1}(\Omega_1) Y^{M-m}_{l_2}(\Omega_2) \tag{5.5}$$

the computation can be simplified. Here, $\langle l_1\ l_2\ m_1\ m_2|L\ M\rangle$ are the Clebsch-Gordan coefficients[11, p. 1004]. By using the coupled spherical harmonic basis, it can be shown that both L and M will be a conserved quantities even when the electron interaction term is considered. For the laser interaction, L will no longer be conserved, but M will still be conserved, reducing the total degrees of freedom in the system from 6 to 5.

Representing the Hamiltonian on spherical collocation points and calculating matrix elements numerically, as described in section 4.2 is not efficient in this case, as the number of spherical grid points required to successfully calculate the matrix elements of the electron interaction term would lead to an enormous wavefunction. Luckily, it is possible to analytically calculate all the angular integrals for the required matrix elements. The electron interaction term can be evaluated by it in a multipole expansion,

$$\frac{1}{|\mathbf{r}_1-\mathbf{r}_2|} = \sum_{l,m} \sqrt{\frac{4\pi}{2l+1}} \frac{r^l_<}{r^{l+1}_>} Y^*_{l,m}(\Omega_1) Y_{l,m}(\Omega_2), \tag{5.6}$$

where $r_<$ and $r_>$ are respectively the smallest and largest of r_1 and r_2. In this form, the angular integrals can be performed analytically.

Similar to the $\{l,m\}$-indices for spherical harmonics, the $\{l_1,l_2,L,M\}$ indices for coupled spherical harmonics does not form a tensor product. All these indices are therefore compressed into one rank in PyProp, represented by a single `CoupledSphericalHarmonicRepresentation` mapping between the unique index k and the coupled indices $\{l_1,l_2,L,M\}$.

Having expanded the wavefunction in coupled spherical harmonics, discretizing the two radial dimensions remains. We have chosen to use the B-Spline basis described in section 4.2.4. Fourier basis functions were discarded due to their weak convergence for radial problems, orthogonal polynomials were discarded as they do not represent the continuum well enough, and high order finite differences require certain tricks[56] with the boundary conditions to work well.

Combining all of the above, we get a rank 3 discretized wavefunction,

$$\psi(\mathbf{r}_1,\mathbf{r}_2,t) = \sum_{i,j,k} c_{i,j,k}(t)\ B_i(r_1)\ B_j(r_2)\ \mathcal{Y}^{L,M}_{l_1,l_2}(\Omega_1,\Omega_2). \tag{5.7}$$

This wavefunction must be parallelized in order to achieve reasonable performance. We have chosen a parallelization scheme where the k-rank is distributed and the two other ranks are processor-local. This is because the non-orthogonality of the B-Spline basis complicates parallelization in those ranks. Because of the choice of propagator, the matrix-vector product must be parallelized. As the Hamiltonian is very sparse in the k-rank, this leads to an efficient parallelization, but it limits the number of processors to the size of the coupled spherical harmonic basis.

5.2.2 Propagation

The system described above is propagated using the Cayley form propagator described in section 3.4, using GMRES to solve the linear system of equations. As noted in section 3.4.1, a preconditioner is required for GMRES to converge within a reasonable number of steps. A block-preconditioner $\mathbf{M}$ has been chosen, with the full r_1 and r_2 matrices in each block, but only the diagonal coupled spherical harmonic elements,

$$M_{(i,j,k),(i',j',k')} = H_{(i,j,k),(i',j',k')}\delta_{k,k'}. \tag{5.8}$$

With the chosen parallelization scheme, this means that the preconditioner is a processor-local sparse matrix. Two solvers have been tested to invert this matrix. First, the SuperLU[17] package was tested, as it can solve the factorize $\mathbf{M} = \mathbf{LU}$ exactly while preserving some of the sparsity structures. However, as the number of B-Spline coefficients increases, the memory requirements and level of fill-in make exact factorization impractical. Thus, the IFPACK preconditioner found in Trilinos[32] has been used instead, providing an incomplete LU factorization (ILU) of $\mathbf{M}$. This was found to be superior in both memory and performance for sufficiently large systems.

As with the project discussed in section 5.1, the main requirement for propagating and finding eigenpairs, was to set up the potential matrices. This was done with the `TensorPotential` framework discussed in section 2.3.2. The angular integrals involved in setting up the matrix elements was all performed analytically, while the automatic basis expansion routines were used to expand the potential in the B-spline bases. The preconditioners used here require direct access to the matrices involved, and not only the matrix vector multiplications. PyProp therefore has routines to convert `TensorPotential`s to `Epetra` (the matrix and vector module in Trilinos) matrices and the wavefunctions to `Epetra` vectors.

5.2.3 Analysis

Among the interesting quantities in this project are energy distributions, $\partial^2 P/\partial E_1 \partial E_2$, and angular distributions, $\partial^4 P/\partial E_1 \partial E_2 \partial \Omega_1 \partial \Omega_2$, as well as the single and double ionization probabilities of the system. In calculating these quantities, the eigenstates of the system is required. The bound states of H_0 can be found by the inverse iteration IRAM scheme described in section 3.5.3. As the time independent problem conserves L, a set of bound states can be found for each L independently.

For the ionizing states of the system, the vast size of the system makes it impractical to find the exact states. Instead, a single electron model is used, where a symmetrized tensor product of two ionizing single electron wavefunctions is used to approximate a double ionization state,

$$\psi^{\pm}_{ion}(\mathbf{r}_1, \mathbf{r}_2) = \psi_a(\mathbf{r}_1)\psi_b(\mathbf{r}_2) \pm \psi_b(\mathbf{r}_1)\psi_a(\mathbf{r}_2) \tag{5.9}$$

Single electron states are found using the same radial discretization scheme as the full two electron problem, but a regular spherical harmonic angular discretization. These systems have perfect spherical symmetry, and can thus be separated in a radial and angular part. The radial parts are small enough to be diagonalized directly, and we get a set of radial eigenstates $R_{n,l}(r)$ for each value of the angular momentum quantum number l. Performing the analysis is then mainly a matter of organizing pairs radial eigenstates and projecting the two electron wavefunction on those pairs.

5.2.4 Conclusion

All the book-keeping code of analyzing the wavefunction is written in Python, with the exception of a few integration routines that are written in C++ for efficiency reasons. In this project, the wavefunction is of the order 100 MB, and can be handled directly on a single processor machine. Under such conditions, the convenience of using Python to perform the analysis is even more significant than in the previously described project. PyProp was significantly helping by giving the analysis tools access to the same discretization and integration routines used in the propagation and eigenvalue solving. Furthermore, the facilities to recreate a wavefunction from a serialized file helped keeping track of the various parameters used in the propagation, such as field strength, time step, grid resolution, etc. Using the `IPython` interactive interpreter, it possible to explore the available data on a completely different level than what is possible in a more traditional edit-compile-run development cycle. The $\partial^4 P/\partial E_1 \partial E_2 \partial \Omega_1 \partial \Omega_2$ data are represented in a four-dimensional array, and it can be difficult to get an overview of what kind of features are present in the data, but by having both data and plotting routines available in an interactive environment it is easy to explore the data to get an overview.

Chapter 6

Conclusions and Outlook

The recurring topic of this thesis has been the development and application of the PyProp framework for solving the time dependent Schrdinger equation. The theoretical background and implementation of PyProp has been discussed in chapters 2-4, while the last chapter demonstrated how PyProp can be used to solve two quite different computationally intensive problems arising in atomic physics.

PyProp currently contains implementations of several methods for both discretization and propagation, but is notably lacking documentation. Some documentation and several examples are availble and distributed with the PyProp source code, but most of this is rather old and not representing the current state of PyProp. Furthermore, it is currently not trivial to compile PyProp on a new platform. There are many dependencies, and some effort is required in order to get a working copy of PyProp on a new computer. This is hindering adoption, and simplifying the installation procedure and updating the documentation are currently the main tasks in maintaining PyProp.

As a way of quickly performing a few initial test calculations on a physical system, PyProp has been very useful. The first parallelized calculations on two-electron helium were performed just few weeks after the project was started. The extensibility has also been successful, as seen by the diversity in the systems that have already been implemented. However, for an ongoing software project as this, it is inevitable that some of the requirements change, making some design decisions unfortunate.

The framework started out as a C++ library with optional Python bindings. Throughout the development, however, the virtues of using Python for scientific software have become apparent. Research is inheritly experimental, and having an interactive environment for plotting and calculations has been proven to be invaluable. Correspondingly, during the development PyProp have become more dependent on Python to the point where it is now a Python extension library written partially in C++. If Python had been tightly integrated from the start, less code would have been written in C++, leading to a more compact code base.

As work was starting on PyProp, propagation schemes based on the split step propagator were the only methods considered in our group. PyProp therefore started out as a framework for selectively combining different one-dimensional transforms. A main requirement for the underlying array library was therefore efficient and simple support for multi-dimensional complex valued arrays. This lead to the conclusion of using the blitz++ array library. While blitz is a very good array implementation in C++, it is possibly too low-level for the current needs of PyProp. Had we started developing PyProp

from scratch now, we would probably have choosen a more high level library for arrays like Trilinos or PETc in order to get more functionality "for free". On the other hand, we have recently implemented some preliminary Trilinos bindings without much trouble, enabling the use of preconditioners like IFPACK, and eigenvalue solvers like Anasazi.

From a physics side, there are now many interesting problems that can be studied with the available tools. The work on double ionization of helium is well under way, and there are many unresolved questions to study, such as the topic of two photon double ionization (TPDI) process[19]. Work is also currently under way to implement a prolate spheroidal coordinate system, which would allow us to further study ionization of diatomic molecules. In this context it would be interesting to follow up Paper **V** with calculations of how good the Franck-Condon approximation really is in this particular case.

We now have a variety of discretization and propagation schemes in a framework which readily supports testing and experimentation. It should therefore be quite possible, and very interesting, to see a comparison of the different methods. Often, such comparisons are limited to model problems, but with the PyProp framework it would be possible to compare the methods on real-world problems.

Chapter 7

Introduction to the papers

Generally, my work has been largely involved in developing PyProp. Particularly, in Papers II and III, my contribution was mainly that of developing and adapting PyProp. However, during the course of this work, I have become more involved in the physical aspects as well. Having some involvment in the physical aspects have been very helpful for me to understand the importance of PyProp not only propagating the wavefunction, but also simplifying analysis.

Paper I: *Parallel Redistribution of Multidimensional Data*

In this paper, we studied redistribution algorithms for multidimensional data. It was found that for data sets of more than two dimensions, it is possible to scale to a very large number of processors by distributing more than one dimension of the data set at the same time. Even if the total amount of data needed to be transferred for a complete redistribution increases with the number of distributed dimensions, the total time spent redistributing was found to be decreased due to the decrease in the total number of data transfer operations. These redistribution schemes are the basis for parallelization in PyProp.

My main contribution to this paper was the implementation and testing of the propagation schemes, an even share of the writing. I also presented this work on the Parco2007 conference.

Paper II: *Structure of lateral two-electron quantum dot molecules in electromagnetic fields*

In this paper, we calculated the eigenstates of a two electron quantum dot, and studied how they change as function of inter-dot distance. This lay the groundwork for further studies of two electron quantum dots[50, 51, 60], in which I did not participate.

My main contribution to this paper was the application of PyProp to calculate the few lowest eigenstates by imaginary time propagation in addition to discussions of the numerical aspects of the calculations.

Paper III: *Classical and quantum-mechanical investigation of the role of nondipole effects on the binding of a stripped HD^{2+} molecule*

In this paper, we considered the possibility of binding two particles of the same charge by introducing extremely intense laser light. In particular, the role of non-dipole terms in the Hamiltonian was studied and found to not have a significant effect on the binding effects from the laser.

In this paper, my main contribution was the discussion of numerical methods and results, as well as the application of PyProp to perform the quantum mechanical calculations.

Paper IV: *Numerical solution of the 3D time dependent Schrödinger equation in spherical coordinates: Spectral basis and effects of split-operator technique*

The topic of this paper is twofold, first it deals with the global discretization schemes for the radial part of the Hamiltonian. Specifically, a transformed Chebyshev collocation method was compared to the Fourier collocation method. Second, the paper discusses the problems of using the split step propagation method when a singular potential, such as the Coulomb potential, is present. We found that splitting singular potentials from the Laplacian in many cases should be and can be avoided at the cost of loosing fast transforms.

My contributions here was to implement the transformed Chebyshev method in PyProp, and performing the numerical experiments on H_2^+. I was also partaking in the general discussions, and in particular the discussion of the splitting error for singular potentials.

Paper V: *Quantum chessboards in the deuterium molecular ion*

In this paper we studied the effect of time delayed laser pulses on the one electron deuterium molecule D_2^+. By first ionizing a D_2 molecule with a femtosecond laser pulse and then carefully delaying a second pulse, it was found possible to selectively populate only a few vibrational states. We demonstrated a scheme in which every second vibrational state was populated and created a very simple model to explain the effect.

I was involved in most of the development and writing process of this paper, using PyProp to perform the numerical calculations, discussing the physical properties, and the development of the simplified model describing the chessboard pattern.

Paper VI: *Parallel Pseudo-spectral Methods for the Time Dependent Schrödinger Equation*

This book chapter gives an overview of some methods for solving the TDSE as well as describing some aspects of the design of PyProp. Similarly to this dissertation, it gives an overview of different propagation and discretization schemes, but of a more limited scope. It was in many ways an early summary of my work.

Paper VII: *Multiphoton Double Ionization in Helium Driven by Intense XUV Attosec-*

ond Pulses

Multiphoton processes have been studied extensively for single electron atoms the last decades. With the increase in computational power, studies of multi-photon processes in two electron systems are now possible as well. In this paper we studied the multi-photon ionization processes in helium, induced by extremely intense XUV laser pulses. In particular, we studied the role of the electron-electron interaction under these conditions. It was found that a very simple one electron model can be sufficient for describing some of the high intensity effects, such as atomic stabilization.

This project is the first of several planned helium studies including two photon double ionization and numerical pump-probe experiments.

In this project I was involved in the physical aspects of the problem the start. I also implemented most of the software required for simulation, analysis and plotting.

Paper VIII: *Non perturbative treatment of single ionization of H_2 by fast, highly charged ion impact*

Here we studied a diatomic molecule ionized by a fast, heavy and strongly charged ion projectile. The ionized electrons can be seen as a superposition of two waves originating from the two nuclei. There are both theoretical and experimental evidence for an interference pattern similar to that of a Young-type experiment. Furthermore, there are experimental evidence for a second order interference pattern, but this has never been successfully modeled by theory. In this paper we tried to find the second order interference patterns by considering a full three dimensional model. We found no second order interference, but found instead a different kind of interference pattern originating from two-center scattering from the target and the *projectile*.

My contribution was initially to develop the numerical method and some of the tools required for analyzing the wavefunction. Later, I have also contributed to the discussion of the physics.

Chapter 8

Scientific results

Bibliography

[1] ABRAHAMS, D., AND GROSSE-KUNSTLEVE, R. W. Building hybrid systems with boost.python. *C/C++ Users Journal July* (2003). 2.2, 2.2.2

[2] ABRAMOWITZ, M., AND STEGUN, I. *Handbook of mathematical functions with formulas, graphs, and mathematical table*. Courier Dover Publications, 1965. 4.2.2

[3] ANDERSON, E., BAI, Z., BISCHOF, C., BLACKFORD, S., DEMMEL, J., DONGARRA, J., DU CROZ, J., GREENBAUM, A., HAMMARLING, S., MCKENNEY, A., ET AL. *LAPACK Users' guide*. Society for Industrial Mathematics, 1999. 3.4

[4] ARNOLDI, W. E. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of applied mathematics 9*, 17 (1951), 17. 3.3, 3.5.2

[5] BACHAU, H., CORMIER, E., DECLEVA, P., HANSEN, J. E., AND MARTIN, F. Applications of b-splines in atomic and molecular physics. *Reports on Progress in Physics 64*, 12 (2001), 1815–1943. 4.2.4

[6] BARMAKI, S., LAULAN, S., BACHAU, H., AND GHALIM, M. The ionization of one-electron diatomic molecules in strong and short laser fields. *Journal of Physics B: Atomic, Molecular and Optical Physics 36*, 5 (2003), 817–824. 4.1

[7] BIRKELAND, T., FOERRE, M., HANSEN, J., AND SELSTOE, S. Dynamics of h (2 p) ionization in ultrashort strong laser pulses. *Journal of Physics B Atomic Molecular and Optical Physics 37*, 20 (2004), 4205–4219. 4.2.3

[8] BIRKELAND, T., AND NEPSTAD, R. Pyprop - a framework for propagating the time dependent schrödinger equation. `http://pyprop.googlecode.com`. 1

[9] BOUMA, H., VILANOVA, A., BESCOS, J., TER HAAR ROMENY, B., AND GERRITSEN, F. Fast and accurate gaussian derivatives based on b-splines. *Scale Space and Variational Methods in Computer Vision* (2008), 406 – 417, doi: 10.1007/978-3-540-72823-8_35. 4.2.4

[10] BOYD, J., RANGAN, C., AND BUCKSBAUM, P. Pseudospectral methods on a semi-infinite interval with application to the hydrogen atom: a comparison of the mapped Fourier-sine method with Laguerre series and rational Chebyshev expansions. *Journal of Computational Physics 188*, 1 (2003), 56–74. 4.2.2

[11] BRANSDEN, B., AND JOACHAIN, C. *Physics of atoms and molecules*, 2nd ed. ed. Prentice Hall, Harlow, 2003. 1, 5.2.1

[12] BURKE, P. G., AND BURKE, V. M. Time-dependent r-matrix theory of multiphoton processes. *Journal of Physics B: Atomic, Molecular and Optical Physics 30*, 11 (1997), L383–L391. 4.2.2

[13] COULANGE, B. *Software Reuse*. Springer Verlag, 1998. 2.2

[14] CRANK, J., AND NICOLSON, P. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Mathematical Proceedings of the Cambridge Philosophical Society 43*, 01 (1947), 50–67, doi: 10.1017/S0305004100023197. 3.4

[15] DAVISON, E. J. A High-Order Crank-Nicholson Technique for Solving Differential Equations. *The Computer Journal 10*, 2 (1967), 195–197, doi: 10.1093/comjnl/10.2.195. 3.4

[16] DE BOOR, C. *A Practical Guid to Splines*. Springer, 2001. 4.2.4

[17] DEMMEL, J. W., EISENSTAT, S. C., GILBERT, J. R., LI, X. S., AND LIU, J. W. H. A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications 20*, 3 (1999), 720–755, doi: 10.1137/S0895479895291765. 3.4, 5.2.2

[18] DUNDAS, D., MCCANN, J. F., PARKER, J. S., AND TAYLOR, K. T. Ionization dynamics of laser-driven h2+. *Journal of Physics B: Atomic, Molecular and Optical Physics 33*, 17 (2000), 3261–3276. 3.3

[19] FEIST, J., NAGELE, S., PAZOUREK, R., PERSSON, E., SCHNEIDER, B. I., COLLINS, L. A., AND BURGDÖRFER, J. Nonsequential two-photon double ionization of helium. *Physical Review A (Atomic, Molecular, and Optical Physics) 77*, 4 (2008), 043420, doi: 10.1103/PhysRevA.77.043420. 6

[20] FEIT, M., JR., J. F., AND STEIGER, A. Solution of the schrödinger equation by a spectral method. *Journal of Computational Physics 47*, 3 (1982), 412 – 433, doi: 10.1016/0021-9991(82)90091-2. 3.2, 4.2.1

[21] FORNBERG, B. Generation of finite difference formulas on arbitrarily spaced grids. *Mathematics of Computation 51*, 184 (1988), 699–706. 4.2.5

[22] Gnu public license. `http://www.gnu.org/copyleft/gpl.html`. 1

[23] GUANTES, R., AND FARANTOS, S. C. High order finite difference algorithms for solving the schr[o-umlaut]dinger equation in molecular dynamics. *The Journal of Chemical Physics 111*, 24 (1999), 10827–10835, doi: 10.1063/1.480446. 4.2.5

[24] GUANTES, R., AND FARANTOS, S. C. High order finite difference algorithms for solving the schr[o-umlaut]dinger equation in molecular dynamics. ii. periodic variables. *The Journal of Chemical Physics 113*, 23 (2000), 10429–10437, doi: 10.1063/1.1324004. 4.2.5

[25] GÜNTHER, M., AND PULCH, R. Part 16. In *Lecture Notes in Numerical Analysis and Simulation of Ordinary Differential Equations*. 2005. 3.4

[26] HAFNER, J., WOLVERTON, C., AND CEDER, G. Toward computational materials design: Density functional theory in materials research. *MRS bulletin 31* (2006), 659. 2

[27] HAGA, C. J. B. Bose-einstein condensates - numerical solution of the gross-pitaevskii equation using finite elements. Master's thesis, University of Oslo, 2006. 3.4.1

[28] HAIRER, E., LUBICH, C., AND WANNER, G. *Geometric Numerical Integration - Structure-preserving Algorithms for Ordinary Differential Equations*. Springer, 2000. 1, 3, 3.1

[29] HAMDI, S., SCHIESSER, W. E., AND W GRIFFITHS, G. Method of lines. *Scholarpedia 2*, 7 (2007), 2859. 2.1.1

[30] HEMMER, P. C. *Kvantemekanikk*. Tapir, 2000. 1.1

[31] HERMANN, M. R., AND FLECK, J. A. Split-operator spectral method for solving the time-dependent schrödinger equation in spherical coordinatess. *Physical Review A 38*, 12 (Dec 1988), 6000–6012, doi: 10.1103/PhysRevA.38.6000. 3.2, 4.2.1

[32] HEROUX, M., BARTLETT, R., HOEKSTRA, V. H. R., HU, J., KOLDA, T., LEHOUCQ, . R., LONG, K., PAWLOWSKI, R., PHIPPS, E., SALINGER, A., THORNQUIST, H., TUMINARO, R., WILLENBRING, J., AND WILLIAMS, A. An overview of trilinos. Tech. Rep. SAND2003-2927, Sandia National Laboratories, 2003. 1, 5.2.2

[33] HOCHBRUCK, M., AND LUBICH, C. On krylov subspace approximations to the matrix exponential operator. *SIAM Journal on Numerical Analysis 34*, 5 (1997), 1911–1925, doi: 10.1137/S0036142995280572. 3.3

[34] HUNTER, J. D. Matplotlib: A 2d graphics environment. *Computing in Science and Engineering 9*, 3 (2007), 90–95, doi: 10.1109/MCSE.2007.55. 1, 2.2

[35] ISERLES, A., MARTHINSEN, A., AND NØRSETT, S. P. On the implementation of the method of magnus series for linear differential equations. *BIT Numerical Mathematics 39*, 2 (June 1999), 281–304. 3

[36] JONES, J. Enhanced interactive python with ipython. *OnLamp* (2005). 2.2

[37] KOHN, W. Nobel lecture: Electronic structure of matter—wave functions and density functionals. *Reviews of Modern Physics 71*, 5 (Oct 1999), 1253–1266, doi: 10.1103/RevModPhys.71.1253. 1

[38] KOHN, W., AND SHAM, L. J. Self-consistent equations including exchange and correlation effects. *Physical Review 140*, 4A (Nov 1965), A1133–A1138, doi: 10.1103/PhysRev.140.A1133. 1

[39] KOZLOV, R. Exponential operator splitting time integration for spectral methods. *Journal of Computational and Applied Mathematics 222*, 2 (2008), 592 – 607, doi: 10.1016/j.cam.2007.12.005. 4.1.2

[40] LANCZOS, C. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of research of the National Bureau of Standards 45*, 4 (1950). 3.3, 3.5.2

[41] LANGTANGEN, H. P. *Python Scripting for Computational Science (Texts in Computational Science and Engineering)*, 3rd ed. ed. Springer, February 2008. 2.2

[42] LUBICH, C. *Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms*. John von Neumann Institute for Computing, 2002, ch. Integrators for Quantum Dynamics: A Numerical Analyst's Brief Review, pp. 459–466. 3, 3.2.4, 3.4

[43] LUNDELAND, M. Spectral approximation for the d-dimensional schrödinger equation: A comparison of the fourier method with laguerre series and hermite series. Master's thesis, University of Bergen, 2007. 4.1.2

[44] MATTHEY, T., CICKOVSKI, T., HAMPTON, S., KO, A., MA, Q., NYERGES, M., RAEDER, T., SLABACH, T., AND IZAGUIRRE, J. Protomol, an object-oriented framework for prototyping novel algorithms for molecular dynamics. *ACM Transactions on Mathematical Software (TOMS) 30*, 3 (2004), 237–265. 1

[45] MCLACHLAN, R. I., AND QUISPEL, G. R. W. Splitting methods. *Acta Numerica 11*, -1 (2002), 341–434, doi: 10.1017/S0962492902000053. 3.2

[46] MEYER, B. *Object-Oriented Software Construction*. Prentice-Hall, 1997. 2.2

[47] MOLER, C., AND LOAN, C. V. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review 45*, 1 (2003), 3–49, doi: 10.1137/S00361445024180. 3, 3.3

[48] MORTENSEN, J. J., HANSEN, L. B., AND JACOBSEN, K. W. Real-space grid implementation of the projector augmented wave method. *Physical Review B 71*, 3 (Jan 2005), 035109, doi: 10.1103/PhysRevB.71.035109. 1

[49] NAUTS, A., AND WYATT, R. E. New approach to many-state quantum dynamics: The recursive-residue-generation method. *Physical Review Letters 51*, 25 (Dec 1983), 2238–2241, doi: 10.1103/PhysRevLett.51.2238. 3.3

[50] NEPSTAD, R., SÆLEN, L., DEGANI, I., AND HANSEN, J. Quantum control of coupled two-electron dynamics in quantum dots. *Journal of Physics: Condensed Matter 21* (2009), 215501. 7

[51] NEPSTAD, R., SÆLEN, L., AND HANSEN, J. Coherent adiabatic theory of two-electron quantum dot molecules in external spin baths. *Physical Review B 77*, 12 (2008), 125315. 7

[52] NEPSTAD, R., VAN DER HART, H. W., AND MCCANN, J. F. Dynamic resonances and tunnelling in the multiphoton ionization of argon. *Journal of Physics B: Atomic, Molecular and Optical Physics 42*, 14 (2009), 145603 (8pp). 5

[53] OLIPHANT, T. Python for scientific computing. *Computing in Science & Engineering 9*, 3 (May-June 2007), 10–20, doi: 10.1109/MCSE.2007.58. 1

[54] PARK, T. J., AND LIGHT, J. C. Unitary quantum time evolution by iterative lanczos reduction. *The Journal of Chemical Physics 85*, 10 (1986), 5870–5876, doi: 10.1063/1.451548. 3.3

[55] PARKER, J. S., MEHARG, K. J., MCKENNA, G. A., AND TAYLOR, K. T. Single-ionization of helium at ti:sapphire wavelengths: rates and scaling laws. *Journal of Physics B: Atomic, Molecular and Optical Physics 40*, 10 (2007), 1729–1743. 4.2.5

[56] PARKER, J. S., MOORE, L. R., MEHARG, K. J., DUNDAS, D., AND TAYLOR, K. T. Double-electron above threshold ionization of helium. *Journal of Physics B: Atomic, Molecular and Optical Physics 34*, 3 (2001), L69–L78. 5.2.1

[57] ROKHLIN, V., AND TYGERT, M. Fast algorithms for spherical harmonic expansions. *SIAM Journal on Scientific Computing 27*, 6 (2006), 1903–1928, doi: 10.1137/050623073. 4.2.3

[58] ROY, A. K., GUPTA, N., AND DEB, B. M. Time-dependent quantum-mechanical calculation of ground and excited states of anharmonic and double-well oscillators. *Physical Review A 65*, 1 (Dec 2001), 012109, doi: 10.1103/PhysRevA.65.012109. 3.5.1

[59] SAAD, Y., AND SCHULTZ, M. H. Gmres: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing 7*, 3 (1986), 856–869, doi: 10.1137/0907058. 3.4.1

[60] SÆLEN, L., NEPSTAD, R., DEGANI, I., AND HANSEN, J. P. Optical control in coupled two-electron quantum dots. *Physical Review Letters 100*, 4 (2008), 046805, doi: 10.1103/PhysRevLett.100.046805. 7

[61] SÆLEN, L., NEPSTAD, R., HANSEN, J. P., AND MADSEN, L. B. The n-dimensional coulomb problem: Stark effect in hyperparabolic and hyperspherical coordinates. *Journal of Physics A: Mathematical and Theoretical 40*, 5 (2007), 1097–1104. 4.1.2

[62] SCHRODINGER, E. Quantisierung als eigenwertproblem. *Annalen der Physik 384*, 4 (1926). 1

[63] Scipy conference. `http://conference.scipy.org/`. 1

[64] SHIRLEY, J. H. Solution of the schrödinger equation with a hamiltonian periodic in time. *Physical Review 138*, 4B (May 1965), B979–B987, doi: 10.1103/PhysRev.138.B979. 3.5

[65] SHULL, H., AND HALL, G. G. Atomic units. *Nature 184*, 4698 (Nov. 1959), 1559–1560, doi: 10.1038/1841559a0. 2

[66] SIDJE, R. B. Expokit: a software package for computing matrix exponentials. *ACM Transactions on Mathematical Software 24*, 1 (1998), 130–156, doi: 10.1145/285861.285868. 3.3

[67] SISOURAT, N., CAILLAT, J., DUBOIS, A., AND FAINSTEIN, P. D. Coherent electron emission from molecules induced by swift ion impact. *Physical Review A 76*, 1 (2007), 012718, doi: 10.1103/PhysRevA.76.012718. 5.1

[68] SLOAN, I., AND WOMERSLEY, R. Extremal systems of points and numerical integration on the sphere. *Advances in Computational Mathematics 21*, 1 (2004), 107–125, doi: 10.1023/B:ACOM.0000016428.25905.da. 4.2.3

[69] SORENSEN, D. Implicit application of polynomial filters in a k-step Arnoldi method. *SIAM Journal on Matrix Analysis and Applications 13*, 1 (1992), 357–385. 3.5.2

[70] STRANG, G. Accurate partial difference methods i: Linear cauchy problems. *Archive for Rational Mechanics and Analysis 12*, 1 (Jan. 1963), 392–402. 3.2

[71] The computer language benchmarks game. `http://shootout.alioth.debian.org/`. 2.2

[72] TYGERT, M. Fast algorithms for spherical harmonic expansions, ii. *Journal of Computational Physics 227*, 8 (Apr. 2008), 4260–4279. 4.2.3

[73] VELDHUIZEN, T. L. Arrays in Blitz++. 2.2.1

[74] WATANABE, N., AND TSUKADA, M. Fast and stable method for simulating quantum electron dynamics. *Physical Review E 62*, 2 (Aug 2000), 2914–2923, doi: 10.1103/PhysRevE.62.2914. 3.4

[75] Weave, a library for integrating python and c code. `http://www.scipy.org/Weave`. 2.2.2

[76] ZANNA, A. Collocation and relaxed collocation for the fer and the magnus expansions. *SIAM Journal on Numerical Analysis 36*, 4 (1999), 1145–1182. 3