ASC Report No. 19/2010

Numerical Methods in Quantum Propagation

Robert Hammerling, Josef Kamleitner, Othmar Koch

Institute for Analysis and Scientific Computing
Vienna University of Technology — TU Wien
www.asc.tuwien.ac.at ISBN 978-3-902627-03-2

Institute for Analysis and Scientific Computing
Vienna University of Technology
Wiedner Hauptstraße 8–10
1040 Wien, Austria

E-Mail: `admin@asc.tuwien.ac.at`
WWW: `http://www.asc.tuwien.ac.at`
FAX: `+43-1-58801-10196`

ISBN 978-3-902627-03-2

Contents

Chapter 1

Introduction

1.1 Motivation

Modern lasers are powerful electromagnetic radiation sources with a frequency domain covering a wide range from the infrared to the ultraviolet. Many laboratories all over the world achieve laser pulses of intensities of $10^{16}\,\frac{\mathrm{W}}{\mathrm{cm}^2}$ and above. The ensuing strong-field phenomena are not accessible through conventional perturbation theory, because the laser field is no longer small compared to the atomic binding forces in the electron shell. In the non-relativistic regime, meaning moderate charge states and moderate laser intensities $< 10^{18}\,\frac{\mathrm{W}}{\mathrm{cm}^2}$, however, the solution of the time-dependent Schrödinger equation is a possible way to understand and analyse these strong-field phenomena (tunneling and above-threshold ionization, high order harmonic generation, nonsequential ionization pronounced dynamic Stark-shifts and dynamic stabilization (the last is not yet confirmed experimentally)).
The first approaches in this direction have been applied to one-electron atoms with the electronic degrees of freedom restricted to the laser polarization direction and have been limited to qualitative analysis. With increasing computer power also three-dimensional grids and not only linearly polarized laser fields can be treated, but due to the "exponential wall" many-electron systems stay numerically inaccessible through the many-body Schrödinger equation [1, 2]. The density functional theory (DFT), relying on the Hohenberg-Kohn and Runge-Gross theorems [3, 4], is based on the electron density instead of many-electron wavefunctions, allowing to treat many-electron systems with linear scaling in the number of particles. Modern formulations and numerical implementations rely on the Kohn-Sham equations which are based on this theoretical observation.
This holds also for the Qprop library package (written in C++) that is the main topic of this work. Based on a one-electron time-dependent Schrödinger solver for interaction with linear polarized laser fields [5] the generalized Qprop library, supporting additionally elliptic polarization and effective many-electron potentials, has been implemented by D. Bauer et al. and documented in [6, 7]. Results and applications can be found in [8, 9].
The aim of this work is to first analyse and document the (numerical) properties of the Qprop library. For this purpose, but also for future use as flexible

input/output (I/O) interface, a front-end is designed and implemented. The detailed analysis shows several possibilities for improvements and optimizations that are implemented in the next step, for instance the order-increasing "Predictor-Corrector" extrapolation step. Finally the Crank-Nicolson time propagator is compared with a standard method of numerical analysis, the newly-implemented Lanczos method.

1.2 Structure of this work

To give briefly a better idea of the intention, contents and results of this work, the following chapters are shortly presented in the following overview:

Chapter 1 gives an introduction to the topic of this work.

Chapter 2 presents the theoretical background of the methods used in QPROP: Time dependent density functional theory and Kohn-Sham equations, (multipole) expansions and approximations, the Hartree and Krieger-Li-Iafrate exchange-correlation potentials.

Chapter 3 introduces the newly implemented QPROP front-end XMLQPROP and contains a detailed manual of this program and the XML (Extensible Markup Language) parameter file specifications.

Chapter 4 describes the most important and the newly implemented C++ classes, points out several improvements to QPROP and gives a detailed analysis of the applied numerical methods.

Chapter 5 presents the main innovations added to QPROP in the course of this work: The additional possibility to use Lanczos method instead of the standard Crank-Nicolson propagator and the introduction of a "Predictor-Corrector" step to improve the order of the numerical method.

Chapter 6 reviews the obtained numerical results. The computational effort and accuracy of the different methods and parameter settings are analysed and compared.

Chapter 7 sums up the main results of this work.

Chapter 2

General analysis of Qprop

This chapter summarizes the general approach of QPROP for the solution of the time-dependent Schrödinger and Kohn-Sham equations. Therefore this chapter mainly merges the information of [6] and [7], but also answers questions that are left open in these two sources.

2.1 Time-dependent Kohn-Sham equations

In the following, Hartree atomic units (a.u.) will be used and the factor $\frac{1}{c}$ is already included in the vector potential $\mathbf{A}$, unless stated otherwise. Table 2.1 shows the values of important constants in a.u. and SI units.
The linear time-dependent Schrödinger equation for a general non-relativistic N-particle quantum system is stated as

$$\mathrm{i}\frac{\partial \Psi(\mathbf{r},t)}{\partial t} = \mathbf{H}\Psi(\mathbf{r},t), \tag{2.1}$$

with the wave-function $\Psi : \mathbb{R}^{3N} \times \mathbb{R} \to \mathbb{C}$ and the Hamiltonian $\mathbf{H}$ acting as a linear operator on the wave-function Ψ. For a single electron the Hamiltonian $\mathbf{H}$ reads

$$\mathbf{H} = \mathbf{H}(\mathbf{r},t) = \frac{1}{2}\left[\mathbf{p} + \mathbf{A}(\mathbf{r},t)\right]^2 + V(\mathbf{r},t) = \frac{1}{2}\left[-\mathrm{i}\nabla + \mathbf{A}(\mathbf{r},t)\right]^2 + V(\mathbf{r},t). \tag{2.2}$$

The extension to many-electron systems is straightforward, but even the two-electron system (atom) exposed to strong laser fields treated in full dimension

dimension	name	sym.	a.u.	SI	
charge	elementary charge	e_0	1	1.60218×10^{-19}	C
mass	electron rest mass	m_e	1	9.10938×10^{-31}	kg
ang. mom.	Planck's red. constant	$\hbar$	1	1.05457×10^{-34}	J s
length	Bohr radius	a_0	1	5.29177×10^{-11}	m
energy	Hartree energy	E_h	1	27.211	eV
time		t_0	1	2.41888×10^{-17}	s

Table 2.1: atomic units

reaches the limits of current supercomputers [1, 2]. The main problem is that the wave-function of many-electron systems depends on $3N$ spatial coordinates, if N is the number of electrons in the system. Thus, the space $\mathbb{R}^{3N}$ would need to be covered by a discrete grid, leading to a number of grid points growing exponentially with the number of electrons ("exponential wall"). For instance neon with 10 electrons and 10 discretization points for each dimension would need 10^{30} grid points, exceeding current and future computational capacities by several orders of magnitude.
Applying time-dependent density functional theory (TDDFT), formulated by the Kohn-Sham equations (2.6) below, instead reduces the numerical effort dramatically as compared to the time-dependent Schrödinger equation (2.1). Density functional theory is based on the Hohenberg-Kohn (steady-state) and the Runge-Gross (time-dependent) theorems, stating that the potential and thus the Hamiltonian $\mathbf{H}$ (without any external potentials) is a functional of the density $n(\mathbf{r}, t)$ only [3, 4].
The electron density is reconstructed from the Kohn-Sham orbitals $\psi_i : \mathbb{R}^3 \times \mathbb{R} \to \mathbb{C}$ by

$$n(\mathbf{r}, t) = \sum_{i=1}^{N} |\psi_i(\mathbf{r}, t)|^2. \tag{2.3}$$

Additionally the Kohn-Sham Hamiltonian $\mathbf{H}_{\mathrm{KS}}$, depending on the Kohn-Sham orbitals via the scalar potential V_{ee}, is given by

$$V_{\mathrm{ee}} = V_{\mathrm{ee}}(\mathbf{r}, t) = V_{\mathrm{ee}}\left[\psi_1, \ldots, \psi_N\right](\mathbf{r}, t), \tag{2.4}$$

$$\mathbf{H}_{\mathrm{KS}} = \mathbf{H}_{\mathrm{KS}}(\mathbf{r}, t) = \frac{1}{2}\left[\mathbf{p} + \mathbf{A}(\mathbf{r}, t)\right]^2 + V(\mathbf{r}, t) + V_{\mathrm{ee}}(\mathbf{r}, t). \tag{2.5}$$

This yields the time-dependent Kohn-Sham equations

$$\begin{aligned} \mathrm{i}\frac{\partial \psi_i(\mathbf{r}, t)}{\partial t} &= \mathbf{H}_{\mathrm{KS}}(\psi_1, \ldots, \psi_N, \mathbf{r}, t)\psi_i(\mathbf{r}, t), && (2.6) \\ \psi_i(\mathbf{r}, 0) &= \psi_i^0, \qquad i = 1, \ldots, N, && (2.7) \end{aligned}$$

where actually the Hamiltonian depends nonlinearly on the orbitals.
This set of partial differential equations describes the same electron density as the many-electron Schrödinger equation, but is computationally tractable even for large N. Note that for the single electron system, the Kohn-Sham equation (2.6) is comparable to the original problem (2.1), if only one Kohn-Sham orbital is used and identified with the physical orbital.

2.2 Expansion into spherical harmonics

Each Kohn-Sham orbital can be expanded into spherical harmonics

$$\psi_i(r, \vartheta, \phi, t) = \frac{1}{r}\sum_{\ell=0}^{\infty}\sum_{m=-\ell}^{\ell} \Phi_{i\ell m}(r, t) Y_\ell^m(\vartheta, \phi). \tag{2.8}$$

Assume that the applied laser field is linearly polarized along the z-axis and that the initial orbitals have well defined magnetic quantum numbers m_i, then m_i

remains a “good” quantum number and therefore all other magnetic quantum numbers in the sum can be neglected:

$$\psi_i(r,\vartheta,\phi,t) = \frac{1}{r}\sum_{\ell=0}^{\infty}\Phi_{i\ell m_i}(r,t)Y_\ell^{m_i}(\vartheta,\phi). \tag{2.9}$$

For the numerical treatment only a finite number of ℓ’s can be included, therefore all ℓ greater than $\ell_{\max} = L-1$ for a given $L \in \mathbb{N}$ are neglected:

$$\psi_i(r,\vartheta,\phi,t) = \frac{1}{r}\sum_{\ell=0}^{\ell_{\max}}\Phi_{i\ell m_i}(r,t)Y_\ell^{m_i}(\vartheta,\phi). \tag{2.10}$$

L has to be chosen carefully and depends on the problem, especially on the intensity of the laser field.

For further simplification and to ease numerical treatment, also the Hamiltonian is approximated and expanded into spherical harmonics. By rewriting equation (2.5) in a different gauge we obtain

$$\mathbf{H}_{\mathrm{KS}} = \mathbf{H}_{\mathrm{KS}}(\mathbf{r},t) = -\frac{1}{2}\nabla^2 + V_{\mathrm{I}}(t) + V(r) + V_{\mathrm{ee}}(\mathbf{r},t), \tag{2.11}$$

with $V_{\mathrm{I}}(t)$ being the interaction with the electromagnetic field in dipole approximation for linear polarization in z-direction

$$V_{\mathrm{I}}(t) = -\mathrm{i}A(t)\frac{\partial}{\partial z} + \underbrace{\frac{A(t)^2}{2}}_{\text{neglected in QPROP}} + zE(t), \tag{2.12}$$

with the vector potential $A(t)$ and the electric field $E(t)$ of the laser pulse, both in z-direction.
The other terms in equation (2.11) are the central-field Coulomb potential $V(r)$ of the nucleus (atomic core) and the Kohn-Sham effective potential $V_{\mathrm{ee}}(\mathbf{r},t)$ expanded into multipole functions $V_{\mathrm{ee}}^j(r,t)$, $j=0,1,2$:

$$V_{\mathrm{ee}} = V_{\mathrm{ee}}(\mathbf{r},t) = V_{\mathrm{ee}}^0(r,t) + V_{\mathrm{ee}}^1(r,t)\cos(\vartheta) + V_{\mathrm{ee}}^2(r,t)\frac{3\cos^2(\vartheta)-1}{2} + \ldots, \tag{2.13}$$

where terms beyond the quadrupole are neglected in QPROP. Note that in (2.13) the multipole functions are defined using the spherical harmonics without the normalization pre-factors.

A finite set of non-linear differential equations for the radial orbitals $\Phi_{i\ell m_i}(r,t)$ is then obtained by inserting (2.10)–(2.13) and

$$V_{\mathrm{eff}}^\ell(r) := V(r) + \frac{\ell(\ell+1)}{2r^2} \tag{2.14}$$

into equation (2.6)

$$\begin{array}{rrlr}
\mathrm{i}\frac{\partial\Phi_{i\ell m_i}(r,t)}{\partial t} & = & \left(-\frac{1}{2}\frac{\partial^2}{\partial r^2}+V_{\text{eff}}^{\ell}(r)\right) & \Phi_{i\ell m_i}(r,t) \\
- & \mathrm{i}A(t)r & \sum_{\ell'}\langle Y_\ell^{m_i}|\cos\vartheta|Y_{\ell'}^{m_i}\rangle\frac{\partial}{\partial r}\frac{1}{r} & \Phi_{i\ell' m_i}(r,t) \\
+ & \mathrm{i}A(t) & \sum_{\ell'}\langle Y_\ell^{m_i}|\sin\vartheta\frac{\partial}{\partial\vartheta}|Y_{\ell'}^{m_i}\rangle\frac{1}{r} & \Phi_{i\ell' m_i}(r,t) \\
+ & rE(t) & \sum_{\ell'}\langle Y_\ell^{m_i}|\cos\vartheta|Y_{\ell'}^{m_i}\rangle & \Phi_{i\ell' m_i}(r,t) \\
+ & & \sum_{\ell'}\langle Y_\ell^{m_i}|V_{\text{ee}}^{\leq 2}(r,t)|Y_{\ell'}^{m_i}\rangle & \Phi_{i\ell' m_i}(r,t),
\end{array} \tag{2.15}$$

$$i=1,\dots,N, \qquad \ell=0,\dots,L-1.$$

Due to the truncation of the multipole expansion of the Kohn-Sham potential V_{ee}, only a few of the matrix elements in equation (2.15) are non-zero. To express the contributing matrix elements explicitly, we define:

Definition 2.1. Coefficients $c_{\ell m}$, $t_{\ell m}$, $p_{\ell m}$ and $q_{\ell m}$

$$c_{\ell m} := \sqrt{\frac{(\ell+1)^2-m^2}{(2\ell+1)(2\ell+3)}}, \tag{2.16}$$

$$t_{\ell m} := (\ell+1)\sqrt{\frac{(\ell+1)^2-m^2}{(2\ell+1)(2\ell+3)}} = (\ell+1)c_{\ell m}, \tag{2.17}$$

$$p_{\ell m} := \frac{\ell(\ell+1)-3m^2}{(2\ell-1)(2\ell+3)}, \tag{2.18}$$

$$q_{\ell m} := \frac{3}{2(2\ell+3)}\sqrt{\frac{[(\ell+1)^2-m^2][(\ell+2)^2-m^2]}{(2\ell+1)(2\ell+5)}}. \tag{2.19}$$

Thus, we obtain:

$$\langle Y_\ell^{m_i}|\cos\vartheta|Y_{\ell\pm1}^{m_i}\rangle = c_{\ell-\frac{1}{2}\pm\frac{1}{2},m_i}, \tag{2.20}$$

$$\left\langle Y_\ell^{m_i}|\sin\vartheta\frac{\partial}{\partial\vartheta}|Y_{\ell\pm1}^{m_i}\right\rangle = \pm t_{\ell-\frac{1}{2}\pm\frac{1}{2},m_i}, \tag{2.21}$$

$$\left\langle Y_\ell^{m_i}|\frac{3\cos^2(\vartheta)-1}{2}|Y_\ell^{m_i}\right\rangle = p_{\ell m_i}, \tag{2.22}$$

$$\left\langle Y_\ell^{m_i}|\frac{3\cos^2(\vartheta)-1}{2}|Y_{\ell\pm2}^{m_i}\right\rangle = q_{\ell-1\pm1,m_i}. \tag{2.23}$$

The $V_{\text{ee}}^j(r,t)$ terms of the expansion of the Kohn-Sham potential $V_{\text{ee}}(r,t)$ (2.13) consist each of the Hartree part U^j and the exchange-correlation part V_{xc}^j. In the following sections these parts are explained.

2.2.1 Expansion of the Hartree potential

The Hartree potential U is defined as

$$U(\mathbf{r},t) = \int_{\mathbb{R}^3}\frac{n(\mathbf{r}',t)}{\|\mathbf{r}-\mathbf{r}'\|}d^3r' \tag{2.24}$$

and expanded into spherical harmonics analogously to the Kohn-Sham potential in (2.13). Then the corresponding multipole terms are

$$U^j = \int_0^\infty \frac{r_<^j}{r_>^{j+1}} 2 \sum_{i=1}^N \tilde{\Lambda}_{ii}^j(r',t)dr', \tag{2.25}$$

with

$$r_< := \min(r, r'), \tag{2.26}$$

and

$$r_> := \max(r, r'). \tag{2.27}$$

Up to the quadrupole term, for $j = 0, 1, 2$, the terms $\tilde{\Lambda}_{ii}^j(r,t)$ are

$$\tilde{\Lambda}_{ii}^0(r,t) = \sum_\ell \overline{\Phi}_{i\ell m_i}(r,t)\Phi_{i\ell m_i}(r,t) = \sum_\ell |\Phi_{i\ell m_i}(r,t)|^2, \tag{2.28}$$

$$\tilde{\Lambda}_{ii}^1(r,t) = \sum_\ell \left(c_{\ell-1,m_i}\overline{\Phi}_{i,\ell-1,m_i}(r,t) + c_{\ell m_i}\overline{\Phi}_{i,\ell+1,m_i}(r,t)\right)\Phi_{i\ell m_i}(r,t), \tag{2.29}$$

$$\tilde{\Lambda}_{ii}^2(r,t) = \sum_\ell \left(q_{\ell-2,m_i}\overline{\Phi}_{i,\ell-2,m_i} + p_{\ell m_i}\overline{\Phi}_{i\ell m_i} + q_{\ell m_i}\overline{\Phi}_{i,\ell+2,m_i}\right)\Phi_{i\ell m_i}, \tag{2.30}$$

with the coefficients of Definition 2.1, and with terms Λ, Θ and Ξ defined as follows:

$$\Lambda(r,t) = 2\sum_{i=1}^N \tilde{\Lambda}_{ii}^0(r,t), \tag{2.31}$$

$$\Theta(r,t) = 2\sum_{i=1}^N \tilde{\Lambda}_{ii}^1(r,t), \tag{2.32}$$

$$\Xi(r,t) = 2\sum_{i=1}^N \tilde{\Lambda}_{ii}^2(r,t). \tag{2.33}$$

Putting the equations (2.25)–(2.33) together, the multipole terms of the Hartree potential read

$$U^0 = \int_0^\infty \frac{1}{r_>}\Lambda(r',t)dr', \tag{2.34}$$

$$U^1 = \int_0^\infty \frac{r_<}{r_>^2}\Theta(r',t)dr', \tag{2.35}$$

$$U^2 = \int_0^\infty \frac{r_<^2}{r_>^3}\Xi(r',t)dr'. \tag{2.36}$$

2.2.2 Krieger-Li-Iafrate (KLI) exchange potential approximation

In DFT, there is a variety of approximations to the exchange-correlation potential V_{xc}. In QPROP the Krieger-Li-Iafrate (KLI) approximation is implemented. The KLI potential $V_{\mathrm{x}}^{\mathrm{KLI}}$ consists of the local Slater exchange potential $V_{\mathrm{x}}^{\mathrm{S}}$ and a residual term $\tilde{V}_{\mathrm{x}}$

$$V_{\mathrm{xc}}(\mathbf{r},t) \approx V_{\mathrm{x}}^{\mathrm{KLI}}(\mathbf{r},t) = V_{\mathrm{x}}^{\mathrm{S}}(\mathbf{r},t) + \tilde{V}_{\mathrm{x}}(\mathbf{r},t). \tag{2.37}$$

In this section, the time dependence will not be stated explicitly for better readability and because all terms depend on the same time variable t. The KLI potential is an approximation to the optimized effective potential (OEP). The OEP is implicitly defined by an integral equation, but this equation is numerically not well treatable. The simplified integral equation for the KLI potential is

$$V_{\mathrm{x}}^{\mathrm{KLI}}(\mathbf{r}) = V_{\mathrm{x}}^{\mathrm{S}}(\mathbf{r}) + \sum_{i=1}^{N-1} \frac{|\psi_i(\mathbf{r})|^2}{n(\mathbf{r})} \int_{\mathbb{R}^3} |\psi_i(\mathbf{r}')|^2 \left(V_{\mathrm{x}}^{\mathrm{KLI}}(\mathbf{r}') - u_{\mathrm{x}i}(\mathbf{r}')\right) d^3r', \tag{2.38}$$

with the Slater potential defined as

$$V_{\mathrm{x}}^{\mathrm{S}}(\mathbf{r}) = \sum_{i=1}^{N-1} \frac{|\psi_i(\mathbf{r})|^2}{n(\mathbf{r})} u_{\mathrm{x}i}(\mathbf{r}). \tag{2.39}$$

Here, $u_{\mathrm{x}i}(\mathbf{r})$ is given by

$$u_{\mathrm{x}i}(\mathbf{r}) = \frac{1}{\overline{\psi}_i(\mathbf{r})} \frac{\delta E_{\mathrm{x}}[\{\psi_k(\mathbf{r})\}]}{\delta \psi_i(\mathbf{r})} = -\sum_{k=1}^{N} \frac{\psi_k(\mathbf{r})}{\psi_i(\mathbf{r})} \int_{\mathbb{R}^3} \frac{\overline{\psi}_i(\mathbf{r}')\psi_k(\mathbf{r}')}{\|\mathbf{r}-\mathbf{r}'\|} d^3r', \tag{2.40}$$

with the set of Kohn-Sham orbitals $\{\psi_k(\mathbf{r})\} = \{\psi_k(\mathbf{r}) | k = 1, \ldots, N\}$ and the exchange energy E_{x},

$$E_{\mathrm{x}}[\{\psi_k\}] = -\sum_{i=1}^{N}\sum_{j=1}^{N} \int_{\mathbb{R}^3}\int_{\mathbb{R}^3} \frac{\overline{\psi}_i(\mathbf{r})\overline{\psi}_j(\mathbf{r}')\psi_j(\mathbf{r})\psi_i(\mathbf{r}')}{\|\mathbf{r}-\mathbf{r}'\|} d^3r' d^3r. \tag{2.41}$$

Applying the integral operator defined for a function $f \in \mathrm{L}^2$ by

$$\langle f \rangle_j := \int_{\mathbb{R}^3} f(\mathbf{r}) |\psi_j(\mathbf{r})|^2 d^3r \tag{2.42}$$

to both sides of the integral equation (2.38), yields the matrix equation (2.44) below for the coefficients Q_i,

$$Q_i := \left\langle V_{\mathrm{x}}^{\mathrm{KLI}} - u_{\mathrm{x}i} \right\rangle, \tag{2.43}$$

$$\sum_{i=1}^{N-1} \left(\delta_{ji} - M_{ji}\right) Q_i = \left\langle V_{\mathrm{x}}^{\mathrm{S}} - u_{\mathrm{x}i} \right\rangle, \tag{2.44}$$

with the symmetric matrix M_{ji} given by

$$M_{ji} = \int_{\mathbb{R}^3} \frac{|\psi_j(\mathbf{r})|^2 |\psi_i(\mathbf{r})|^2}{n(\mathbf{r})} d^3r. \tag{2.45}$$

It can be shown that Q_N is equal to 0, therefore the N^{th} term is neglected in some of the sums above. The solution of (2.44) provides all necessary coefficients such that the right-hand side of (2.38) is explicitly known as well as the KLI potential $V_{\mathrm{x}}^{\mathrm{KLI}}$.
In QPROP only the monopole term $V_{\mathrm{x}}^{\mathrm{KLI},0}$ of the potential is computed, this is sufficient if most of the electrons stay inside the atomic shell and the deviation from spherical symmetry remains small.

2.2.3 The approximate Hamiltonian

To summarize the results and approximations of Section 2.2 we observe that the set of differential equations integrated in QPROP is

$$\mathrm{i}\frac{\partial \Phi_{i.m_i}(r,t)}{\partial t} = \mathbf{H}_{\mathrm{KS}}(\mathbf{r},t)\Phi_{i.m_i}(r,t), \tag{2.46}$$

with the approximate Hamiltonian (semidiscretized: discrete ℓ, continuous r, t)

$$\mathbf{H}_{\mathrm{KS}} = \mathbf{H}_{\mathrm{at}} + \mathbf{H}_{\mathrm{mix}} + \mathbf{H}_{\mathrm{ang}}^{(1)} + \mathbf{H}_{\mathrm{ang}}^{(2)} + \mathbf{H}_{\mathrm{ang}}^{(3)}, \tag{2.47}$$

with the parts

$$\begin{aligned}
\mathbf{H}_{\mathrm{at}} &= \delta_{\ell\ell'}\left(-\frac{1}{2}\frac{\partial^2}{\partial r^2} + V_{\mathrm{eff}}^{\ell}(r) + V_{\mathrm{ee}}^{0}(r,t) + p_{\ell m_i} V_{\mathrm{ee}}^{2}(r,t)\right), && (2.48)\\
\mathbf{H}_{\mathrm{mix}} &= -\mathrm{i}A(t)\left(\delta_{\ell-1,\ell'} c_{\ell-1,m_i} + \delta_{\ell+1,\ell'} c_{\ell m_i}\right)\frac{\partial}{\partial r}, && (2.49)\\
\mathbf{H}_{\mathrm{ang}}^{(1)} &= \mathrm{i}A(t)\left(\delta_{\ell-1,\ell'} t_{\ell-1,m_i} - \delta_{\ell+1,\ell'} t_{\ell m_i}\right), && (2.50)\\
\mathbf{H}_{\mathrm{ang}}^{(2)} &= \left(rE(t) + V_{\mathrm{ee}}^{1}(r,t)\right)\left(\delta_{\ell-1,\ell'} c_{\ell-1,m_i} + \delta_{\ell+1,\ell'} c_{\ell m_i}\right), && (2.51)\\
\mathbf{H}_{\mathrm{ang}}^{(3)} &= V_{\mathrm{ee}}^{2}(r,t)\left(\delta_{\ell-2,\ell'} q_{\ell-2,m_i} + \delta_{\ell+2,\ell'} q_{\ell m_i}\right), && (2.52)
\end{aligned}$$

where δ is the Kronecker delta.

2.3 Overview of the propagation algorithm

2.3.1 The discretized Hamiltonian

Based on the approximate Hamiltonian (2.47) and Kohn-Sham orbitals (2.10) already discrete in ℓ- and m- space, only the radial discretization has to be specified. This discretization is given on an equidistant grid

$$r_{i_r} = \frac{i_r}{n_r} r_{\max}, \qquad i_r = 0, \ldots, n_r, \tag{2.53}$$

where the wavefunction value at the point $r = 0$ is not stored, because the wavefunction is zero there. In general all properties of the wavefunction at $r = 0$ are taken into account, if necessary. Internally, however, the index $i_x = i_r - 1$

starts at 0 and ends at $n_x - 1 = n_r - 1$, as it is standard in C++. This means that the radial wavefunction values are stored in the interval $[\Delta r, r_{\max}]$ with a step width

$$\Delta r = \frac{r_{\max}}{n_r}, \tag{2.54}$$

as a complex valued one dimensional array $\mathbf{\Phi}$ in the `wavefunction` class (Section 4.1.7). The `grid` class (Section 4.1.6) offers mapping functions which map the indices for the Kohn-Sham orbital ($i = i_z + 1$), for the angular momentum ($\ell = i_y$) and the radial position ($i_r = i_x + 1$) to the general index $\imath$ via

$$\imath = ((i_z n_y) + i_y n_x) + i_x, \tag{2.55}$$

with all indices starting at zero in this equation (i and i_r start at one and are only used in theory, but not in the implementation). This means that the components of the discrete vector $\mathbf{\Phi}$ as discretization of the wavefunction at a certain time t are

$$\mathbf{\Phi}_\imath(t) = \Phi_{i\ell m_i}(r_{i_r}, t). \tag{2.56}$$

The discrete Hamiltonian acts on this vector depending on the time propagation algorithm. In the standard propagation mode using a Crank-Nicolson approximation (Section 4.2.3) the Hamiltonian parts (2.47),(2.48)–(2.52) are split into 2×2 matrices that are calculated and applied consecutively. In the Lanczos method propagation mode the whole Hamiltonian is computed and stored in a sparse matrix format (see Section 5.1.2) and then applied using Lanczos method. All the potentials occurring in the Hamiltonian (2.47) are realized as discrete vectors with values at the same radial points as the wavefunciton itself. An overview on the discretization and approximation steps is given in the following Section 2.3.2. The differential operators, where also boundary conditions matter, are the subject of Section 4.2.1, while the additionally introduced imaginary absorbing potential is described in Section 4.2.2.

2.3.2 Summarizing the discretization

As conclusion of this chapter a short summary of the approximation and discretization steps in QPROP is given, starting with the continuous partial differential equation (PDE) and finishing with the propagator being discrete in space and time. While the time has been an arbitrary real number up to now, from now on it is assumed to be in the interval $[0, T]$ for a positive real number T. This is the interval where the solution is computed. The range of commonly (unless stated otherwise) used variables and parameters are listed in Table 2.2.

a) Continuous PDE-problem

For the Kohn-Sham orbitals

$$\psi_i : \left\{ \begin{array}{rcl} \mathbb{R}^3 \times [0,T] & \to & \mathbb{C} \\ (\mathbf{r}, t) & \mapsto & \psi_i(\mathbf{r}, t) \end{array} \right. , \qquad \forall i, \tag{2.57}$$

$$\psi_i(., t) \in L^2(\mathbb{R}^3, \mathbb{C}), \qquad \forall t, i, \tag{2.58}$$

variable or index	range	remark
$\mathbf{r}$	$\mathbb{R}^3$	
r	$\mathbb{R}^+$	$r = \|\mathbf{r}\|$
r	$0, \Delta r, \ldots, r_{\max}$	discrete grid
ϑ	$[-\pi, \pi]$	
ϕ	$[0, 2\pi]$	
t	$[0, T]$	
t	$0, \Delta t, \ldots, T$	discrete grid, $\Delta t = h$
i	$1, \ldots, N$	
i_z	$0, \ldots, n_z - 1$	$i_z = i - 1$, $n_z = N$
ℓ	$0, \ldots, \ell_{\max}$	$\ell_{\max} = L - 1$, $n_y = L$
i_y	$0, \ldots, \ell_{\max}$	$i_y = \ell$
i_r	$0, \ldots, n_r$	
i_x	$0, \ldots, n_x - 1$	$i_x = i_r - 1$, $n_x = n_r$
$\imath$	$0, \ldots, n_z n_y n_x - 1$	$\imath = ((i_z n_y) + i_y n_x) + i_x$

Table 2.2: Range of variables and parameters

the Kohn-Sham equations are

$$\mathrm{i}\frac{\partial \psi_i(\mathbf{r},t)}{\partial t} = \mathbf{H}_{\mathrm{KS}}(\psi_1, \ldots, \psi_N, \mathbf{r}, t)\psi_i(\mathbf{r},t), \tag{2.59}$$

$$\psi_i(\mathbf{r},0) = \psi_i^0, \qquad \forall i. \tag{2.60}$$

b) Spherically discrete, radially and temporally continuous PDE-problem

Both the orbitals and the Hamiltonian are expanded into spherical harmonics and only the first L angular momenta with $\ell = 0, \ldots, \ell_{\max} = L - 1)$ are taken into account. This yields the following spherically discrete orbitals

$$\Phi_{i\ell m_i}(r,t) : \begin{cases} \mathbb{R}^+ \times [0,T] & \to & \mathbb{C} \\ (r,t) & \mapsto & \Phi_{i\ell m_i}(r,t) \end{cases}, \qquad \forall i, \ell, \tag{2.61}$$

$$\Phi_{i\ell m_i}(.,t) \in L^2(\mathbb{R}^+, \mathbb{C}), \qquad \forall t, i, \ell, \tag{2.62}$$

and equations

$$\mathrm{i}\frac{\partial \Phi_{i.m_i}(r,t)}{\partial t} = \mathbf{H}_{\mathrm{KS}}(\Phi_{1.m_1}, \ldots, \Phi_{N.m_N}, r, t)\Phi_{i.m_i}(r,t), \qquad \forall i, \tag{2.63}$$

$$\Phi_{i\ell m_i}(r,0) = \Phi_{i\ell m_i}^0, \qquad \forall i, \ell. \tag{2.64}$$

c) Spatially discrete, temporally continuous ODE-problem

By additional discretization in radial direction, for the fully discretized orbitals

$$\mathbf{\Phi}_\imath : \begin{cases} [0,T] & \to & \mathbb{C} \\ t & \mapsto & \mathbf{\Phi}_\imath(t) \end{cases}, \qquad \forall \imath, \tag{2.65}$$

$$\mathbf{\Phi}(t) \in \mathbb{C}^{n_z n_y n_x}, \qquad \forall t, \tag{2.66}$$

the following ordinary differential equation (ODE) is obtained

$$\begin{aligned} \mathrm{i}\frac{\partial \boldsymbol{\Phi}(t)}{\partial t} &= \mathbf{H}_{\mathrm{KS}}(\boldsymbol{\Phi}(t), r, t)\boldsymbol{\Phi}(t), && (2.67)\\ \boldsymbol{\Phi}(0) &= \boldsymbol{\Phi}^0. && (2.68) \end{aligned}$$

d) Discrete time propagator

Finally also the time is discretized by a numerical propagator

$$U_{\mathrm{num}}(t+h, t, V_{\mathrm{ee}}(\boldsymbol{\Phi}(t))) : \left\{ \begin{array}{ccc} \mathbb{C}^{n_z n_y n_x} & \to & \mathbb{C}^{n_z n_y n_x} \\ \boldsymbol{\Phi}(t) & \mapsto & \boldsymbol{\Phi}(t+h) \end{array} \right. , \quad \forall t, \qquad (2.69)$$

computing the discrete orbitals $\boldsymbol{\Phi}$ for all discrete time grid points starting at zero

$$\begin{aligned} \boldsymbol{\Phi}(t+h) &= U_{\mathrm{num}}(t+h, t, V_{\mathrm{ee}}(\boldsymbol{\Phi}(t)))\boldsymbol{\Phi}(t), && (2.70)\\ \boldsymbol{\Phi}(0) &= \boldsymbol{\Phi}^0, && (2.71) \end{aligned}$$

with time step size $h = \Delta t$. This scheme is used in the standard propagation mode. As described in Section 5.2 in detail, the structure of the propagator changes with the introduction of a “Predictor-Corrector” step to

$$\begin{aligned} \tilde{\boldsymbol{\Phi}}(t+h) &= U_{\mathrm{num}}\left(t+h, t, V_{\mathrm{ee}}(\boldsymbol{\Phi}(t))\right)\boldsymbol{\Phi}(t), && (2.72)\\ \boldsymbol{\Phi}(t+h) &= U_{\mathrm{num}}\left(t+h, t, \frac{V_{\mathrm{ee}}(\boldsymbol{\Phi}(t)) + V_{\mathrm{ee}}(\tilde{\boldsymbol{\Phi}}(t+h))}{2}\right)\boldsymbol{\Phi}(t), && (2.73)\\ \boldsymbol{\Phi}(0) &= \boldsymbol{\Phi}^0, && (2.74) \end{aligned}$$

where (2.72) is the “Predictor” and (2.73) the “Corrector” step.

2.3.3 Imaginary time propagation

The Qprop package and the implemented numerical methods of quantum propagation are designed to solve the time-dependent Schrödinger / Kohn-Sham equations for an atom interacting with an external laser field. As starting point of this computed time evolution an initial state is necessary, but in most cases not available in closed form. Often, this initial state is the ground state of the unperturbed quantum system (external fields equal to zero). Many numerical methods computing the ground state of an atom are based on the fact that the total energy of the ground state is minimal. One of these methods is applied in Qprop, where the real time propagator is used to compute the ground state via the so-called imaginary time propagation.
The imaginary time propagation simply replaces the real time step $h = \Delta t$ by the imaginary time step $h = -\mathrm{i}\Delta t$.
For the linear one-particle Schrdinger equation, with the wavefunction $\Psi(\mathbf{r}, 0)$ at time equal to zero expanded into eigenstates $\psi_n^0(\mathbf{r})$ of the Hamiltonian with eigenvalues E_n (superposition principle), the wavefunction at positive time reads

$$\Psi(\mathbf{r}, t) = \sum_n a_n \exp\left(-\mathrm{i}E_n t\right)\psi_n^0(\mathbf{r}). \qquad (2.75)$$

The propagation by one imaginary time step yields

$$\Psi(\mathbf{r}, h = -\mathrm{i}\Delta t) = \sum_n a_n \exp\left(-E_n \Delta t\right) \psi_n^0(\mathbf{r}). \tag{2.76}$$

For a negative ground state energy $E_{n_0} < 0$ the corresponding eigenstate $\psi_{n_0}^0(\mathbf{r})$ explodes fastest. Therefore propagation of an initial guess (even random) wavefunction by many imaginary time steps and wavefunction re-normalization after each of these consecutive steps converges to the ground state $\psi_{n_0}^0(\mathbf{r})$.
Although theoretical results for the non-linear case are not clear, it is possible to use this imaginary time propagation also in the case of several Kohn-Sham orbitals. The Pauli exclusion principle is satisfied by performing Gram-Schmidt orthogonalization in each imaginary time step.

Chapter 3

The Qprop front-end XMLQprop

3.1 Intention and purpose of XMLQprop

The QPROP package consists of both the QPROP library and some example programs using this library. These example programs demonstrate the computation of the imaginary and real time propagation of hydrogen and neon as well as some other possible applications of QPROP like ionization calculations.
Nearly the whole QPROP package source code is written in C++, only a small number of C and Fortran functions is used, mainly from external libraries such as LAPACK and BLAS.
While the QPROP library is written in an object oriented programming style, the examples only show how the library should be used, but the spectrum of covered problems is limited. For instance all variables and parameters are set inside the source code, in some cases by using global variables. This is sufficient to illustrate the interfaces of the QPROP library and to solve some special problems, but for a detailed analysis of the QPROP code much more flexible software is necessary.
This led to the development of the program XMLQPROP (originally called np, the acronym for "new program") as flexible front-end for the QPROP library and basis for the results obtained in this work. Several goals should be achieved by XMLQPROP:

1. To enable a detailed analysis of the QPROP library and its abilities. Therefore the program has to be able to solve a set of problems contained in the original QPROP examples.

2. To provide the solution of problems to the QPROP library and only negligibly induce overhead in CPU-time and memory requirements. Improvements of QPROP are to be done directly in the library, not outside in the front-end XMLQPROP.

3. To remove parameter definitions from the source code. The parameters are defined in an XML-file, such that the program can solve different problems, specified only by these XML-files, without re-compilation.

4. To remove global variables, such that only use of local variables is sufficient.
5. To replace the former globally defined functions for potentials by subtypes of a newly defined class potential. Using dynamic binding for these subtypes depending on XML-file parameters enables highest flexibility with respect to the used potentials.
6. To provide the ability to solve one problem with a variety of spatial and temporal discretization parameters and the possibility to vary additional parameters like the laser field intensity. The results for the different parameters should be summarized for easy comparison.
7. To add the possibility to time the CPU-time that is consumed by Qprop for solving the problems posed by these different varied parameters.
8. To produce several output files, containing details on each solved problem, a summary of loops over various parameters and of computation times of the different parts of Qprop.
9. To achieve a high re-usability of the program code. Therefore it should be designed using object oriented programming paradigms.
10. To use standard solutions in the form of standard libraries for standard problems, for instance LAPACK and BLAS for linear algebra and Xerces-C for XML parsing.
11. To design the program to run on several platforms independently of the system type.
12. To add extensive documentation for the code by comments inside the source code as well as by the manual given in Section 3.2 and the README-file in Appendix A.

3.2 XMLQprop manual

The program XMLQprop is a non-interactive C++-program using the Qprop library and an additional `Dynpar` class to achieve the goals defined in Section 3.1 above. Furthermore, the standard C++ libraries, the LAPACK, BLAS, f2c/g2c, expokit and Xerces-C libraries are included.
The structure of the program is very simple, it consists of two main parts: First the information from the XML parameter file is read in and then the part performing all computations and outputs follows.

3.2.1 Paths, compilation and execution

In the `qprop/src/xmlqprop` directory all data immediately used by the program are contained. The program's source code is located in the `xmlqprop.cc` C++ source file, the XML parameter files are contained in the `param` sub-directory and output files are written into the `res` sub-directory.
The `Makefile` contains all path inclusions and library links necessary for compilation of the program. Before `make` compiles the code properly, it has to be

ensured, that all necessary libraries (including development files in some cases) are installed at the system (see Appendix A). Especially the delivered EXPOKIT library has to be previously compiled.
Depending on whether the f2c or the g2c library is used, make has to be called by `make` (if the f2c library is located in a directory that is included for linking by default) or `make xmlqprop_64` (if the g2c library is contained in the `/usr/lib64` directory as it is common for 64 bit systems). For other installations with different f2c/g2c library paths, the `Makefile` has to be changed such that this library is linked. If the f2c library is not installed on the system the f2c library version in the directory `libf2c` (located in the same directory as QPROP) can be compiled calling `make` in this directory (see also Appendix A). Then `make xmlqprop_c` has to be called to compile the program properly.
To execute XMLQPROP using a parameter file `paramfile1.xml` (located in the `param` subdirectory) the following statement has to be entered on the command line: `./xmlqprop paramfile1`. Note that the `.xml` file extension has to be omitted as well as the `param` directory name. To run the application in background with niceness 10 (such that other applications of standard priority do not slow down significantly) the shell script `xqs` (acronym for XMLQPROP-shell-script) may be called via `sh ./xqs paramfile1`. The output is then piped into `res/paramfile1.out` and the error output into `res/paramfile1.err`.

In the following sections the two main parts of the program XMLQPROP will be described in detail.
In Section 3.2.2 the first part, reading in the XML parameter file information, will be explained as well as the meaning of the parameters and their implications for the program.
Section 3.2.3 does not describe any parts of the program but the input and output files and their contents.
For an overview of the program as a black-box algorithm, meaning that only the input parameters and the results are of interest, these two sections are sufficient. To get detailed information on how the program works, the description of the algorithmic and computational second part of the program (Section 3.2.4) should also be perused.

3.2.2 Parameter initialization and XML file structure

Before the initialization of parameters as defined in the XML parameter file, most of the variables and pointer variables used in the main program are declared, only a few of them are also initialized using default values.
Then the number of arguments is checked. This has to be equal to 2, meaning that one variable, the parameter file name, is specified. If this is not the case, the user is informed how to call the program correctly.
Otherwise, if the number of variables is correct, the program continues with the XML DOM (Document Object Model) parser block. First, the XML DOM parser is initialized, then the program to open the parameter file with the user-specified name. If the file exists and can be opened, the program starts to parse the XML parameter file to initialize the parameter values. Otherwise respective error messages are returned.
Corresponding to the XML specifications there is only one root element, called `params`. The content of this element is in between the start-tag `<params>` and

the end-tag `</params>`. The same structure is also valid for the child elements in the content of the root element, they consist of the start-tag, their content, and the end-tag. All elements may have child elements and their number is not limited.
The root element owns four child elements (these are 1. `static_params`, 2. `dynamic_params`, 3. `init_params` and 4. `lanczos_params`), each summarizing parameters of different meaning. Their child elements are the parameters and arrays of parameters. Because XMLQprop searches only for element names and does not classify by the parent elements, the names of elements in the root, the first and the second level have to be pairwise different. This is only a weak restriction because of the limited number of parameters as compared to the number of possible element names, but it enables to introduce new classes of parameters to increase the clarity of the structure. For instance the initialization parameter layer `init_params` was introduced during the program development because of the increasing number of parameters describing wave-function initialization.
The third level of child elements is only used for array elements, therefore it is no problem if their names are identical to elements of other arrays, because the program only searches for the array's, the parent element's, name.
An example for a parameter file is shown in the Appendix, Section B.1. To distinguish between integer and double parameters having integer values, the latter ones have a ".0" added. The meaning of the parameters and their effects on the behaviour of the program XMLQprop is described in the following list.

1. `static_params`

`realtimemode` 0 for imaginary time propagation, 1 for real time propagation. If imaginary time propagation is selected, the parameters `extrapolate`, `numtisranget` and all `init_params`, except `i_initmode` are neglected or not used and have therefore no effect. Otherwise, if real time propagation is selected, `i_initmode` is obsolete.

`hydrogenlike` 0 for many-electron atoms, 1 for one-electron atoms/ions like the hydrogen atom. The `extrapolate` and `i_slateronly` parameters are neglected for hydrogen-like atoms, because the "Predictor-Corrector" extrapolation step and the KLI potential are only meaningful for many-electron atoms.

`extrapolate` 1 if the "Predictor-Corrector" extrapolation step should be performed, 0 otherwise. Only useful in real time propagation mode for many-electron atoms.

`numtisranget` 0 means that the parameter `range_t` stands for T, the length of the time interval $[0, T]$ that is computed using enough steps of length $h = \Delta t$ (the number of steps depends on the length of the time interval). 1 means, on the other hand, that `range_t` stands for the number of time steps of length $h = \Delta t$ that are computed, such that the length of the time interval depends on the number of steps ($T =$`range_t`Δt). In imaginary time propagation mode this parameter is ignored and the program always behaves as if it were 1, because only a stationary solution is to be computed after a number of steps and this does not have anything to do with a time interval. For real time propagation mode this parameter is mostly set equal to 0 because the main intention is to compute the solution within a time interval for different temporal discretization step size. Setting the parameter equal to 1 in real time mode is useful if one wants to test the time reversibility of the algorithm by performing only one step forward and one step back afterwards.

`lanczos` 0 for using the standard Crank-Nicolson time propagation, 1 for using the newly implemented Lanczos propagation algorithm. Naturally, if this parameter is 0, then all `lanczos_params` are neglected.

`i_v` is an old parameter for switching additional output on (value 1) and off (value 0). The usage of this parameter in XMLQPROP is very limited.

`i_slateronly` is a parameter important for the computation of the KLI potential (2.37) and therefore only used for many-electron atoms. If it is 0 the matrix equation (2.44) is solved, otherwise, if it is 1, only the Slater potential is used. For atoms with a single orbital, like helium, only `i_slateronly` equal to 1 is possible.

`dimens` 34 for linear polarization (standard value within this work) and 44 for circular polarization of the laser field.

`n_orb` stands for the number of Kohn-Sham-orbitals N. Internally the enumeration of the Kohn-Sham-orbitals is done by the so-called z-index i_z, such that `n_orb` is also called n_z (see Table 2.2). The `degener`, `ms`, `ells` and `really_propagate` arrays below, describing the Kohn-Sham-orbitals individually, are of length `n_orb`.

`n_ang_mom` stands for the number of included angular momenta, this is the constant L as defined in Section 2.2, defining the upper limit $\ell_{\max} = L - 1$ in equation (2.10). The higher `n_ang_mom`, the finer is the angular disretization. Internally ℓ is implemented as the so-called y-index i_y, `n_ang_mom` is therefore also called n_y (see also Table 2.2).

`degener` is an array of length `n_orb` defining the degeneracy of each Kohn-Sham-orbital. For most orbitals it is to be chosen equal 2, because the electron density is assumed to be spin-unpolarized. For the ground state of neon, for example, where the closed shells are spherically symmetric, it is also possible to treat all 2p-electrons by one Kohn-Sham-orbital with degeneracy 6.

`ms` is an array whose i^{th} element defines the magnetic quantum number m_i of the i^{th} Kohn-Sham-orbital.

`ells` is an array whose i^{th} element defines the initial angular momentum quantum number ℓ of the i^{th} Kohn-Sham-orbital. It is used for the initialization of the Kohn-Sham orbitals as hydrogenic orbitals, but due to imaginary and further real time propagation the orbital wavefunctions cover all included ℓ's from 0 to $\ell_{\max}$, they are not fixed to the values given in the `ells` parameter array.

`really_propagate` is an array defining which Kohn-Sham-orbitals should be frozen (value 0) and which should be propagated (value 1). In this work always all orbitals are propagated. If only the outer shell is of interest, the inner shell orbitals may be frozen to reduce the computation time.

`nuclear_charge` is the charge of the atom's nucleus in atomic units (terms of unit charge e).

`imagpot_ampl` sets the amplitude A_{imag} of the imaginary absorbing potential (4.13) at the outer radial boundary. A standard value is 100.

`imagpot_inner_border` The inner border, where the imaginary absorbing potential starts. In detail this parameter defines the ratio $\frac{r_{\text{abs}}}{r_{\text{max}}}$ in equation (4.13) below.

2. `dynamic_params`

`range_x` is the radial range covered by the spatial discretization points, corresponding to the maximal radius r_{max} that is treated within the computation such that the electron density outside the sphere of radius r_{max} is neglected. Depending on the size of the electron shells, this parameter should be chosen in the area of 10 to several hundred atomic units.

`range_t` Normally this parameter defines the length of the time interval $[0, T]$ in which the solution should be computed. For imaginary time propagation or if the parameter `numtisranget` is 1, `range_t` stands for the number of time steps to be computed.

`n_outputs` stands for the maximal number of outputs, i.e. how often additional information (orbital energies, etc.) is computed, displayed and written to the output files. These computations are not necessary at each time step but have non-negligible numerical effort. Especially for small time steps it is sufficient and useful to choose `n_outputs` much less than the number of time steps. If this parameter is greater than the number of time steps, the additional computations and output are performed each time step.

The following dynamic parameters correspond to `Dynpar`-class instances that are initialized using the child element parameters `i`, `f`, `a` and `b`. The `Dynpar`-class realizes dynamic parameters, starting at an initial value ($v_0 =$`i`), returning the next value ($v_{j+1} = \mathtt{b}v_j + \mathtt{a}$) by the member function `next()` until the final value (`f`) is reached. The member function `p(j)` returns the value v_j. $\mathtt{i} \overset{\leq}{\underset{>}{=}} \mathtt{f}$ is allowed, the class recognizes if the parameter values increase or decrease and after the values go beyond / below the final value, `NaN` is returned. So loops over all `Dynpar`-class parameter values can be terminated if `NaN` is returned. To avoid infinite loops it has only to be assured that the iteration increases the parameter value if `i`$\leq$`f` and decreases it if `i`$>$`f`. Using only one parameter value can be realized by setting `i`$=$`f` and, for example, `b`$= 2$.

`dx` is the spatial radial discretization step width Δr in atomic units as a dynamic parameter. The quotient of `range_x` over `dx` gives the number of radial discretization points $n_r = n_x$.

`dt` is the time step size Δt in atomic units as a dynamic parameter.

`intens` is the intensity of the laser field as a dynamic parameter.

3. `init_params`

`i_initmode` is an integer value defining the Kohn-Sham orbital initialization mode for imaginary time propagation. If this parameter is 1, the Kohn-Sham orbitals are initialized with random numbers, otherwise, if it is 2, the Kohn-Sham orbitals are initialized using the hydrogen orbitals. In the real time propagation mode this parameter is neglected.

The following initialization parameters are only used in the real time propagation mode and describe the properties of the ground state wave-function used for initialization at $t = 0$.

`filenamepart_ini` is used in real time propagation mode to define the input file name of the file containing the initial wave-function. If `multiinit` (described below) is 0, then a single file, named `./res/[filenamepart_ini].dat` (if `filenamepart_ini` is `default` this means `./res/default.dat`), is used for initialization for all different dynamic parameter sets.
If `multiinit` is 1, then a different file is used for wave-function initialization depending on the specific dynamic parameter set. The filename is therefore specified similar to the imaginary time propagation final wavefunction output filename `"./res/%s_%s_i%g_dx%g_dt%g.dat"` with `filenamepart_ini` replacing the first `%s`, "`if`" replacing the second `%s` and the dynamic parameter values for intensity, spatial and temporal step width replacing the three `%g`'s.
If `multiinit` is 2 the initial wavefunction files differ only for different `dx`, their names depend neither on `intens` nor on `dt`. Therefore the filename is defined by `"./res/%s%g.dat"` with the `%s` replaced by `filenamepart_ini` and the `%g` replaced by the current `dx` value.

`multiinit` specifies, whether one general (value 0) or multiple specific files, depending on the dynamic parameter values (value 1 or 2), are used for the initialization of the wave-function in real time propagation mode.
If 0, all initialization parameters that are described below specify the properties of the initial wave-function, that is to be read in from the file. Otherwise, if not 0, the initialization parameter `dx_ini` is replaced by the dynamic parameter value `dx`. The difference between value 1 and 2 lies only in the parameter file names as mentioned above in the description of `filenamepart_ini`.

`n_orb_ini` stands for the number of Kohn-Sham orbitals N in the initial wave-function.

`n_ang_mom_ini` stands for the number of included angular momenta L in the initial wave-function

`range_x_ini` is the maximal radius $r_{\max}$ of the spatial discretization area of the initial wave-function.

`dx_ini` is the radial discretization parameter Δr of the initial wave-function

After the initial wave-function is read in from the file it is transformed from the initial coordinate space to the real time propagation mode coordinate space (trivial transformation for `multiinit`= 1). In space the transformation is piecewise linear, in the other coordinates (orbital and angular momentum), as well as in space, previously undefined domains are initialized with 0 and values in cut off domain parts are simply neglected.

4. `lanczos_params`

These parameters are only considered in the Lanczos propagation mode, otherwise they are neglected.

`adaptive` if 0, exactly one step using the Lanczos method is performed each time step (as defined by the dynamic parameter `dt`). If 1, the adaptive step width control is switched on and the step width for the Lanczos method is individually chosen inside each time step as defined by `dt`. Depending on the problem and the step width `dt`, up to several hundred adaptive steps are performed within one time step of length `dt` (see also `mxstep`).

`hermitian` specifies whether the matrix of the Hamiltonian is Hermitian (value 1) or not (value 0). If the matrix is not Hermitian, the Arnoldi method is used instead of the Lanczos method, leading to additional numerical effort. Additionally, in this case, it is not possible to use the memory saving Lanczos implementation of the `Expokit_memsave`-class instead of the `Expokit`-class (the parameter `memsave` is ignored) and furthermore the memory requirement for storing the Hamiltonian matrix (in compressed sparse column (CSC) instead of Hermitian CSC (HCSC) format) nearly doubles.

`memsave` if 0 the standard Lanczos / Arnoldi method implementation is used, otherwise, if 1, the memory saving Lanczos implementation is used. This implementation allows to use only the memory for three vectors of length $L \times n_r$. In the standard version `max_dim_krylov` of these vectors (or up to two more, depending on the parameter `adaptive`) consume memory. The disadvantage of the memory saving version is that the vectors have to be recomputed, but not the inner products. So the number of operations increases, but does not double, while the memory requirement is reduced significantly especially for large `max_dim_krylov`.

`dx_order` specifies the order of the approximation of the differential operators in the Hamiltonian by the Hamiltonian matrix. If 2, the standard explicit second order approximation is used, otherwise (value 4) the explicit fourth order approximation is used. Different values result in a warning and the use of the 2$^{\text{nd}}$ order version. Up to now Numerov and Simpson approximation, as used in the non-Lanczos-method mode, are not implemented, because this would require a newly implemented matrix class with an internally split Hamiltonian matrix (for instance using Cholesky decomposition and HCSC matrices).

`max_dim_krylov` is the maximal dimension of the Krylov space used by the Lanczos / Arnoldi method. This is the main parameter for the specification of accuracy and numerical effort of the method, especially in the non-adaptive case. Useful values lie in the region of 10 to 40, depending on the specific problem.

`loc_tol` defines the local tolerance for the Lanczos method. This parameter is important for the so-called "happy breakdown" as well as for the adaptive step size in the Lanczos method implementation. If it is lower than *eps*, $\sqrt{eps}$ is chosen instead.

`mxstep` is the maximal number of (adaptive) steps in which the given fixed step is divided for the Lanczos method. A typical number is several hundreds. This parameter value should not be much bigger, otherwise the Hamiltonian matrix is not updated frequently enough. On the other hand, if it is much less than this, the adaptive step algorithm cannot work properly. If `mxstep` is exceeded, a warning is displayed because then not the whole time interval is computed, the computation stops after `mxstep` steps. Thus this parameter cannot be used to influence the adaptive step selection, it is only important to check whether it works in a useful way.

`mxreject` determines how often a proposed time step size can be rejected before a step is performed. This parameter is commonly chosen equal to 1000 and in normal operation conditions it should not be reached. But at certain points of the time interval this upper limit for resizing the step size may help to avoid infinite loops such that the adaptive algorithm can go over such difficult points where it would slow down or even stop otherwise.

`pade_deg` controls the degree of the Padé approximation of the matrix exponential $\exp(-\mathrm{i}\tau T)$ in the Lanczos / Arnoldi method. The standard value is 6. For rather large `max_dim_krylov` this value should also be increased.

`lanczos_delta` This parameter is important for the adaptive step size selection, the local error is enforced to be lower than (`lanczos_delta`) $\times$ (time step size) $\times$ (the tolerance `loc_tol`) for the current step size to be accepted. The typical value is 1.2.

`lanczos_gamma` also plays an important role for the adaptive time stepping. To calculate the next step size that is expected to be the best (in the beginning, after a successful step or a step size rejection), a certain formula using `lanczos_gamma` as a kind of step size reduction factor is applied. The typical recommended value is around 0.9 [10].

3.2.3 Input and output files and their contents

In this section the structure and contents of input and output files (except the XML parameter file) will be briefly described.

Initial and final wavefunction files

These files store the initial and final wavefunction of the calculation. There are no format strings in the file, each line consists of the real part, followed by the imaginary part of a complex number. These complex numbers are the wavefunction values and the numbering of the lines corresponds to the numbering of the elements as provided by the `grid` class.
By the first base string `"./res/%s_%s_i%g_dx%g_dt%g.dat"` the filename is in principle defined, the first "`%s`" is replaced by the XML parameter file name and the second one by the identifier string. All possible identifier strings are listed

identifier string	meaning
`ii`	imaginary initial wavefunction
`if`	imaginary final wavefunction
`ri`	real time initial wavefunction
`rf`	real time final wavefunction
`io`	imaginary observation file
`ro`	real time observation file
`sum`	summary file
`atiming`	the all parts timing file
`ptiming`	the propagation timing file

Table 3.1: File identifier strings

in Table 3.1 The "`%g`"s are replaced by the values of the intensity, the spatial and the time step size.
In imaginary time propagation mode the wavefunction is initialized by the program XMLQprop and written to the inital wavefunction file, after the computation the final wavefunction is written to the final wavefunction file. In real time propagation the behaviour for the input file depends on the `multiinit` parameter (see also description of `multiinit` and `filenamepart_ini` above). If it is 0 only one wavefunction file, whose name is defined in the parameter file, is loaded and the wavefunction transformed to the current grid is written into an initial wavefunction file. Otherwise pairwise distinct wavefunctions are loaded from wavefunction files named dynamic parameter dependent and no other initial wavefunction files are created. For the final wavefunction the real time propagation mode does the same as the imaginary propagation mode.
For input the function `init(grid,file*,...)` and for output the function `dump_to_file_sh(grid,file*,...)` are used, both are `wavefunction` class members.

Observation file

The observation file starts with a comment line that describes the contents of the columns. This header line depends on the real / imaginary time propagation mode. The file name is also based on the first base string and the file identifier strings are listed in Table 3.1.

Summary file

The summary file summarizes all information about the final state after the time propagation for the different combinations of dynamic parameters. It contains a header line, in the first three columns there are the dynamic parameter values, followed by the total energy, the wavefunction norm and the overall computation time for this dynamic parameter set.
Because there is only one summary file per XML parameter file and this does not depend on the values of the dynamic parameters, its name is created by the second base string `"./res/%s_%s_li%d_lx%d_lt%d.dat"`. The place holders for the string have the same function as for the first base string, they stand for the XML parameter file name and the file identifier, that is `sum` in this case. The

"`%d`"s stand for the number of different dynamic parameters in intensity, space and time.

Timing files

The front-end XMLQprop times several parts of the computation, and the results of these timings are stored into the "all parts" and the propagation timing files, named corresponding to the internally used class instance names `atimer` and `ptimer` of the `JKtimer` class (see Section 4.1.4). In the "all parts" file nearly all computational steps in the inner (time discretization) loop are resolved, but this timer does not go into called functions, only their CPU time consumption is measured. For the propagation algorithm this is not sufficient, therefore the `ptimer` instance is passed to the propagate function and internally parts of the algorithm are timed.
The name of the file, besides the file identifier (Table 3.1), corresponds to that of the summary file. In the file there is no header, the lines correspond to the parameter sets as in the summary file and the columns correspond to the timed parts as they are at best determined directly out of the source code. This is clear, because timing beyond the summarized time of the summary file is only necessary for direct optimization or evaluation of source code parts.

3.2.4 Dynamic parameter loops, algorithms and computation

After parsing the XML parameter file and initializing the corresponding parameter sets, the second and main part of the program XMLQprop starts.
This part consists of three big loops over the dynamic intensity, space and time parameters. The latter, inner loops are nested into the first, outer loops. Additionally there is some initialization before and some finalization after each loop. The description of the program part consists of the description of the loops, can be seen as a sort of algorithm description and follows now.

Intensity loop

initializations before Directly after parsing the XML-file as described in Section 3.2.2 above, the maximal ℓ and m quantum numbers are determined and the `Coefficients` class (Section 4.1.1) instance `coeff` is initialized. Afterwards, immediately before the loop over the dynamic intensity parameter starts, the summary file, the "atiming" and "ptiming" files are opened. While the summary file summarizes the total final energy, the wavefunction norm and the computation time for each combination of the three dynamic parameters, the "atiming" file lists the computation time for program parts in the program XMLQprop (for instance the `propagate(...)`, `calculate_kli_zero(...)` and other functions) and the "ptiming" file lists the computation time for the different parts of the propagation algorithm. In all cases the computation time is provided by the `JKtimer` class (see Section 4.1.4).

the loop The computations inside the loop consist of the radial spatial discretization loop, the initializations before and the finalizations after this loop, all described in the next paragraph.

finalizations afterwards After the intensity loop the three files that were opened before it are closed and allocated variables (for instance `hamilton` and `coeff`) are deleted. Finally, before the program terminates, the message *"finished loops over parameters :)"* is displayed.

Radial spatial discretization loop

The potential `pot` of class `Potential` is initialized before the loop and the current laser field intensity is displayed. The computations inside the loop consist of the time discretization / step size loop, the initializations before and the finalizations afterwards, all described in the next paragraph. After the radial spatial discretization loop the allocated potential `pot` is deleted.

Time discretization / step size loop

Here, inside the inner loop, but also before it, where various initializations are performed, especially for all variables that depend on space, but not on time discretization, lie the main parts of the program.

initializations before After the radial discretization parameter and the number of spatial discretization points are computed and displayed several class instances are initialized: The grid `g` (of class `grid`), the Hamiltonian `hamilton` (of class `Hamop`, representing the static potential and the laser field potential `pot`), the static potential `staticpot` (of class `wavefunction`), and, in case of Lanczos mode, also the double precision arrays `realstaticpot` and `expimagstaticpot` as well as the Expokit-implementing class instance (`expo`, `cexpo` or `mexpo`, depending on the `memsave` and `hermitian` parameters). Furthermore the wavefunction `wf` (of class `wavefunction`) is initialized. In the case of many-electron atoms the Hartree and exchange correlation potentials are also initialized and, if the "Predictor-Corrector" step is to be performed, also the additionally required intermediate value potentials and wavefunctions.

the loop First the current step size is displayed and the timers as well as the `expimagstaticpot` are initialized. Then the program is split by an if-statement into the real time propagation and the imaginary propagation part. After these two parts the results are written into the summary and the timing files and the timers are deleted.
For both the real and imaginary time propagation first the number of time steps is computed and displayed, the same is done for the number of outputs. Then in both cases the wavefunction is initialized: For the imaginary time propagation with random numbers or hydrogenic orbitals depending on `i_initmode`, for the real time propagation the data is read in from files, depending on the `multiinit` parameter, and transformed to the current grid using the `transform_grid_lin(...)` function provided by the `wavefunction` class. Then the initial wavefunction is written to a file and the time step size as well as the starting time is set, in both modi, before the observation file (the main

output file) is opened and a header is written into it.
Then again, the program splits into two branches, one for hydrogen-like atoms and one for many-electron atoms. After this splitting the observation file is closed and the final wavefunction is written into a file.
All four inner program branches have the same structure, a for-loop over the index of the time steps, with only slight differences depending on the `realtimemode` and `hydrogenlike` parameters. This time step loop can be characterized by three main parts: potential computation, wavefunction propagation and output. First the Hartree and exchange correlation potentials are calculated, clearly only in the many-electron case. Then the propagation over the time step is done using the propagate function corresponding to the `hydrogenlike` and Lanczos parameters. In the Lanczos propagation mode the imaginary potential is not applied within the `propagate` function, but by calling the `apply_imagpot` function of the `wavefunction` class. If a "Predictor-Corrector" is performed the calculation of the potentials and the propagation (with the corrected potentials) is done a second time. Then the time variable `t` is updated in the real time mode while the wavefunction is normalized in the imaginary time propagation mode. After the propagation part the output part follows, but this last part is not called in each cycle of the for-loop, it is only run `n_outputs` times. First all relevant data for the output are computed, these are several energies and the norm of the wavefunction. Additionally the current time and time step index are contained in the output on the screen and in the observation file.

finalizations afterwards Compared to the initializations before this loop the finalization afterwards is very short, only a few explicitly allocated class instances and arrays are deallocated using the `delete` statement.

Chapter 4

Detailed analysis and improvements

Although some observations supplementing the Qprop manuals have been explained in Chapter 2, the standard references to the algorithms underlying the Qprop code are [6] and [7]. In contrast to this general information only little detailed analysis of the Qprop code has been previously available. So the first main target was to give this detailed analysis of the Qprop code, as it will be done in the following. The basis (Chapter 2) as well as the tool (XMLQprop, see Chapter 3) have already been described, now the results, as well as the improvements that were implemented based on these results, will be discussed.

4.1 Basic computations and improvements

4.1.1 Coefficients class

In Qprop the coefficients $c_{\ell m}$, $t_{\ell m}$, $p_{\ell m}$ and $q_{\ell m}$, defined in Definition 2.1, equations (2.16)–(2.19), are used for many computations.
Therefore the `Coefficients` class is introduced, implementing the computation of all necessary coefficients when initialized at the beginning of the program and offering get-functions to use the coefficients' values in arbitrary computations. To support old programs that use Qprop and do not initialize this class the old interfaces are retained (Section 4.1.5) and specific `Coefficients`-subclasses are used for these properties.
In previous versions of Qprop these coefficients are computed on demand. Their values are computed each time immediately before their usage. Although there is some optimization, like computation outside of loops, in general the implementation concerning these coefficients has still left a lot of further optimization potential. Additionally the readability of the code is improved when using the coefficients' names instead of their definitions. A further advantage is that the coefficient computation is only implemented once reducing the probability of an error in the formula.

4.1.2 Potential class

In the examples provided with QPROP version 1.6 the external laser field and internal atomic potentials are defined by several functions in a source file named `potentials.cc`. This file also contains the parameters of the potentials as global variables. Inside the example programs, for instance `hydrogen_re.cc` these global variable values are set inside the source code.
This type of implementation has many disadvantages. To change the initialization of the global parameters, such that they are read in from a parameter file, would be one simple improvement because then the program does not have to be recompiled when using different parameters. But then the code still has to be compiled for potentials of different shapes that cannot be described by these parameters, for instance if one wants to use a potential ramp and also a sine-shaped potential. For these cases it would be possible to introduce further parameters and if statements inside the function returning the strength of the potential for a radial position. But for this solution it would be very difficult to add further different potentials.
Therefore another way is chosen in this work, enabling an arbitrary number of different potentials in an easy way. A new class `Potential` is introduced. This class offers all necessary functions formerly defined in the included `potentials.cc` file as virtual functions. A certain potential is then a subtype of the `Potential` class where these virtual functions are implemented.

4.1.3 Hamop class

The `hamop` class (uncapitalised) in QPROP version 1.6 is the link between the globally defined potential functions from `potentials.cc` and their use in the QPROP library, for instance in the `propagate(...)` functions. The implementation of this class uses function pointers to the globally defined potential functions. This approach is not only suboptimal because of the use of global functions, the concept of function pointers is also incompatible with non-static class member functions and does therefore not work with the `Potential` class solution.
Thus, it has been necessary to implement a new `Hamop` class (capitalised) as a first step. Its two subclasses `Hamop_ptr` and `Hamop_pot` copy the function pointer properties of the `hamop` class in the first (`ptr` for pointer) and implement the link to the `Potential` class in the second case (`pot` for potential). To ensure full backwards compatibility a second step has to be performed, to replace the `hamop` class by the `Hamop_ptr` class is not sufficient, because then the function interfaces cannot be kept unchanged. Here, the main problem is, that the concept of function pointers is very inflexible and static as already mentioned above. The only possibility and therefore the chosen solution was to pass the function pointers, pointing at functions of arbitrary names, to global functions with a fixed naming. The only restriction herein is that it is only possible to use one old `hamop` class instance at the same time. This is more a theoretical restriction, because “at the same time” means the parallel use of at least two `wavefunction` class member functions to which different `hamop` class instances are passed as parameters. In contrast to this, consecutive use of different `hamop` class instances, that may be in memory at the same time, a more likely scenario, cause no problems, because the global functions are updated at

each function call.

4.1.4 JKtimer class

To be able to measure the CPU-time required by the different parts of the program, the `JKtimer` class is introduced. An instance `timer` has to be created. The clock is started by the function `timer->stop_t(-1)`, by calling `timer->stop_t(m)` the `m`'th partial time (internally called "lap" like in sports timing) is stopped. The function `timer->get_dt(m)` then returns the arithmetic average of the `m`'th part's time requirement. The use of this class is an easy way to measure the timings of algorithm parts and requires only negligible additional effort.

4.1.5 Backwards compatibility

All improvements and extensions in Qprop are implemented ensuring backwards compatibility. This means that all programs that are using Qprop in an older version are able to use the Qprop library in the new enhanced version. The old function interfaces refer internally to the new, overloaded function interfaces. When calling via the old function interfaces, some new functionality is not available. For instance instead of the fully implemented `JKtimer` class only a dummy class, without any functionality, is used in between, or the coefficients in the `Coefficients` class are initialized each time `propagate(...)` is called. This is no disadvantage, because the numerical effort of the new version is lower in general and the programs designed for the old Qprop version have no use for the additional features.
In addition, some extra overloaded function interfaces (for `propagate(...)`, for example) are introduced. This offers the possibility to use only a part of the new improvements, for instance timing is very important in this work for analyzing the numerical effort, but for other purposes this function may be obsolete. In general everyone can choose the interfaces being most useful for a certain problem using a powerful base implementation.

4.1.6 Grid class

One of the basic classes in Qprop is the `grid` class. If stores and provides information about the polarization of the laser field (function `dimens()`, value 34 for linear, 44 for circular polarization) and the (spatial) discretization grid. The class was originally used to describe Cartesian coordinates x, y, and z that correspond to the coordinates r, ℓ and the Kohn-Sham orbital index i. Internally the variable and function names remained unchanged, therefore the internally used indices i_x, i_y and i_z mean index in r, index in ℓ and index i of the Kohn-Sham orbital. In addition the starting index has to be taken care of, which internally is always zero (see Section 2.3, Table 2.2). The discretization parameter (step size) in radial direction is also part of the `grid` class (variable `delta_x`). The corresponding variables for y and z as well as the offset variables are obsolete in Qprop.
This very important class is not very complex. Thus, only the index mapping functions computing the general index (2.55) are optimized, all other parts are trivial from the numerical point of view and left no margin for improvements.

4.1.7 Wavefunction class

The `wavefunction` class is the substantial and most comprehensive part of the Qprop library, it contains practically all functions with appreciable computational effort. Therefore it was clear to concentrate on this class to optimize Qprop and eliminate possible errors. Additional functions and interfaces were also introduced to facilitate optimal usability of the class and further several comments were added to improve the readability of the source code, too. In the following the most important changes, improvements, optimizations, corrections and amendments are discussed.

1. *Constants*
   In the `wavefunction.cc` file some constants have been defined (by `#define OOS 1.0/6.0`, for instance). This sort of definition as fractions can easily be improved by replacing the definition with the floating point value (`#define OOS 0.16666666666666667`) such that no division is performed any more. The use of the floating point representation `M_SQRT2` of the square root of 2 instead of `sqrt(2)` achieves even a higher reduction of necessary computations.

2. *Initialization*
   As additional options it is now possible to initialize a wavefunction either by passing a complex array and its length as parameters or by passing a pointer to a wavefunction.

3. *Operators, get, set and other trivial functions*
   The new function `clone()` returns a deep copy of a `wavefunction`, `null()` a zero-valued wavefunction of the same size. The functions `realpart()` and `imagpart()` return the element-wise real respectively imaginary part of the wavefunction, while `array()` returns the pointer to the complex array that stores the wavefunction values. The standard get and set functions `getentry(...)` and `setentry(...)` should be used rarely and carefully.

4. *Linear grid transformation*
   A rather simple but powerful function is `transform_grid_lin(...)`, allowing the change of the grid each wavefunction is based on by a piecewise linear transformation. Previously only the function `regrid(...)` has been available, allowing to add or remove orbitals, outer angular momenta and outer radial discretization points. This enables to extend or reduce the grid size, but not to change the radial discretization parameter Δr. If an initial ground state wavefunction on a defined grid is given and for real time propagation a finer grid is required or a coarser grid is sufficient, this is a serious problem. Its solution is the already mentioned `transform_grid_lin(...)` function. It uses a spline defined by the initial grid points and maps the values of this spline to the new grid. If the grid area is extended, the additional points in this region are initialized with zero, as it is the case in `regrid(...)`.

5. *Exchange-correlation potential computation functions*
   To compute the exchange-correlation potential, three types of functions

are used: the first type computes the underlying vectors Λ, Θ and Ξ (2.31)–(2.33), the second the Hartree potential multipole terms U^j (2.34)–(2.36) and the third the KLI potential monopole term V_x^{KLI} (2.37).
All these functions of all three types were investigated for errors and optimization possibilities. For the potential computation functions and `calculate_Lambda(...)` no changes were necessary. The remaining functions `calculate_Theta(...)` and `calculate_Xi(...)` were both corrected and improved and additionally the use of the `Coefficients` class was implemented for further optimization.

6. *Propagate-interface*
   Several new interfaces are added to the `propagate(...)` function. The `propagate(...)` function interfaces select one of the three private functions `do_muller...` depending on the laser polarization (linear or circular, defined by parameter `dimens`-function `grid.dimens()`) and on whether the atom has one or more electrons (parameter `hydrogenlike`). The private interfaces were changed such that the `Coefficients` and `JKtimer` classes (see Sections 4.1.1 and 4.1.4) are additionally passed and that the new `Hamop` class is used instead of the `hamop` class (see Section 4.1.3). These private interfaces can be changed without losing backwards compatibility, because they are not visible from outside the class, but the original public `propagate(...)` function interface has to be kept untouched to enable the use of the actual QPROP library in older programs (see also Section 4.1.5). These original public interfaces, as well as all newly introduced more common public function interfaces, call the most general new public interface where the private interface is selected and finally called. This allows the user to choose the best option for his application from a number of interfaces.

7. *Private propagate functions*
   (a) *do_muller_ellm(...)*
   This function computes the propagation for circular polarized laser fields that are not the main topic of this work, which concentrates on linear polarized laser fields. Therefore only a few optimizations for some constant terms have been implemented.
   (b) *do_muller_general_tddft(...)*
   As mentioned above the interface is extended by the additional parameters of `Coefficients` and `JKtimer` class. The main improvements in this function are therefore in the context of these classes. The huge amount of coefficient calculations is changed to accessing the coefficient's value in the memory via the `Coefficients` class. Before and after each step the CPU time consumption is measured and stored by the `JKtimer` class.
   (c) *do_muller_ell(...)*
   In this function that is used for hydrogen-like atoms, the same optimizations and improvements as in `do_muller_general_tddft(...)` are realized. Note, that the fact has to be taken into account that, due to the lower complexity of this function as compared to the many-electron one (only one orbital, therefore there are no orbital-orbital-interactions), the numbering of timed parts is not the same.

4.1.8 Lanczos method classes

The new `Expokit`, `Expokit_memsave`, `CSCmatrix` and `HCSCmatrix` classes implement the Lanczos method as an additional option and improvement of QPROP. These classes are described in Section 5.1.

4.1.9 Other improvements

All other and further improvements and optimizations concerning QPROP, as the introduction of a "Predictor-Corrector" step, are implemented within the new QPROP front-end XMLQPROP and described in the corresponding Chapter 3 (front-end) and Section 5.2 ("Predictor-Corrector").

4.2 Split-step Crank-Nicolson propagator

4.2.1 Boundary conditions

For the parts of the Hamilton operator linking orbitals with $\Delta\ell = 1, 2$ ($H_{\text{ang}}^{(i)}, i = 1, 2, 3$, H_{mix}, see Section 2.2.3, equations (2.47) and (2.48)–(2.52), for definition and explanation) there are only the trivial homogeneous Dirichlet boundary conditions in ℓ and no boundary conditions in space, because these terms operate on each spatial point independently. They are diagonal in r-space.
For the parts H_{mix} and H_{at} of the Hamiltonian, containing spatial derivatives, spatial boundary conditions have to be chosen carefully.
The second radial derivative is computed using the operator $-2M_2^{-1}\Delta_2$ with M_2 and Δ_2 defined as

$$M_2 := -\frac{1}{6}\begin{pmatrix} 10 & 1 & 0 & \dots & 0 \\ 1 & 10 & 1 & \ddots & \vdots \\ 0 & 1 & 10 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 1 \\ 0 & \dots & 0 & 1 & 10 \end{pmatrix}, \tag{4.1}$$

$$\Delta_2 := \frac{1}{h^2}\begin{pmatrix} -2 & 1 & 0 & \dots & 0 \\ 1 & -2 & 1 & \ddots & \vdots \\ 0 & 1 & -2 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 1 \\ 0 & \dots & 0 & 1 & -2 \end{pmatrix}. \tag{4.2}$$

This operator gives the so-called Numerov approximation to the second derivative and is of fourth order:

$$f'' = -2M_2^{-1}\Delta_2 f + \mathcal{O}(h^4). \tag{4.3}$$

At the origin ($r = 0$) the Coulomb potential for $\ell = 0$ gives

$$\Phi''_{i0m_i}(r = 0, t) = -2Z\Phi'_{i0m_i}(r = 0, t) \neq 0, \qquad \forall i. \tag{4.4}$$

By modifying the upper left elements of both M_2 and Δ_2, equation (4.4) can be realized without losing the Hermiticity of the operator:

$$(\widetilde{M_2})_{1,1} := -2\left(1 + \frac{h^2}{12}(\widetilde{\Delta_2})_{1,1}\right), \tag{4.5}$$

$$(\widetilde{\Delta_2})_{1,1} := -\frac{2}{h^2}\left(1 - \frac{Zh}{12-10Zh}\right). \tag{4.6}$$

For the other boundary $r = r_{\max} + \Delta r$ and the origin for $\ell \neq 0$, the operator is not modified, assuming a vanishing wavefunction outside $r = r_{\max}$ (at $r = r_{\max} + \Delta r$) and at the origin for $l \neq 0$.
The first derivative is computed using the Simpson operator $M_1^{-1}\Delta_1$ with M_1 and Δ_1 defined as

$$M_1 := \frac{1}{6}\begin{pmatrix} 4 & 1 & 0 & \dots & 0 \\ 1 & 4 & 1 & \ddots & \vdots \\ 0 & 1 & 4 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 1 \\ 0 & \dots & 0 & 1 & 4 \end{pmatrix}, \tag{4.7}$$

$$\Delta_1 := \frac{1}{2h}\begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ -1 & 0 & 1 & \ddots & \vdots \\ 0 & -1 & 0 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 1 \\ 0 & \dots & 0 & -1 & 0 \end{pmatrix}. \tag{4.8}$$

This operator gives the so-called Simpson approximation to the first derivative and is of fourth order:

$$f' = M_1^{-1}\Delta_1 f + \mathcal{O}(h^4). \tag{4.9}$$

By modifying the upper left and lower right elements of both M_1 and Δ_1 anti-Hermiticity of (4.9) is ensured [5]:

$$(\widetilde{M_1})_{1,1} := (\widetilde{M_1})_{n_r,n_r} := \frac{1}{6}\left(2+\sqrt{3}\right), \tag{4.10}$$

$$\begin{array}{rcl} (\widetilde{\Delta_1})_{1,1} & := & \frac{1}{2h}\left(\sqrt{3}-2\right), \\ (\widetilde{\Delta_1})_{n_r,n_r} & := & \frac{1}{2h}\left(2-\sqrt{3}\right). \end{array} \tag{4.11}$$

4.2.2 Imaginary absorbing potential

The well-understood Dirichlet boundary conditions as discussed in Section 4.2.1 may cause the problem of non-physical reflections at the outer boundary ($r = r_{\max}$). Increasing $r_{\max}$ could solve this problem only to a certain extent and increases the computational effort dramatically.
One standard solution is to introduce an imaginary absorbing potential that

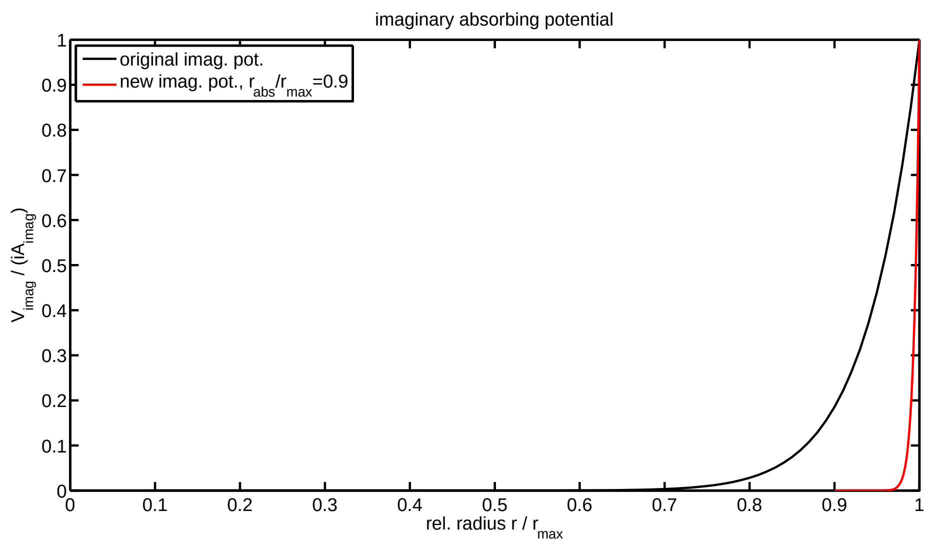

Figure 4.1: The imaginary absorbing potential

becomes strongly absorbing near $r = r_{\max}$ and has only negligible strength inside the orbitals. The transition in between has to be at least of 5th order if 4th order methods are used. The implementation in QPROP version 1.6 uses an imaginary absorbing potential defined as follows:

$$V_{\text{imag}}(r,t) = V_{\text{imag}}(r) = \mathrm{i}A_{\text{imag}} \left(\frac{r - \frac{\Delta r}{2}}{r_{\max}} \right)^{16} . \tag{4.12}$$

This potential is non-zero for all spatial points although the potential is very small for small r. For a further reduction of the influence of this imaginary absorbing potential in the inner region an additional XML-file parameter, the imaginary potential inner border `imagpot_inner_border`, also called r_{abs}, is introduced as well as the parameter `imagpot_ampl` defining the amplitude of the imaginary absorbing potential A_{imag}. The constructors and `imagpot` functions of the `Potential` subtypes are adopted, such that the new and more flexible imaginary absorbing potential is computed by

$$V_{\text{imag}}(r,t) = V_{\text{imag}}(r) = \begin{cases} 0, & r < r_{\text{abs}}, \\ \mathrm{i}A_{\text{imag}} \left(\frac{r - \frac{\Delta r}{2} - r_{\text{abs}}}{r_{\max} - r_{\text{abs}}} \right)^{16}, & r_{\text{abs}} \le r \le r_{\max}. \end{cases} \tag{4.13}$$

The function stays smooth enough and has no direct influence for $r < r_{\text{abs}}$. For $r_{\text{abs}} = 0$ the originally used potential is obtained because then equation (4.13) is equivalent to equation (4.12). For illustration the originally used imaginary absorbing potential and the new version (with $\frac{r_{\text{abs}}}{r_{\max}} = 0.9$) are shown in Figure 4.1.

4.2.3 Crank-Nicolson propagator

For an explicitly time-dependent Hamiltonian, such that $\partial_t \mathbf{\Phi}(t) = H(t)\mathbf{\Phi}(t)$, the exact propagator $U(t+h,h)$, defined as

$$\mathbf{\Phi}(t+h) = U(t+h,t)\mathbf{\Phi}(t), \tag{4.14}$$

can be approximated (of second order for a linear, time-dependent Hamiltonian) by

$$U(t+h,t) \approx \exp\left(-\mathrm{i}hH\left(t+\frac{h}{2}\right)\right). \tag{4.15}$$

The implicit time dependence of the Hamiltonian (it also depends on the Kohn-Sham orbitals) does not allow to evaluate all parts of the Hamiltonian (2.47) at the time point $t+\frac{h}{2}$ in our case. Some can only be evaluated at t, several are evaluated at $t+h$ to optimize the propagation for more than one time step, where the right-hand terms of the previous time step meet the left-hand terms of the current time step. In summary, the propagator is approximated by the following product

$$\begin{aligned} U_{\mathrm{split}}(t+h,t) &= \textstyle\prod_{\underline{\ell}=3}^{L} \exp\left(-\mathrm{i}\frac{h}{2}H^{(3)}_{\mathrm{ang},L-\underline{\ell}}(t)\right) \times \\ &\times \textstyle\prod_{\underline{\ell}=2}^{L} \exp\left(-\mathrm{i}\frac{h}{2}H^{(1+2)}_{\mathrm{ang},L-\underline{\ell}}(t)\right) \exp\left(-\mathrm{i}\frac{h}{2}H_{\mathrm{mix},L-\underline{\ell}}(t)\right) \times \\ &\times \exp\left(-\mathrm{i}\frac{h}{2}H_{\mathrm{at}}(t)\right) \times \\ &\times \textstyle\prod_{\overline{\ell}=0}^{L-2} \exp\left(-\mathrm{i}\frac{h}{2}H_{\mathrm{mix},\overline{\ell}}(t+h)\right) \exp\left(-\mathrm{i}\frac{h}{2}H^{(1+2)}_{\mathrm{ang},\overline{\ell}}(t+h^*)\right) \times \\ &\times \textstyle\prod_{\overline{\ell}=0}^{L-3} \exp\left(-\mathrm{i}\frac{h}{2}H^{(3)}_{\mathrm{ang},\overline{\ell}}(t)\right) \end{aligned} \tag{4.16}$$

with $t+h^* = t+h$ as argument in $A(t)$ and $E(t)$ and $t+h^* = t$ as argument in $V^1_{\mathrm{ee}}(t)$. It is clear, that for these evaluation points, being different to those in equation (4.15), order 2 cannot be expected any more. This is also shown by the numerical results, Chapter 6, and can be improved by a "Predictor-Corrector" step (see Section 5.2).
The next step is to approximate the exponentials in equation (4.16) by unitary Crank-Nicolson (abbreviated C-N) approximants of second order

$$U_{\mathrm{C-N}}(t+h,t) = \left(\mathbf{1} + \mathrm{i}\frac{h}{2}H\right)^{-1}\left(\mathbf{1} - \mathrm{i}\frac{h}{2}H\right) = \exp(-\mathrm{i}hH) + O(h^3). \tag{4.17}$$

With

$$R = \left(\mathbf{1} + \mathrm{i}\frac{h}{4}H^{(1+2)}_{\mathrm{ang}}\right)^{-1}\left(\mathbf{1} - \mathrm{i}\frac{h}{4}H^{(1+2)}_{\mathrm{ang}}\right), \tag{4.18}$$

$$X_\pm = \mathbf{1} \pm \mathrm{i}\frac{h}{4}H_{\mathrm{mix}}, \tag{4.19}$$

$$Q_\pm = \mathbf{1} \pm \mathrm{i}\frac{h}{2}H_{\mathrm{at}}, \tag{4.20}$$

$$Z = \left(\mathbf{1} + \mathrm{i}\frac{h}{4}H^{(3)}_{\mathrm{ang}}\right)^{-1}\left(\mathbf{1} - \mathrm{i}\frac{h}{4}H^{(3)}_{\mathrm{ang}}\right) \tag{4.21}$$

and equation (4.16) the C-N approximation (4.17) becomes

$$U_{\mathrm{C-N}}(t+h,t) = \prod_{\underline{\ell}} Z \prod_{\underline{\ell}} \left(RX_+^{-1}X_-\right) Q_+^{-1}Q_- \prod_{\overline{\ell}} \left(X_+^{-1}X_-R\right) \prod_{\overline{\ell}} Z. \tag{4.22}$$

The X and Q matrix products are further simplified [6]:

$$X_+^{-1}X_- = B^\mathsf{T}Y_+^{-1}Y_-B, \tag{4.23}$$

$$Q_+^{-1}Q_- = W_+^{-1}W_-, \tag{4.24}$$

finally leading to the representation (4.25) of the C-N propagator as it is applied internally in the Qprop code:

$$U_{\mathrm{C-N}}(t+h,t) = \prod_{\underline{\ell}} Z \prod_{\underline{\ell}} \left(RB^\mathsf{T}Y_+^{-1}Y_-B\right) W_+^{-1}W_- \prod_{\overline{\ell}} \left(B^\mathsf{T}Y_+^{-1}Y_-BR\right) \prod_{\overline{\ell}} Z. \tag{4.25}$$

Chapter 5

Lanczos method and "Predictor-Corrector" step

5.1 Implementation of Lanczos method

For a Hamiltonian H given as a matrix $H(t)$ and the wavefunction given as complex vector valued function $\mathbf{\Phi}(t)$, Lanczos method computes

$$\mathbf{\Phi}(t+h) = e^{-ihH\left(t+\frac{h^*}{2}\right)}\mathbf{\Phi}(t) \tag{5.1}$$

with $t+\frac{h^*}{2} = t+\frac{h}{2}$ as argument in time-dependent parts of the Hamiltonian $H(t)$ that do not depend on the wavefunction (like $A(t)$ and $E(t)$), while $t+\frac{h^*}{2} = t$ as argument in all parts of the Hamiltonian $H(t)$ that depend on the wavefunction (V_{ee}).

5.1.1 Theoretical background of the Lanczos method

Lanczos method is a Krylov subspace method that does not compute the full matrix exponential $\exp(-\mathrm{i}hH)$, but only its action $\exp(-\mathrm{i}hH)\,v$ on a vector v. To describe the method, first the Krylov subspace and the Lanczos basis are defined:

Definition 5.1. Krylov subspace
Let $H \in \mathbb{C}^{N\times N}$ be an Hermitian matrix, and let $v \in \mathbb{C}^N\backslash\{\mathbf{0}\}$ be a vector. The m^{th} Krylov subspace of $\mathbb{C}^N$ with respect to H and v is

$$\mathcal{K}_m(H,v) = \operatorname{span}\left(v, Hv, H^2v, \ldots, H^{m-1}v\right), \tag{5.2}$$

that is the space of all polynomials of H up to degree $m-1$ acting on the vector v.

Definition 5.2. Lanczos basis
The ordered Lanczos basis $(v_1, v_2, \ldots, v_m)$ is the orthonormal basis of the m^{th} Krylov subspace of $\mathbb{C}^N$ with respect to H and v built by Gram-Schmidt orthogonalization:

$$\begin{aligned} v_1 &:= \frac{v}{\|v\|}, \\ v_{k+1} &:= \frac{Hv_k - \sum_{j=1}^{k}\tau_{jk}v_j}{\tau_{k+1,k}}, \quad k = 1, 2, \ldots, m-1, \end{aligned} \tag{5.3}$$

with

$$\begin{array}{rcll} \tau_{jk} &=& \overline{v}_j^{\mathsf{T}} H v_k, & 1 \le j \le k, \quad k = 2,3,\ldots,m \\ \tau_{k+1,k} &=& \|Hv_k - \sum_{j=1}^{k} \tau_{jk} v_j\|, & \qquad\qquad k = 1,2,\ldots,m-1, \\ \tau_{jk} &=& 0, & j > k+1, \quad k = 1,2,\ldots,m. \end{array} \tag{5.4}$$

If $\tau_{k+1,k} = 0$ for $k = k_0 := \min\{\kappa | \tau_{\kappa+1,\kappa} = 0\} < m-1$ the process terminates ("happy breakdown") and the Lanczos basis consists only of the first k_0 basis vectors $(v_1, v_2, \ldots, v_{k_0})$ and the dimension of the m^{th} Krylov subspace of $\mathbb{C}^N$ with respect to H and v is equal to k_0.

The last statement in the Definition 5.2 above yields the following lemma 5.1:

Lemma 5.1. "happy breakdown"
If $\tau_{k+1,k} = 0$ for $k = k_0 := \min\{\kappa | \tau_{\kappa+1,\kappa} = 0\} < m-1$ then:

- *The process terminates ("happy breakdown")*
- *The Lanczos basis consists only of the first k_0 basis vectors $(v_1, v_2, \ldots, v_{k_0})$.*
- *The dimension of the m^{th} Krylov subspace of $\mathbb{C}^N$ with respect to H and v is equal to k_0.*
- *The Lanczos method (approximation* (5.10)*) is exact for analytic functions $f(H)$ of the Hamiltonian, in particular for the exponential function* (5.11)*.*

Subsequently, we will use the following definitions:

Definition 5.3. V_m, T_m, e_j

$$\begin{array}{rcll} V_m &:=& (v_1 | \ldots | v_m) & \in \mathbb{C}^{N\times m}, \\ T_m &:=& (\tau_{jk}) & \in \mathbb{C}^{m\times m}, \\ e_j^{\mathsf{T}} &:=& (0,\ldots,0,1,0,\ldots,0), & \textit{the } j\textit{-th unit vector in } \mathbb{C}^m. \end{array} \tag{5.5}$$

Going one step further than m the following equation holds:

$$HV_m = V_m T_m + \tau_{m+1,m} v_{m+1} e_m^{\mathsf{T}}, \tag{5.6}$$

and this equation also holds if no further step can be performed, because then the terms containing indices $m+1$ are assumed to be equal to 0.
Multiplication of equation (5.6) by V_m^{H} from the left side implies, due to orthonormality of the Lanczos basis,

$$T_m = \overline{V}_m^{\mathsf{T}} H V_m. \tag{5.7}$$

Equation (5.7) shows that T_m is an Hermitian matrix. Because of this property and the third equation (5.4), T_m is also a tridiagonal matrix.
Thus the Gram-Schmidt orthogonalization iteration equations (5.3) simplify to

$$\begin{array}{rcll} v_1 &=& \frac{v}{\|v\|}, & \\ v_{k+1} &=& \frac{Hv_k - \sum_{j=k-1}^{k} \tau_{jk} v_j}{\tau_{k+1,k}}, & k = 1,2,\ldots,m-1. \end{array} \tag{5.8}$$

Definitions 5.1–5.3 and equations (5.2)–(5.8) already allow to implement an algorithm as described in Section 5.1.3 to compute V_m and T_m in the standard

version of Lanczos iteration. For larger m the practical implementation may suffer from error propagation and the loss of orthogonality due to rounding errors. This may be improved by (selective) reorthogonalization (see [11]), but is not applied here, because m is assumed to be small.
Definitions 5.1–5.3 and equations (5.2)–(5.7) also hold for non-Hermitian matrices H and are implemented by the Arnoldi process (see Section 5.1.3).
Based on equations (5.6) and (5.7) the following lemma is easily proven [12]:

Lemma 5.2. *Let H be an Hermitian matrix and v a vector of unit norm.*
(a) If all eigenvalues $(\lambda_j, j = 1, \ldots, N)$ of H are in the interval $[a, b]$, then so are those of T_m.
(b) For every polynomial p_{m-1} of degree at most $m - 1$, it holds that

$$p_{m-1}(H)v = V_m p_{m-1}(T_m)e_1. \tag{5.9}$$

Lemma 5.2 and the diagonalization of $H = \overline{U}^{\mathsf{T}}\operatorname{diag}(\lambda_j)U$ in combination with a complex valued function $f : [a, b] \to \mathbb{C}$ gives an approximation to $f(H) = \overline{U}^{\mathsf{T}}\operatorname{diag}(f(\lambda_j))U$:

$$f(H)v \approx V_m f(T_m)e_1. \tag{5.10}$$

For the matrix exponential this approximation reads

$$e^{-\mathrm{i}hH}v \approx V_m e^{-\mathrm{i}hT_m}e_1. \tag{5.11}$$

Using the scaling and squaring method in combination with Padé approximation for the exponential $e^{-\mathrm{i}hT_m}$ is the first choice for the implementation of this approximation (see [13], Sections 5.1.3 and 5.1.4).
The following optimality results and error bounds for the Lanczos method are also given in [12]:

Theorem 5.3. Optimality of the Lanczos method
Let f be a complex-valued function defined on an interval $[a, b]$ that contains the eigenvalues of the Hermitian matrix H, and let v be a vector of unit norm. Then, the error of the Lanczos approximation to $f(H)v$ is bounded by

$$\|V_m f(T_m)e_1 - f(H)v\| \leq 2 \inf_{p_{m-1}\in\mathbb{P}_{m-1}} \max_{x\in[a,b]} |p_{m-1}(x) - f(x)|, \tag{5.12}$$

where the infimum is taken over all polynomials of degree at most $m - 1$.

Theorem 5.4. Eventual superlinear error decay
Let H be an Hermitian matrix all of whose eigenvalues are in the interval $[a, b]$, and let v be a vector of unit Euclidean norm. Then, the error of the Lanczos method (5.11) is bounded by

$$\|V_m e^{-\mathrm{i}hT_m}e_1 - e^{-\mathrm{i}hH}v\| \leq 8\left(e^{1-\left(\frac{\omega}{2m}\right)^2}\frac{\omega}{2m}\right)^m, \qquad m \geq \omega \tag{5.13}$$

with $\omega = \frac{h(b-a)}{2}$.

5.1.2 Matrix of the Hamiltonian for the Kohn-Sham-equations

Before the Lanczos method can be applied, the matrix of the Hamiltonian is assembled. While the Hamiltonian is split into several parts for the standard C-N method (see Section 4.2), here the full matrix is assembled and stored. Because storing a full $(n_x \times n_y \times n_z)^2$ matrix would exceed memory capacities by several orders of magnitude, two improvements (three if the Hamiltonian is Hermitian) are made:

1. The Hamiltonian is diagonal in the Kohn-Sham orbital space and therefore acting independently on each Kohn-Sham orbital. This fact reduces the memory requirement by a factor of n_z^{-1}, because the n_z Hamiltonian matrices of dimension $(n_x \times n_y)^2$ can be applied consecutively, each on its corresponding orbital, and not the whole Hamiltonian matrix of dimension $(n_x \times n_y \times n_z)^2$ has to be in memory.

2. The n_z Hamiltonian matrices are block-quint-diagonal each (or less, if $n_y < 3$). The ℓ-Blocks are again only quint-diagonal, tridiagonal or diagonal. Using a sparse matrix format, like the CSC (<u>c</u>ompressed <u>s</u>parse <u>c</u>olumn) format chosen here, the memory requirement of the $(n_x \times n_y)^2$ dimensional matrix is reduced to $n_{\text{diagonals}} \times (n_x \times n_y)$ and therefore depends only linearly and no longer quadratically on the discretization parameters.

3. Normally the Hamiltonian is Hermitian. The non-Hermitian imaginary absorbing potential (as described in Section 4.2.2) is applied separately in the Lanczos method mode to preserve the Hermiticity of the Hamiltonian. Thus, only the diagonal and the elements below have to be saved in the CSC format (leading to the HCSC (<u>H</u>ermitian <u>CSC</u>) format) reducing the memory requirement further by nearly one half.

The three-level structure of the Hamiltonian matrix as roughly described above will now be stated more precisely.
The matrix of the Hamiltonian H is a block-diagonal matrix of n_z blocks H_i of size $(n_x \times n_y)^2$,

$$H = \begin{pmatrix} H_1 & 0 & \cdots & 0 \\ 0 & H_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & H_{n_z} \end{pmatrix}. \tag{5.14}$$

Each H_i block acts on the corresponding i^{th} Kohn-Sham orbital and consists of several $D_{\ell_1\ell_2}$-blocks forming a block-quint-diagonal matrix:

$$H_i = \begin{pmatrix} D_{00} & D_{01} & D_{02} & 0 & \cdots & \cdots & 0 \\ D_{10} & D_{11} & D_{12} & D_{13} & \ddots & \ddots & \vdots \\ D_{20} & D_{21} & D_{22} & D_{23} & \ddots & \ddots & \vdots \\ 0 & D_{31} & D_{32} & D_{33} & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & D_{n_y-2,n_y-1} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & D_{n_y-2,n_y-1} \\ 0 & \cdots & \cdots & 0 & D_{n_y-1,n_y-3} & D_{n_y-1,n_y-2} & D_{n_y-1,n_y-1} \end{pmatrix}, \tag{5.15}$$

with each D-block depending on the Kohn-Sham-orbital index i

$$D_{\ell_1\ell_2} = \left(D_{\ell_1\ell_2}\right)_i . \tag{5.16}$$

These D-blocks couple the different ℓ-states within each Kohn-Sham-orbital and there are three types of D-blocks depending on

$$|\Delta\ell| = |\ell_2 - \ell_1| \in \{0, 1, 2\} . \tag{5.17}$$

$|\Delta\ell| = 0$ corresponds with the blocks on the diagonal

$$D_{\ell\ell} = H_{\text{at}} = -\frac{1}{2}\partial_r^2 + V_{\text{eff}}^{\ell} + V_{\text{ee}}^{0} + p_{\ell m_i} V_{\text{ee}}^{2}, \tag{5.18}$$

with all variables and operators in discretized form and vectors as diagonal matrices. In detail this means that

$$D_{\ell\ell}^{\text{e2}} = \begin{pmatrix} \begin{pmatrix} \frac{1}{h_x^2} + V_{\text{eff}}^{\ell}(r_1) \\ +V_{\text{ee}}^{0}(r_1) \\ +p_{\ell m_i} V_{\text{ee}}^{2}(r_1) \end{pmatrix} & -\frac{1}{2h_x^2} & 0 & \cdots & 0 \\ -\frac{1}{2h_x^2} & \ddots & -\frac{1}{2h_x^2} & \ddots & \vdots \\ 0 & -\frac{1}{2h_x^2} & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -\frac{1}{2h_x^2} \\ 0 & \cdots & 0 & -\frac{1}{2h_x^2} & \begin{pmatrix} \frac{1}{h_x^2} + V_{\text{eff}}^{\ell}(r_{n_x}) \\ +V_{\text{ee}}^{0}(r_{n_x}) \\ +p_{\ell m_i} V_{\text{ee}}^{2}(r_{n_x}) \end{pmatrix} \end{pmatrix} \tag{5.19}$$

for the second order approximation of the differential operator and

$$D^{\mathrm{e}4}_{\ell\ell} = \begin{pmatrix} \begin{pmatrix} \frac{5}{4h_x^2} + V^{\ell}_{\mathrm{eff}}(r_1) \\ +V^0_{\mathrm{ee}}(r_1) \\ +p_{\ell m_i} V^2_{\mathrm{ee}}(r_1) \end{pmatrix} & -\frac{2}{3h_x^2} & \frac{1}{24h_x^2} & 0 & \cdots \\ -\frac{2}{3h_x^2} & \begin{pmatrix} \frac{5}{4h_x^2} + V^{\ell}_{\mathrm{eff}}(r_2) \\ +V^0_{\mathrm{ee}}(r_2) \\ +p_{\ell m_i} V^2_{\mathrm{ee}}(r_2) \end{pmatrix} & -\frac{2}{3h_x^2} & \frac{1}{24h_x^2} & \ddots \\ \frac{1}{24h_x^2} & -\frac{2}{3h_x^2} & \ddots & -\frac{2}{3h_x^2} & \ddots \\ 0 & \frac{1}{24h_x^2} & -\frac{2}{3h_x^2} & \ddots & \ddots \\ \vdots & \ddots & \ddots & \ddots & \ddots \end{pmatrix} \tag{5.20}$$

for the fourth order approximation of the differential operator.

On the super- $(\Delta\ell = 1)$ and sub- $(\Delta\ell = -1)$ -diagonal the D-blocks are

$$D_{\ell,\ell\pm1} = -\mathrm{i}A(t)c_{\ell m_i}\partial_r + \left(r_{i_x} E(t) c_{\ell m_i} + V^1_{\mathrm{ee}}(r_{i_x}) c_{\ell m_i} \mp \mathrm{i}\frac{A(t)t_{\ell m_i}}{r_{i_x}} \right)_{i_x=1,\ldots,n_x}, \tag{5.21}$$

with all variables and operators in discretized form and vectors as diagonal matrices. In detail this means that

$$D^{\mathrm{e}2}_{\ell,\ell\pm1} = \begin{pmatrix} \begin{pmatrix} r_1 E(t) c_{\ell m_i} \\ +V^1_{\mathrm{ee}}(r_1) c_{\ell m_i} \\ \mp\mathrm{i}\frac{A(t)t_{\ell m_i}}{r_1} \end{pmatrix} & -\mathrm{i}\frac{c_{\ell m_i}}{2h_x}A(t) & 0 & \cdots & 0 \\ \mathrm{i}\frac{c_{\ell m_i}}{2h_x}A(t) & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -\mathrm{i}\frac{c_{\ell m_i}}{2h_x}A(t) \\ 0 & \cdots & 0 & \mathrm{i}\frac{c_{\ell m_i}}{2h_x}A(t) & \begin{pmatrix} r_{n_x} E(t c_{\ell m_i}) \\ +V^1_{\mathrm{ee}}(r_{n_x}) c_{\ell m_i} \\ \mp\mathrm{i}\frac{A(t)t_{\ell m_i}}{r_{n_x}} \end{pmatrix} \end{pmatrix} \tag{5.22}$$

for the second order approximation of the differential operator and

$$D^{\mathrm{e}4}_{\ell,\ell\pm1} = \begin{pmatrix} \begin{pmatrix} r_1 E(t) c_{\ell m_i} \\ +V^1_{\mathrm{ee}}(r_1) c_{\ell m_i} \\ \mp \mathrm{i}\frac{A(t) t_{\ell m_i}}{r_1} \end{pmatrix} & -\mathrm{i}\frac{2c_{\ell m_i}}{3h_x}A(t) & \mathrm{i}\frac{c_{\ell m_i}}{12h_x}A(t) & 0 & \cdots \\ \mathrm{i}\frac{2c_{\ell m_i}}{3h_x}A(t) & \begin{pmatrix} r_2 E(t) c_{\ell m_i} \\ +V^1_{\mathrm{ee}}(r_2) c_{\ell m_i} \\ \mp \mathrm{i}\frac{A(t) t_{\ell m_i}}{r_2} \end{pmatrix} & -\mathrm{i}\frac{2c_{\ell m_i}}{3h_x}A(t) & \mathrm{i}\frac{c_{\ell m_i}}{12h_x}A(t) & \ddots \\ -\mathrm{i}\frac{c_{\ell m_i}}{12h_x}A(t) & \mathrm{i}\frac{2c_{\ell m_i}}{3h_x}A(t) & \ddots & -\mathrm{i}\frac{2c_{\ell m_i}}{3h_x}A(t) & \ddots \\ 0 & -\mathrm{i}\frac{c_{\ell m_i}}{12h_x}A(t) & \mathrm{i}\frac{2c_{\ell m_i}}{3h_x}A(t) & \ddots & \ddots \\ \vdots & \ddots & \ddots & \ddots & \ddots \end{pmatrix} \tag{5.23}$$

for the fourth order approximation of the differential operator.
The blocks on the 2 and -2 diagonal have a simpler structure, they are diagonal and contain only the term V^2_{ee},

$$D_{\ell,\ell\pm2} = q_{\ell m_i} V^2_{\mathrm{ee}} \tag{5.24}$$

with all variables and operators in discretized form and vectors as diagonal matrices. In detail this means that

$$D_{\ell,\ell\pm2} = \begin{pmatrix} q_{\ell m_i} V^2_{\mathrm{ee}}(r_1) & 0 & \cdots & 0 \\ 0 & q_{\ell m_i} V^2_{\mathrm{ee}}(r_2) & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & q_{\ell m_i} V^2_{\mathrm{ee}}(r_{n_x}) \end{pmatrix} . \tag{5.25}$$

Naturally this expression does not depend on the differential operator approximation because this term is derivative-free.

A further extension of the matrix of the Hamiltonian presented above is straight forward. Higher exchange-correlation potential multipole terms, different differential operator approximations and several electromagnetic field gauges can be included and implemented in the HCSC and CSC matrix class constructor functions. In contrast to the standard C-N method no changes in the `propagate(...)` algorithm are necessary.

5.1.3 Algorithm and implementation details

The implementation of the Lanczos method for computing the matrix exponential is based on EXPOKIT, a software package for computing matrix exponentials [10]. The expokit package is written in FORTRAN using a special interface for the main subroutine calls such that the package could not be simply included into the C++ based QPROP library. To change only the interface is no option, because then the matrix classes (`CSCmatrix` and `HCSCmatrix`) could not be used directly for matrix-vector computations. On the other hand the effort to rewrite all required expokit parts in C++ would be too high and would contradict the goal to use library functions if possible.

In this situation the best option is to rewrite the main algorithm in C++ tailored to the requirements of the QPROP package and to use the expokit, BLAS and LAPACK functions that are also used by the original expokit implementation. An additional advantage of this approach is that the new C++ source code is much more readable than the old Fortran implementation that uses lots of `GOTO` statements. The Lanczos and Arnoldi algorithms are now implemented in the `Expokit` class, the Lanczos version with lower memory requirements in the `Expokit_memsave` class.
To propagate the wavefunction, the `Expokit.propagate(...)` function is called as the `wavefunction.propagate(...)` in the standard C-N version, and the same holds for the memory saving version (`Expokit_memsave.propagate(...)`). For both versions of the Lanczos algorithm the `propagate` function works as follows.

Algorithm 5.1. `Expokit.propagate(...)`

```
 some initialization
begin for loop for $i_z$ over all Kohn-Sham orbitals
    initialize $p$ as the $i_z^{\text{th}}$ Kohn-Sham orbital wavefunction
    initialize the corresponding Hamiltonian matrix $H_i$
    call zexpv(p) to compute $\exp(-\mathrm{i}hH_i)p$ using Lanczos/Arnoldi method
    write $p$ into the wavefunction as new Kohn-Sham orbital data
end for loop
```

The function `zexpv(p)` implements the Lanczos/Arnoldi process similarly to the Algorithm 3.1 in [10]. The only significant difference is that similarly Algorithm 5.2 implements different methods that are chosen depending on the Lanczos parameters (hermitian/non-hermitian and adaptive/fixed step size).

Algorithm 5.2. `Expokit.zexpv(complex* v)`

```
 some initialization
begin loop until $t_{\text{now}} = t_{\text{out}}$, mxstep repeats at maximum
    compute $\beta$ and the first basis vector;
    if(adaptive)
       if(hermitian)
          Lanczos loop for adaptive step computes $T_m$, $V_m$;
       else
          Arnoldi loop for adaptive step computes $T_m$, $V_m$;
       end if
    else
       if(hermitian)
          Lanczos loop for fixed step computes $T_m$, $V_m$;
       else
          Arnoldi loop for fixed step computes $T_m$, $V_m$;
       end if
    end if
    $T_m * = -\mathrm{i} * \mathrm{sgn}(h)$;
    if(adaptive)
       begin do loop
          compute the adaptive stepsize $\tau$;
          $f = \exp(\tau T_m)e_1$;
          compute the local error estimate err_loc;
```

```
        end do loop, while err_loc$> \delta * \tau *$ tol, mxreject repeats at maximum
        $v = \beta * V_m * f$;
    else
        $\tau = t_{\text{out}} - t_{\text{now}}$;
        $f = \exp(\tau T_m)e_1$;
        $v = \beta * V_m * f$;
    end if
    $t_{\text{now}} += \tau$;
end loop
```

Algorithm 5.2 is nearly the same in the `Expokit_memsave` class, the differences are that only Lanczos is possible (no Arnoldi), only two column vectors of V_m are stored at the same time (plus an additional temporary vector) and that the Lanczos loop has to be performed a second time (without computing T_m again) when v is computed.
Now the Lanczos and Arnoldi loops will be presented in detail, while the adaptive time step computation algorithm as well as the Padé approximation are discussed in Section 5.1.4.

Algorithm 5.3. Lanczos loop

```
 input: $H$, $v$, output: $\beta$, $V_m$, $T_m$;
$\beta = \|v\|$;
$V_m(0,:) = \frac{v}{\beta}$;
for($j = 0$;$j < m$;$j++$){
    $p = H * V_m(j,:)$;
    if($j > 0$) $p -= T_m(j,j-1) * V_m(j-1,:)$;
    $T_m(j,j) = V_m(j,:)^{\mathsf{H}} * p$;
    $p -= T_m(j,j) * V_m(j,:)$;
    if($j+1 < m$){
        $T_m(j+1,j) = \|p\|$;
        $T_m(j,j+1) = T_m(j+1,j)^{\mathsf{H}}$;
        if($|T_m(j+1,j)| <$ tol $* \|H\|$){
            output: "happy breakdown :)";
            hbd=true;
            $T_m(j+1,j) = 0$;
            $T_m(j,j+1) = 0$;
            break;
        }
        $V_m(j+1,:) = \frac{p}{T_m(j+1,j)}$;
    }
}
```

Algorithm 5.4. Arnoldi loop

```
 input: $H$, $v$, output: $\beta$, $V_m$, $T_m$;
$\beta = \|v\|$;
$V_m(0,:) = \frac{v}{\beta}$;
for($j = 0$;$j < m$;$j++$){
    $p = H * V_m(j,:)$;
    for($i = 0$;$i < j$;$i++$){
        $T_m(i,j) = V_m(i,:)^{\mathsf{H}} * p$;
        $p- = T_m(i,j) * V_m(i,:)$;
    }
    if($j+1 < m$){
        $T_m(j+1,j) = \|p\|$;
        if($|T_m(j+1,j)| <$ tol $* \|H\|$){
            output: ”happy breakdown :)”;
            hbd=true;
            $T_m(j+1,j) = 0$;
            $T_m(j,j+1) = 0$;
            break;
        }
        $V_m(j+1,:) = \frac{p}{T_m(j+1,j)}$;
    }
}
```

5.1.4 Adaptive time stepping and Padé approximation

The algorithms and source code details for the adaptive time stepping can be found in Appendix B.3.1. The first step size depends mainly on the norm of the matrix of the Hamiltonian $\|H\|$, while each consecutive step size depends mainly on the previous one, the local error estimate `err_loc` and a shrinking factor (parameter `lanczos_gamma`). In both cases the tolerance `tol` and the dimension of the Krylov subspace play also an important role.
The step size τ is then used to compute the matrix exponential $\exp(\tau T_m)$ with the scaling and squaring method using Padé approximation. The matrix exponential is scaled to $\exp(\frac{\tau T_m}{2^s})$, with 2^s of the same order of magnitude as $\|\tau T_m\|$. The scaled exponential is approximated by Padé approximants (rational functions) and then the approximation is squared s times to get an approximation of the original exponential. A detailed review of this method can be found in [13], for example.
Finally, the local error estimate is computed to decide whether the Padé approximation's accuracy is sufficient or if the step size has to be further reduced. For the source code and details see Appendix B.3.2.

5.2 Introduction of a “Predictor-Corrector” step

As already mentioned in Section 4.2.3, the approximation (4.16) cannot be expected to be of second order, because the time evaluation points do not correspond to a half time step as proposed in equation (4.15). The numerical results in Chapter 6 show that the approximation is only of first order because of this choice of evaluation points. In contrast to this, the next approximation step by the C-N propagator is of second order, for Lanczos method even higher order can be achieved. Thus, one order of the algorithm is lost by the evaluation of the implicitly time dependent Hamiltonian, and it was one of the main goals of this work to remove this problem and gain one order of magnitude.
As equation (4.15) shows, the Hamiltonian should be evaluated at $t + \frac{h}{2}$ instead of t. The problem is now that the Hamiltonian is not only depending explicitly on time, but does also depend on the Kohn-Sham orbitals via the exchange-correlation potential. So the Hamiltonian at $t + \frac{h}{2}$ cannot be computed directly

or exactly, but it can be approximated. First, it is assumed that the Hamiltonian varies only slowly compared to the time step size. Then

$$H\left(t+\frac{h}{2}\right)=\frac{H(t)+H(t+h)}{2}+O(h^3) \tag{5.26}$$

holds for such a Hamiltonian H. But $H(t+h)$ is also not explicitly available and therefore an additional approximation has to be made. Therefore the propagation is split into two parts, where each part has the same effort as the standard propagation algorithm of first order. In the first part the Kohn-Sham orbitals at $t+h$ are computed using the standard algorithm

$$\tilde{\mathbf{\Phi}}(t+h)=U_{\text{num}}\left(t+h,t,V_{\text{ee}}(\mathbf{\Phi}(t))\right)\mathbf{\Phi}(t) \tag{5.27}$$

where $U_{\text{num}}(t+h,t,V_{\text{ee}}(\mathbf{\Phi}(t)))$ is the numerical implementation of the C-N propagator or Lanczos method with all implicit terms depending on $\mathbf{\Phi}(t)$, this is the so-called "Predictor" step. Then, in the second part, the following "Corrector" step is performed:

$$\mathbf{\Phi}(t+h)=U_{\text{num}}\left(t+h,t,\frac{V_{\text{ee}}(\mathbf{\Phi}(t))+V_{\text{ee}}(\tilde{\mathbf{\Phi}}(t+h))}{2}\right)\mathbf{\Phi}(t), \tag{5.28}$$

where all implicit terms are approximated by the mean value of their approximations at t and $t+h$. The explicitly time dependent terms are evaluated as before, at t and $t+h$ for the C-N and at $t+\frac{h}{2}$ for the Lanczos method propagator. Using this "Predictor-Corrector" step approximation, the numerical results (in Chapter 6) show that the algorithm is of second order in time at an approximately doubled numerical effort.

Chapter 6

Numerical results

In this chapter the computational and numerical properties of QPROP are analysed. The main focus will be on the order with respect to the discretization parameters, on the comparison of the standard Crank-Nicolson (C-N) propagator to the Lanczos method (Lanczos$_2$ for second order and Lanczos$_4$ for fourth order spatial differential operator approximation) and on the advantages of the "Predictor-Corrector" (PC) step.
In general one can expect that the accuracy and numerical effort of the algorithms grow (and the errors decrease) with increasing $r_{\max}$, n_r and $L = \ell_{\max}$ and decreasing Δr and $\Delta t = h$. In the following, the concept of order will be used to refer to such dependencies, where it will be clear from the context whether the asymptotics are meant with respect to increase or decrease of a certain quantity.
All relative (error) quantities are dimension-less by definition ($\epsilon = \frac{|E_{\text{num}} - E_{\text{exact}}|}{|E_{\text{exact}}|}$ for the relative energy error) and the "exact" value is the result of the same numerical computation at superior accuracy (finer grid) if no exact value is explicitly available. All absolute quantities are of dimension atomic units, except the computation/CPU time that is measured in seconds. This allows to omit the units in this Chapter because they should be clear from the context unless stated otherwise.

6.1 Computation time and discretization parameters

The computation time depends linearly on both the space discretization parameter Δr and the time discretization parameter $\Delta t = h$, as shown in Figures 6.1–6.4.
Additionally the numerical effort depends also linearly on the number of orbitals N (number of possibly occupied orbitals).
The dependence on the number of angular momenta L (number of possible quantum numbers ℓ) is not as strict, especially for L lower than 3 some exchange and correlation effects between orbitals with $\Delta\ell = \pm 2$ (and $\Delta\ell = \pm 1$ for $L < 2$) do not occur for the applied approximation (2.10). For a neon atom, where L has to be at least equal to 2, the almost linear dependence is shown in Figure 6.5.

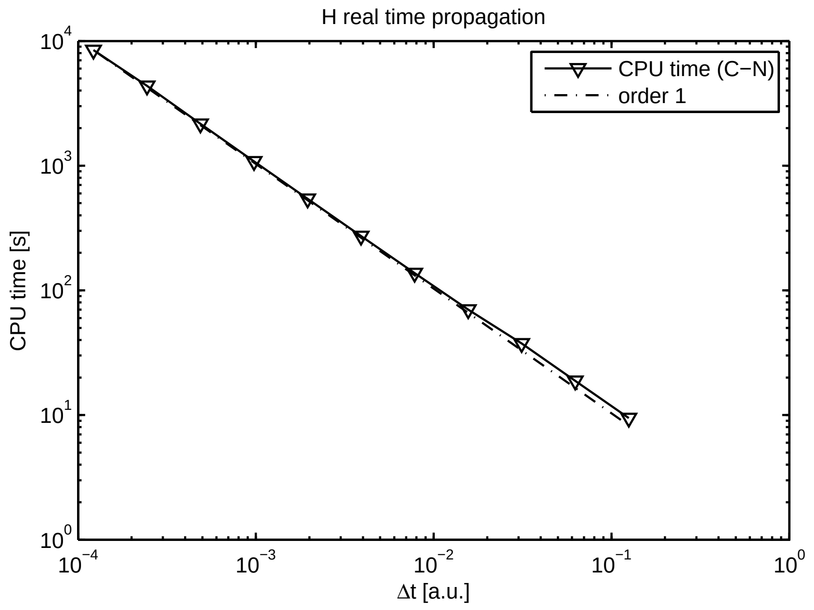

Figure 6.1: Computation time for hydrogen vs. $\Delta t = h$
Real time propagation of hydrogen, C-N propagator, $L = 4$, $r_{\max} = 100$, $\Delta r = \frac{1}{200}$.

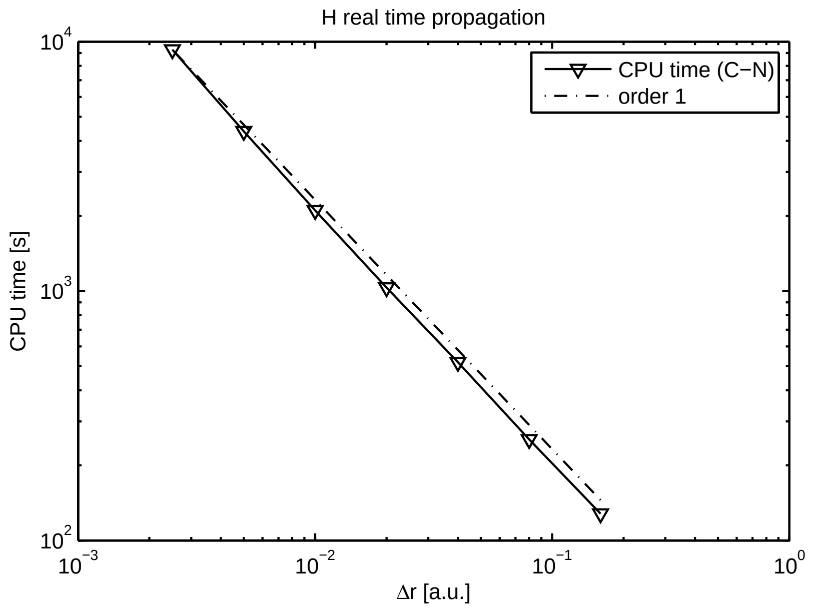

Figure 6.2: Computation time for hydrogen vs. Δr
Real time propagation of hydrogen, C-N propagator, $L = 4$, $r_{\max} = 100$, $\Delta t = \frac{1}{4096}$.

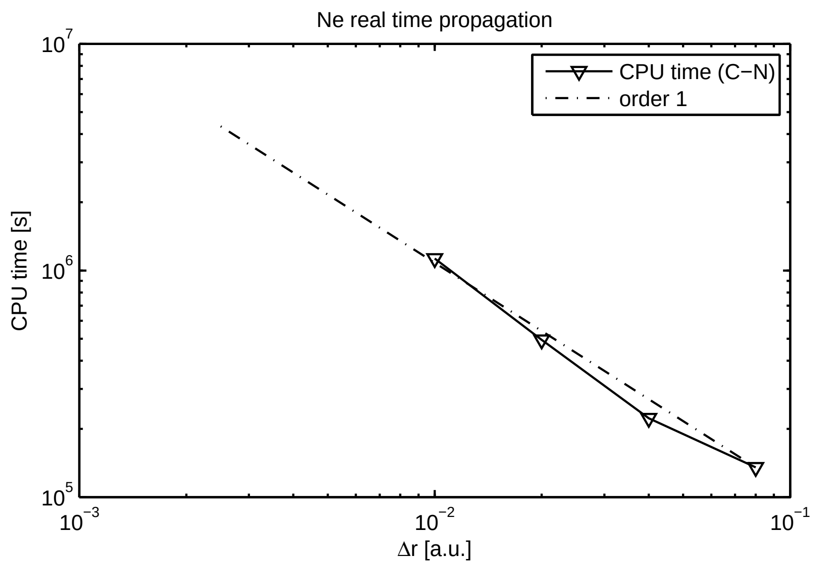

Figure 6.3: Computation time for neon vs. Δr
Real time propagation of neon, C-N propagator.

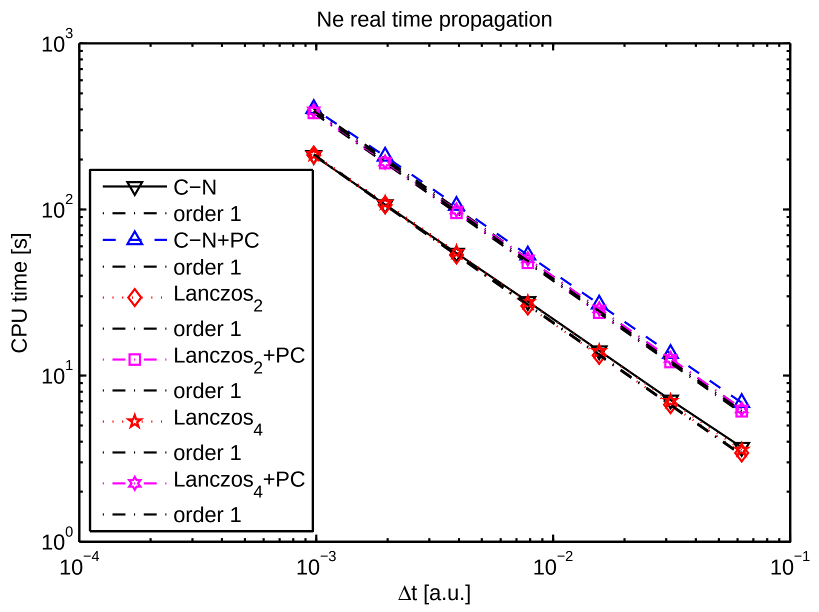

Figure 6.4: Computation time for neon vs. $\Delta t = h$
Real time propagation of neon, $L = 6$, $r_{\max} = 100$, $\Delta r = \frac{1}{50}$.

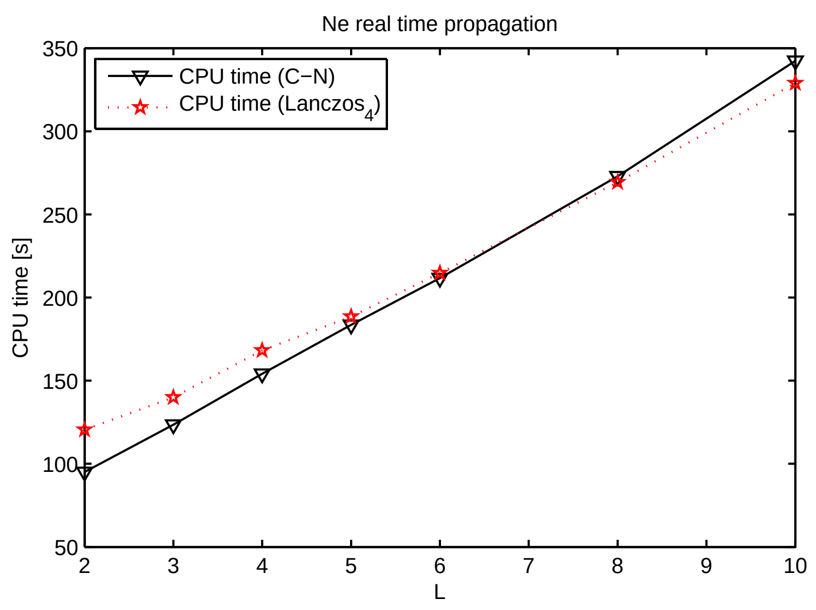

Figure 6.5: Computation time for neon vs. L
Real time propagation of neon, $r_{\max} = 100$, $\Delta r = \frac{1}{50}$, $\Delta t = \frac{1}{1024}$.

6.2 Computation time of algorithm parts

6.2.1 C-N propagator

Timing of all parts

In Figures 6.6–6.9 the CPU time requirement of all substantial parts of the QPROP propagation algorithm is compared. For small L (Figure 6.6) the most time is consumed by the computation of the KLI exchange-correlation potential V_x^{KLI}, while the other parts of the Hamiltonian consume only little CPU time. The computational effort of the `propagate(...)` function, applying the Hamiltonian to the wavefunction, becomes more important for larger L, as can be seen in Figure 6.7. For most computations L has to be rather large for sufficient accuracy, therefore optimization of the `propagate(...)` function is a main goal of this work. The normalization of the wavefunction that is done after each propagation step in imaginary time propagation takes also a non-negligible part of the CPU time. The effort to compute the energies is of the same order of magnitude too, but not mentioned here because it is not done each, typically only each tenth or hundredth step, and therefore this part is negligible.
For real time propagation the time fractions are of comparable size, apart from the normalization that is neither necessary nor useful, see Figures 6.8 and 6.9. Additionally, these two figures both show that the "Predictor-Corrector" step requires 50% of the numerical effort. This means that this step increases the numerical effort by a factor of 2 as compared to the standard propagation without the "Predictor-Corrector" step (this latter case is not explicitly shown, because the corresponding pie plot would simply stay the same, but without the

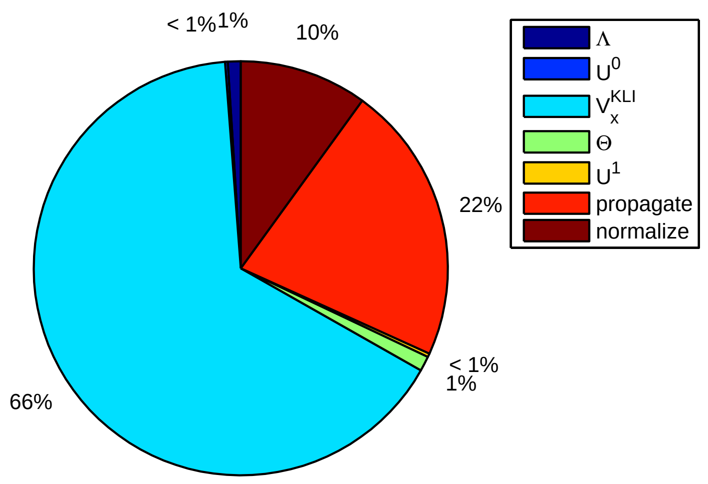

Figure 6.6: Computation time fractions for imaginary time propagation
Imaginary time propagation of neon, $L = 2$, $r_{\max} = 100$, $\Delta r = \frac{1}{400}$.

"Predictor-Corrector" step part and the percentage of all other parts doubled).

The propagator parts timing

As equation (4.25) shows, the C-N propagator consists of many parts that are applied to the wavefunction consecutively. The Figures 6.10 and 6.11 show the effort of these parts, again for small and large L. In the first case the Z matrix, linking orbitals with $\Delta\ell = \pm 2$, is neglected and therefore consumes no CPU time. But also in the second case its time fraction is only small, most of the effort is required for the terms linking orbitals with $\Delta\ell = \pm 1$ ($Y_\pm$) and containing differential operators ($W_\pm$). In both cases the parts where an inversion is necessary are dominating the other parts.
The described Figures 6.10 and 6.11 show the timing data for imaginary time propagation. The figures for real time propagation are nearly identical and are therefore not shown.

6.2.2 Lanczos method

Timing of all parts

For the Lanczos method the timing of all parts is identical to the timing of all parts for the C-N propagator, except the `propagate(...)` function, that is different and therefore consumes a different amount of CPU time. The comparison of the Lanczos method and the C-N propagator concerning numerical effort and accuracy will be given in Section 6.4.3.

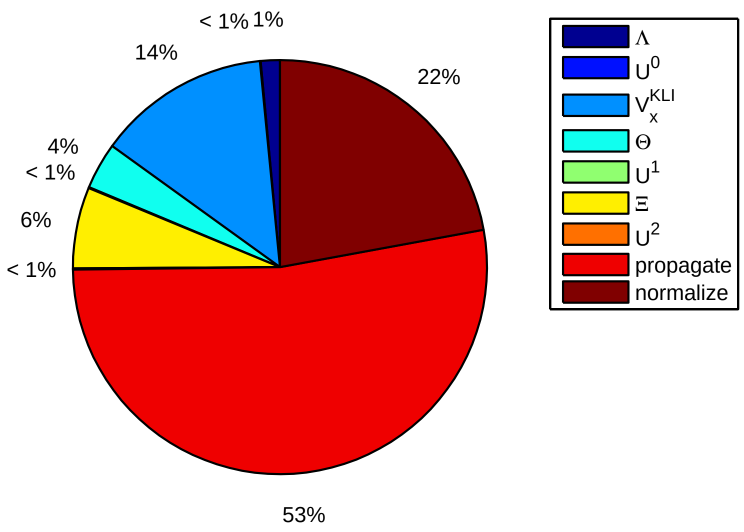

Figure 6.7: Computation time fractions for imaginary time propagation
Imaginary time propagation of neon, $L = 10$, $r_{\max} = 100$, $\Delta r = \frac{1}{400}$.

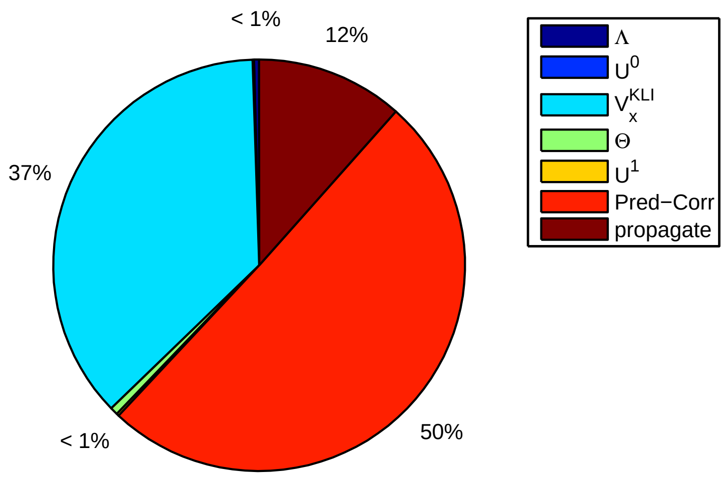

Figure 6.8: Computation time fractions for real time propagation
Real time propagation of neon with “Predictor-Corrector” step, $L = 2$, $r_{\max} = 100$, $\Delta r = \frac{1}{400}$.

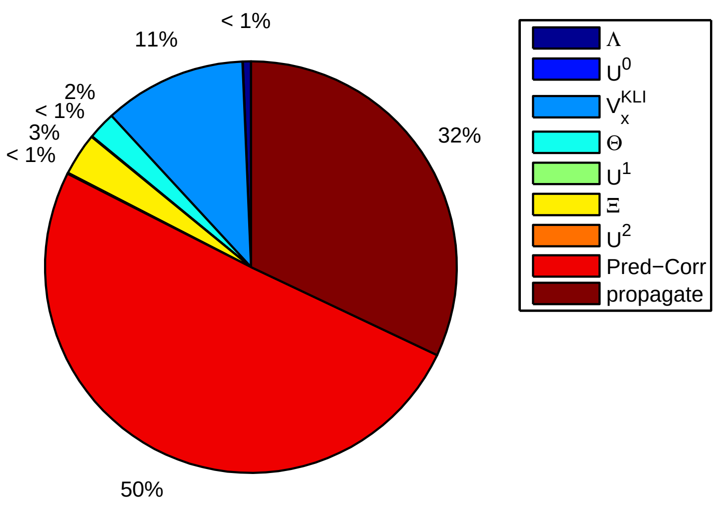

Figure 6.9: Computation time fractions for real time propagation
Real time propagation of neon with "Predictor-Corrector" step, $L = 8$, $r_{\max} = 100$, $\Delta r = \frac{1}{400}$.

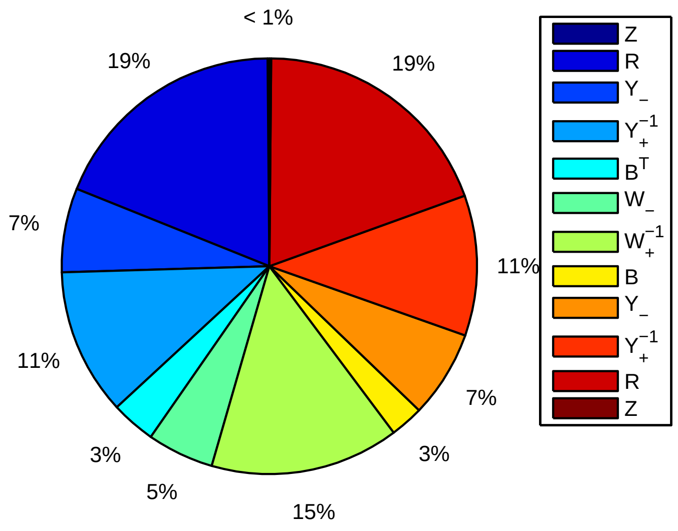

Figure 6.10: Computation time fractions for C-N propagator (4.25)
Imaginary time propagation of neon, $L = 2$, $r_{\max} = 100$, $\Delta r = \frac{1}{400}$.

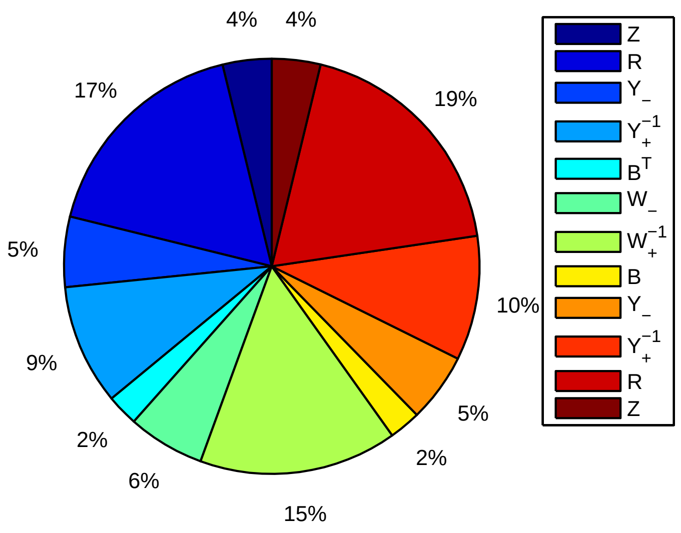

Figure 6.11: Computation time fractions for C-N propagator (4.25)
Imaginary time propagation of neon, $L = 10$, $r_{\max} = 100$, $\Delta r = \frac{1}{400}$.

The propagator parts timing

The Lanczos method consists of far fewer parts, only 5 have non-negligible contributions. These are the initialization of the matrix of the Hamiltonian (5.14) and the determination of its norm, the computation of the Lanczos basis (5.3), the Padé approximation of the exponential function and the final multiplication.
First results (Figure 6.12) have shown a rather high CPU time consumption of the final multiplication (implemented using standard C++ `for` loops), for some parameter settings even higher than for the computation of the Lanczos basis that is expected to require the most numerical effort. Due to this fact the storage of the Lanczos basis has been changed to enable the use of the matrix-vector-multiplication BLAS subroutine `zgemv(...)` in a new branch of the QPROP library. This reduces the numerical effort of the Lanczos method by about 10%. Thus, the part for the final multiplication is much smaller in Figure 6.13 showing the results for the improved algorithm. For propagation of other atoms like neon the CPU time requirement of the parts do not differ very much, as can be seen in Figure 6.14. This figure shows additionally the reduction of numerical effort compared to the old branch using C++ loops.

6.3 Imaginary time propagation

For imaginary time propagation the accuracy of the resulting ground state wavefunction depends mainly and heavily on the grid. The grid has to cover the whole space where non-negligible electron density occurs, therefore $r_{\max}$ and

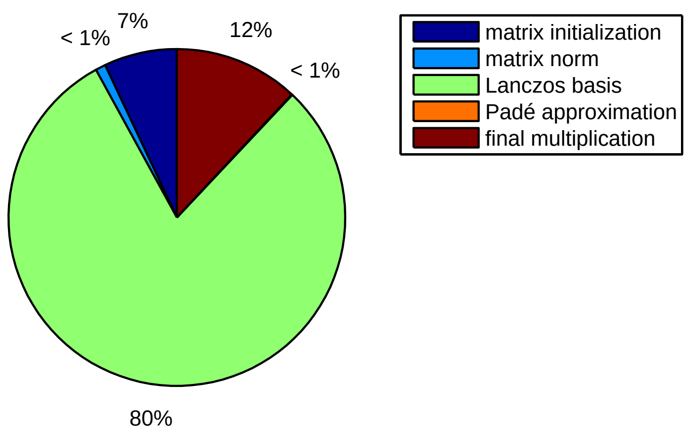

Figure 6.12: Computation time fractions for Lanczos method propagator
Real time propagation of helium, $L = 4$, $r_{\max} = 100$, $\Delta r = \frac{1}{400}$, `dx_order`$= 4$, `max_dim_krylov`$= 16$, C++ loops used for final multiplication.

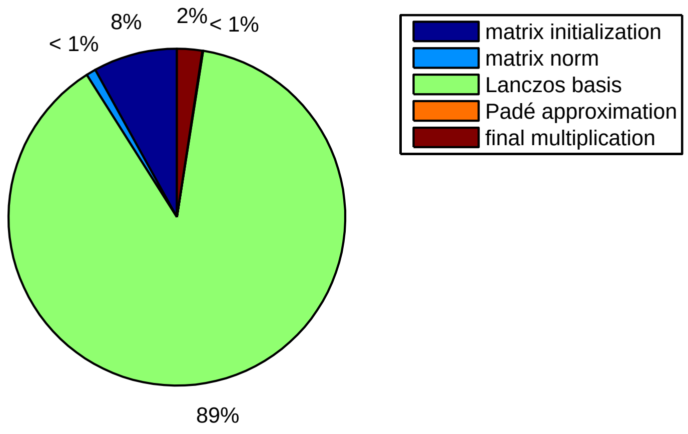

Figure 6.13: Computation time fractions for Lanczos method propagator
Real time propagation of helium, $L = 4$, $r_{\max} = 100$, $\Delta r = \frac{1}{400}$, `dx_order`$= 4$, `max_dim_krylov`$= 16$, BLAS routine used for final multiplication.

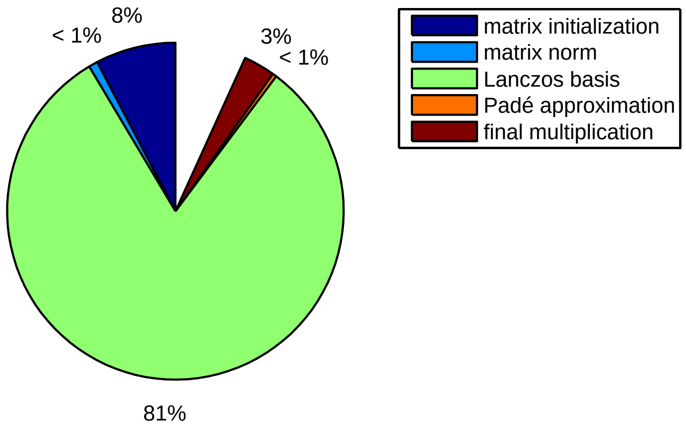

Figure 6.14: Computation time fractions for Lanczos method propagator Real time propagation of neon, $L = 6$, $r_{\max} = 100$, $\Delta r = \frac{1}{50}$, `dx_order`$= 4$, `max_dim_krylov`$= 20$, BLAS routine used for final multiplication compared to 100% represented by the total CPU time for the old branch using C++ loops for final multiplication.

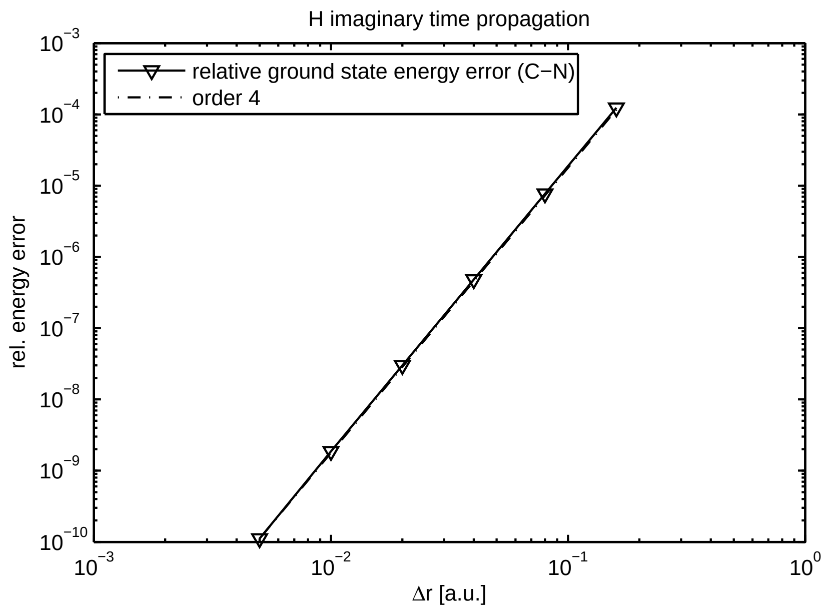

Figure 6.15: Energy error over space discretization parameter Δr for a hydrogen atom, C-N propagator, $r_{\max} = 100$, $\Delta t = \frac{1}{16}$.

$L = \ell_{\max} + 1$ have to be chosen sufficiently large ($\ell_{\max}$ has to be greater than or equal to the maximal angular momentum of physical orbitals that are occupied in ground state, for example 1 for neon because of the p-orbitals). This condition guarantees that the accuracy depends on the mesh width Δr of 4th order for both hydrogen-like as well as many-electron atoms.

6.3.1 Hydrogen-like atoms

As an example for single electron atoms/ions hydrogen is investigated. This element shows a nearly perfect 4th order dependence of the relative energy error on the space discretization parameter Δr as can be seen in Figure 6.15.

6.3.2 Many-electron atoms

Figure 6.16 visualises the convergence of the wavefunction towards the ground state for carbon. The symmetric distribution of p-electrons (two thirds for each 2p-orbital) shows a faster convergence than the asymmetric distribution that corresponds to Hund's rules.

Neon serves as an example for the determination of the order for many-electron atoms. Figure 6.17 shows, as for hydrogen before, also 4th order dependence of the relative energy error on Δr.

For spherically symmetric (including all closed-shell) atoms like hydrogen, helium or neon it is sufficient to include only the angular momenta that are necessary to describe the initial hydrogenic orbitals ($L = 1$ for helium, $L = 2$ for neon) because of the spherical symmetry that removes the ℓ-coupling (H_{mix} and $H_{\text{ang}}^{(1,2,3)}$ are zero, compare equations (2.48)–(2.52)). Therefore it makes no sense

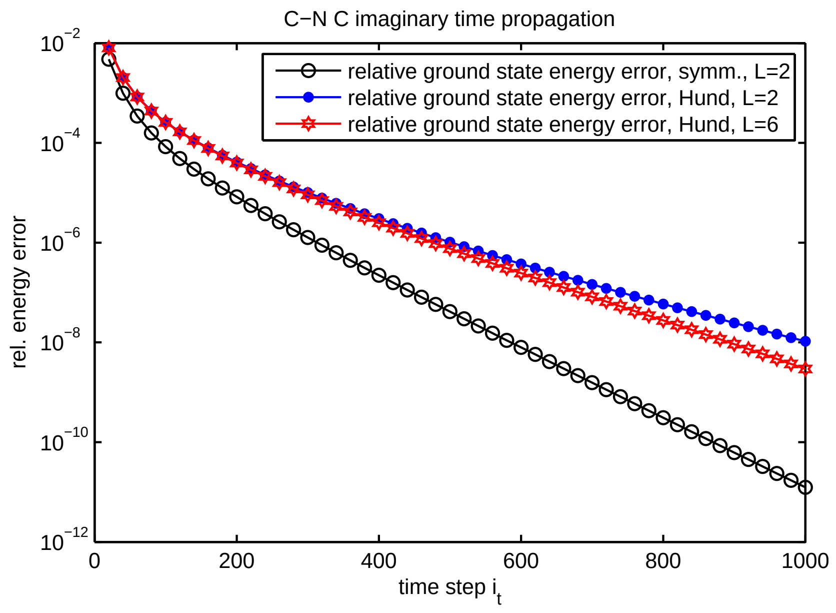

Figure 6.16: Convergence of the ground state energy approximation Energy error over time step i_t for imaginary time propagation of a carbon atom. Comparison of convergence between symmetrically distributed p-electrons (legend: symm.) and the p-electron distribution corresponding to Hund's rules (legend: Hund) with different $L = \ell_{\max} - 1$. Further parameters: $r_{\max} = 100$, $\Delta r = \frac{1}{200}$, $n_t = 1000$ steps, $\Delta t = h = \frac{1}{16}$.

to include higher ℓ states in the ground state calculation of closed shell atoms. For atoms with partially filled shells this is no longer the case if the electrons are distributed non-symmetrical as corresponding to Hund's rules. Here, Figure 6.18 shows an exponential error decay for increasing L until the region of round-off errors is reached. For the carbon atom, $L \geq L_{\min} = 2$ is necessary, and, because of the rapidly decreasing error level, L should be chosen such that the angular momentum disretization error lies below other discretization errors. Thus, for most applications involving carbon, $L = L_{\min} + 2 = 4$ is sufficient. In general, $L = L_{\min} + 2$ or $L = L_{\min} + 4$, depending on the required accuracy, is sufficient.

6.3.3 Random initializations

The standard method of initializing the Kohn-Sham orbitals for imaginary time propagation is to use the exact hydrogenic orbitals, but it is also possible to use random numbers. Figure 6.19 shows that the orbitals converge for both initialization modes to the ground state. It is clear that for hydrogen the initialization with hydrogenic orbitals is already exact, such that the round-off errors due to discretization and energy computation are shown in this case. For many-electron atoms like helium both initialization modes converge faster than the hydrogen random number initializations and the difference between the two initialization modes is much smaller. The random number initialization requires

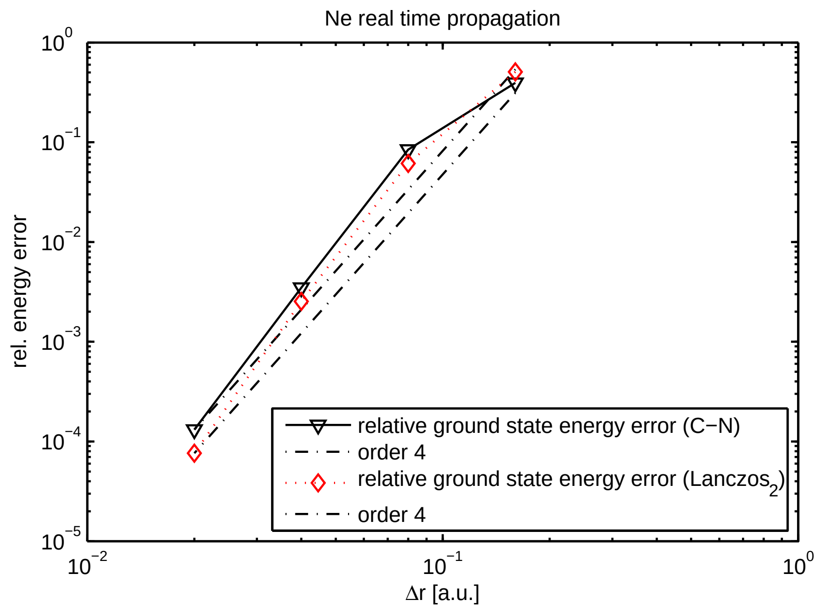

Figure 6.17: Ground state energy error over Δr
Imaginary time propagation of a neon atom, $r_{\max} = 100$, $n_t = 2000$ steps, $\Delta t = \frac{1}{8}$.

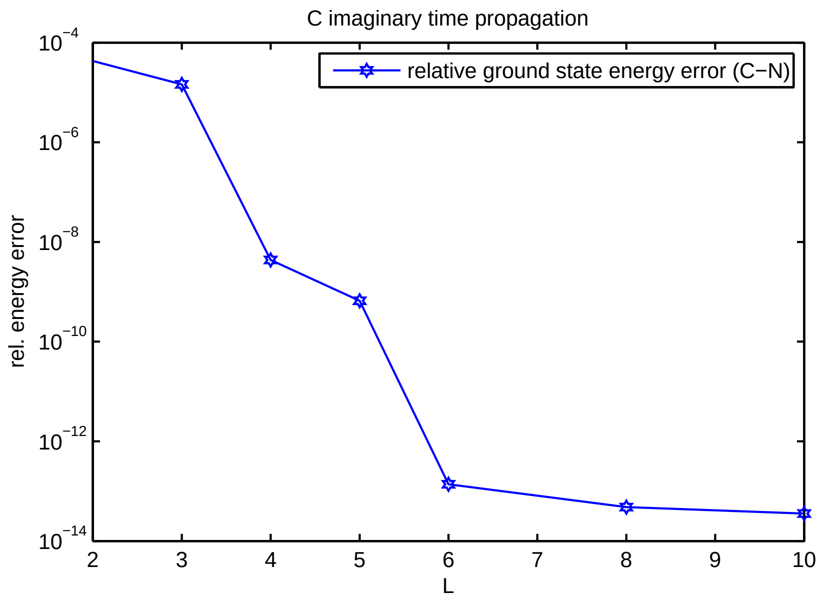

Figure 6.18: Angular momentum discretization parameter L dependence
Energy error over angular momentum discretization parameter L for imaginary time propagation of a carbon atom, $r_{\max} = 100$, $\Delta r = \frac{1}{200}$, $n_t = 1000$ steps, $\Delta t = \frac{1}{16}$.

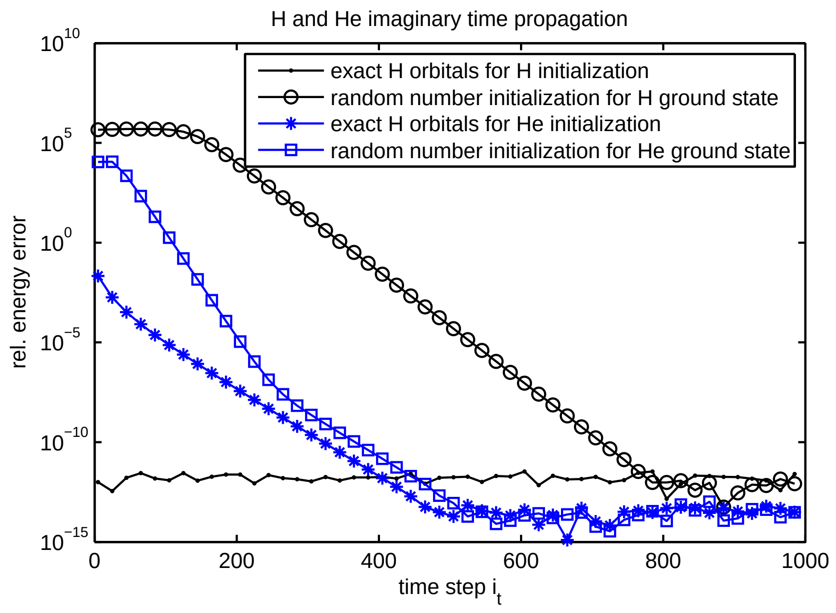

Figure 6.19: Convergence of the ground state energy approximation
Energy error over time step i_t for imaginary time propagation of a hydrogen and a helium atom. Comparison of convergence between initialization with hydrogenic orbitals and with random numbers, $r_{\max} = 100$, $\Delta r_{\mathrm{H}} = \frac{1}{400}$, $\Delta r_{\mathrm{He}} = \frac{1}{100}$, $n_t = 1000$ steps, $\Delta t = \frac{1}{16}$.

only 40 additional steps (approximately 10 percent of the number of steps) to reach the same accuracy level as the hydrogenic initialization.

6.4 Real time propagation

6.4.1 Hydrogen-like atoms

For hydrogen the global discrete propagation error depends quadratically on the time step size $\Delta t = h$, see Figure 6.20. The hydrogen real time propagation algorithm benefits from the fact that the Hamiltonian explicitly only depends on time and is therefore known for all times because it does not depend on the electron density at all. For many-electron atoms this is no longer the case.

6.4.2 Many-electron atoms

In many-electron atoms the electrons interact with each other and therefore the Hamiltonian in the Kohn-Sham equations depends on the electron density. This causes an implicit time dependence of the Hamiltonian. Using the same algorithm as for hydrogen before and additionally evaluating the time dependent exchange-correlation part of the Hamiltonian only at the starting point of the actual time propagation interval reduces the order of the algorithm to 1.

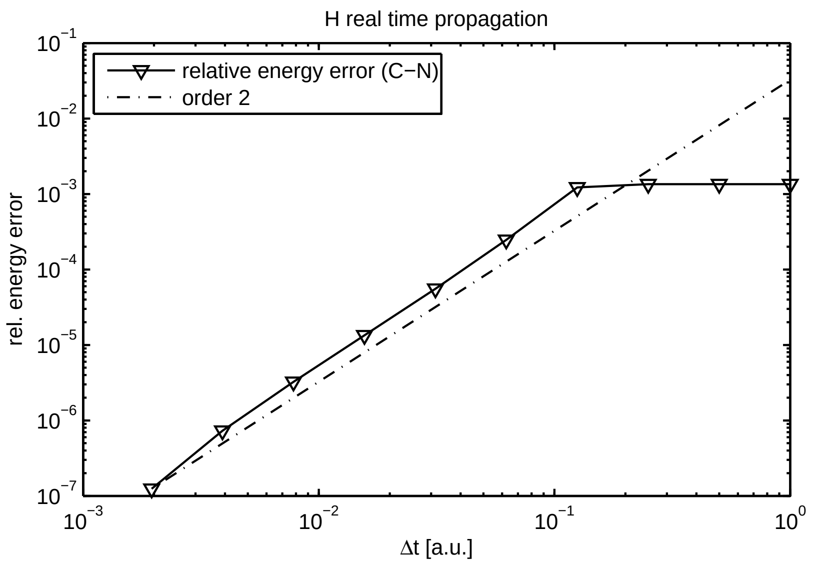

Figure 6.20: Energy error over $\Delta t = h$
for a hydrogen atom in real time propagation at $T = 20$, $\Delta r = 0.02$.

"Predictor-corrector" extrapolation step

This can be improved by a "Predictor-corrector" extrapolation step as explained in Section 5.2.
The results are shown in Figure 6.21. The order of the algorithm is raised to 2 by this predictor-corrector step, while the numerical effort only doubles. The dependence on the spatial discretization parameter Δr is of order 4, see Figure 6.22.

6.4.3 Comparison to Lanczos method

Figures 6.23–6.24 show the relative energy error over required CPU time. Thus, the lower the error and the CPU time consumption is, the higher is the performance of the algorithm. It is clear, that it cannot be expected, that the differences between the C-N propagator and Lanczos method are not the same for all atoms and parameter sets, therefore a selection of different atoms and parameter sets is presented.
For a rather coarse grid ($\Delta r = 0.16$) in helium real time propagation the standard C-N propagator is advantageous, the grid is even not fine enough to let the "Predictor-Corrector" step play an important role. In this setting the Lanczos method is orders of magnitude worse, see Figure 6.23.
For a neon atom with a finer grid ($\Delta r = 0.04$, Figure 6.24) the situation changes, here the Lanczos method achieves better accuracy at more CPU time consumption, while the C-N propagator is optimal for fast computation at an only slightly increased error level.

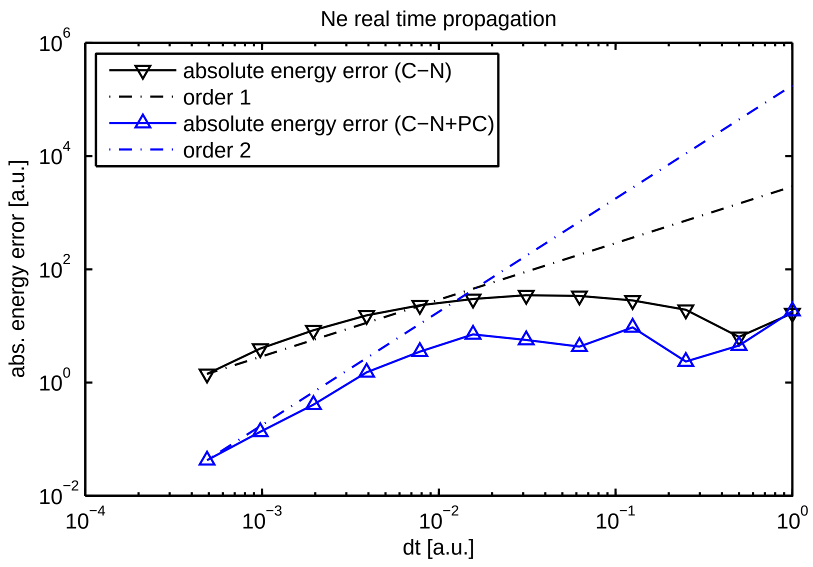

Figure 6.21: "Predictor-Corrector": energy error over $\Delta t = h$ for a neon atom in real time propagation at $T = 10$, with and without "Predictor-Corrector" extrapolation step.

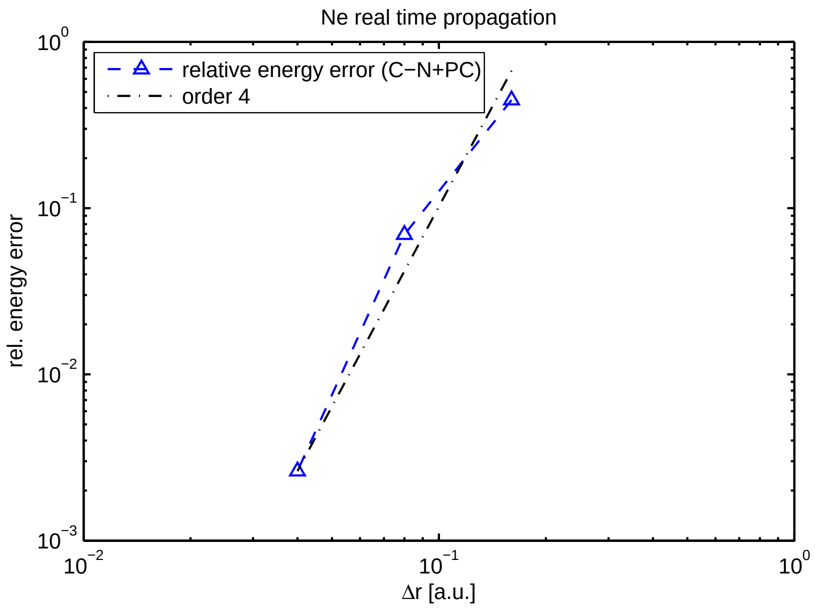

Figure 6.22: Energy error over space discretization parameter Δr for a neon atom in real time propagation at $T = 100$, $\Delta t = h = \frac{1}{512}$.

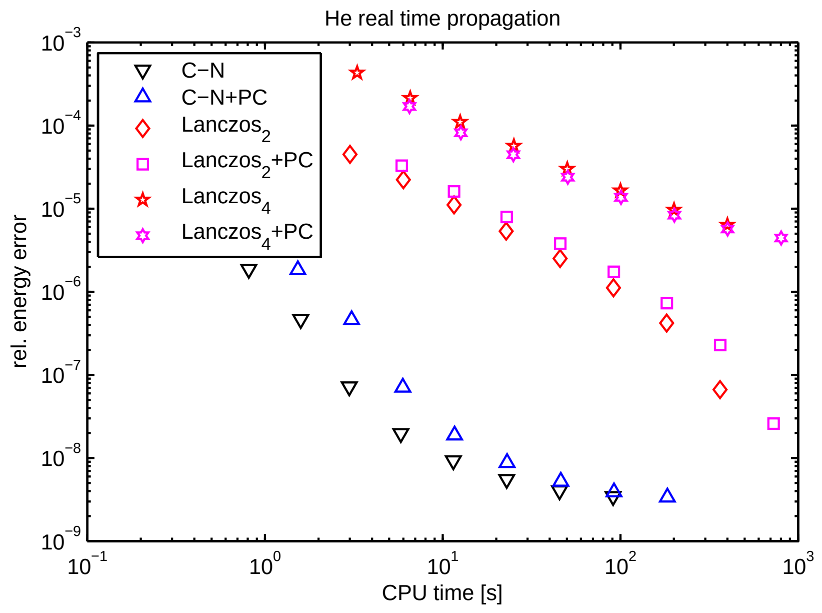

Figure 6.23: Relative energy error over CPU time for a helium atom in real time propagation at $T = 20$, $\Delta r = 0.16$.

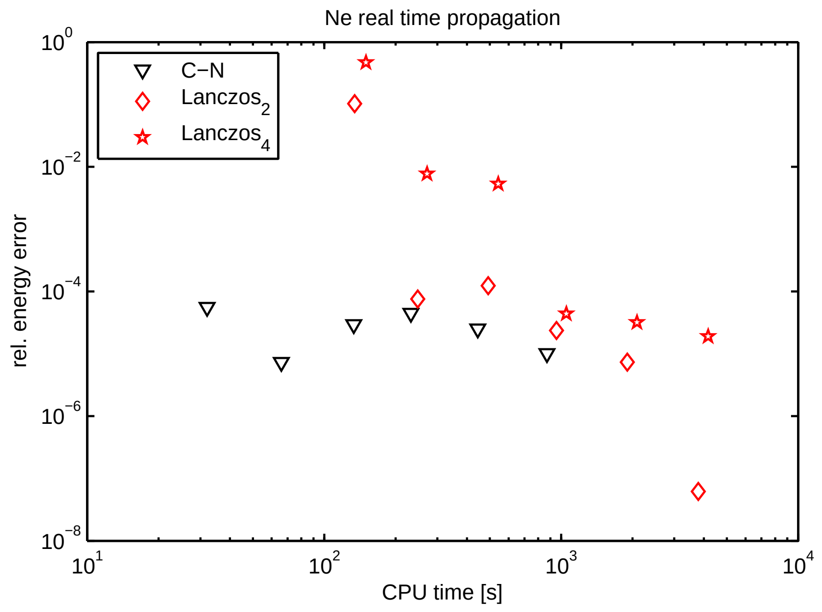

Figure 6.24: Relative energy error over CPU time for a neon atom in real time propagation at $T = 20$, $\Delta r = 0.04$.

6.5 Conserved quantities

The conservation of norm and energy are important properties of a propagator. Therefore numerical tests are run for both quantities for the C-N as well as the Lanczos method propagator.
It is clear that both properties are only valid for the real time propagation, for imaginary time propagation they make no sense. In imaginary time propagation mode the norm is not conserved by definition and therefore the wavefunction is normalized after each propagation step. The energy, on the other hand, converges to the best approximation in imaginary time propagation.
Starting with the ground state in real time propagation without any external field, both the norm and the energy of the wavefunction have to be conserved. With an external field the norm still has to be conserved, as long as the field is weak enough such that the part of the wavefunction which is absorbed by the imaginary absorbing potential (ionization) is negligible.
After a previously interacting laser field is switched off, the energy and norm have to be conserved again.

6.5.1 Norm conservation

Figures 6.25 and 6.26 illustrate the results for a helium atom. First, the external fields are zero for all time, where the Lanczos method keeps the relative deviation from norm 2 within 10^{-12} as compared to the initial deviation. The C-N propagator shows a slight drift such that the conservation of norm is about two orders of magnitude worse.
With an external laser field turned on within the period $[0, 60]$, the Lanczos method still conserves the norm at nearly the same accuracy as without a field, while the C-N propagator suffers increasing deviations after the end of the pulse that reach up to a relative level of about 10^{-7}, which are five orders of magnitude more than for the Lanczos method.
The Lanczos implementation seems preferable in this respect.

6.5.2 Energy conservation

The results for a helium atom are shown in Figures 6.27–6.29. Comparing Figure 6.27 (C-N) with Figure 6.28 (Lanczos method) the Lanczos method shows an approximately two orders of magnitude reduced energy deviation if the laser field intensity is zero.
For a non-zero laser pulse in the time interval $[0, 60]$ and the external electromagnetic field being zero for $t > 60$ the situation changes (see Figure 6.29), especially directly after the laser pulse the Lanczos method shows increased deviations as compared to the relatively constant C-N propagator.

6.6 Time reversibility

The reversibility of the algorithm is tested by going one step forward in time and then going this step back again. A reversible algorithm has to reproduce the initial wavefunction, especially its norm and energy. But even a “perfect”

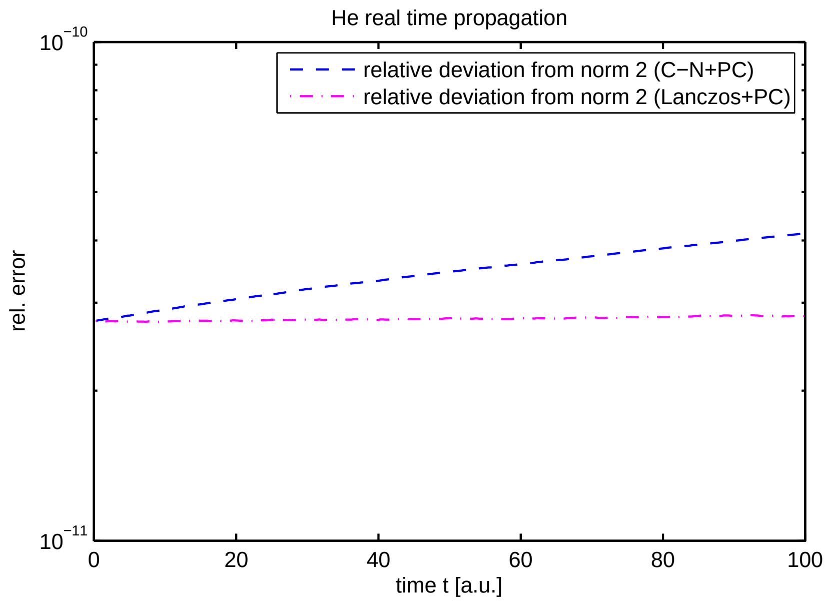

Figure 6.25: Relative norm deviation
for a helium atom without external fields, comparison of C-N and Lanczos propagator, both with PC step, time interval $[0, 100]$, $\Delta t = h = \frac{1}{128}$, $r_{\max} = 100$, $\Delta r = \frac{1}{400}$.

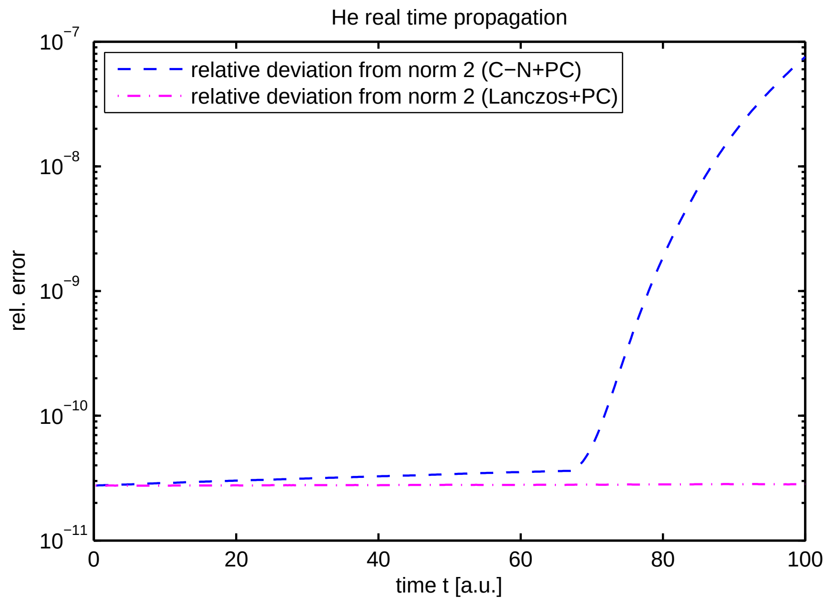

Figure 6.26: Relative norm deviation
for a helium atom with external laser field ($I = 5 \cdot 10^{12}$, period $[0, 60]$), comparison of C-N and Lanczos propagator, both with PC step, time interval $[0, 100]$, $\Delta t = h = \frac{1}{128}$, $r_{\max} = 100$, $\Delta r = \frac{1}{400}$.

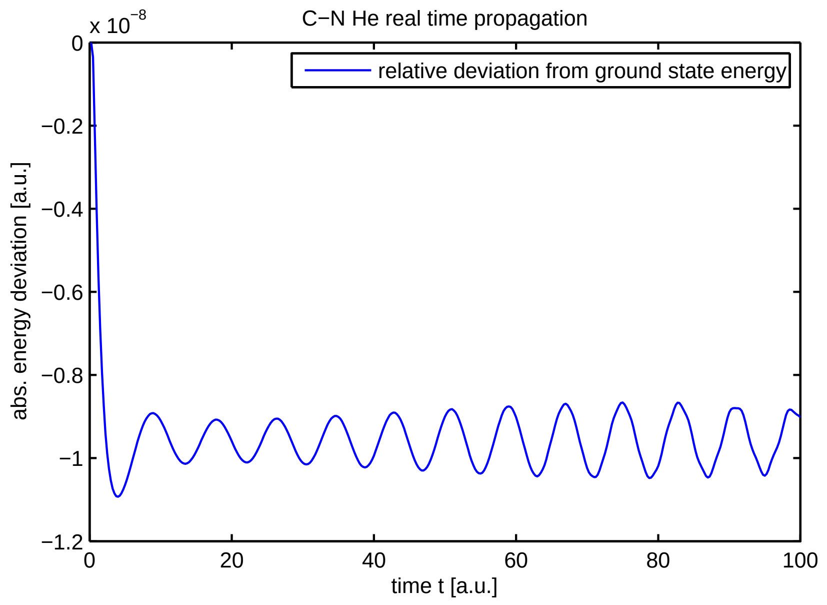

Figure 6.27: C-N energy deviation for a helium atom with "Predictor-Corrector" C-N propagator without external fields, time interval $[0, 100]$, $\Delta t = h = \frac{1}{128}$, $r_{\max} = 100$, $\Delta r = \frac{1}{400}$.

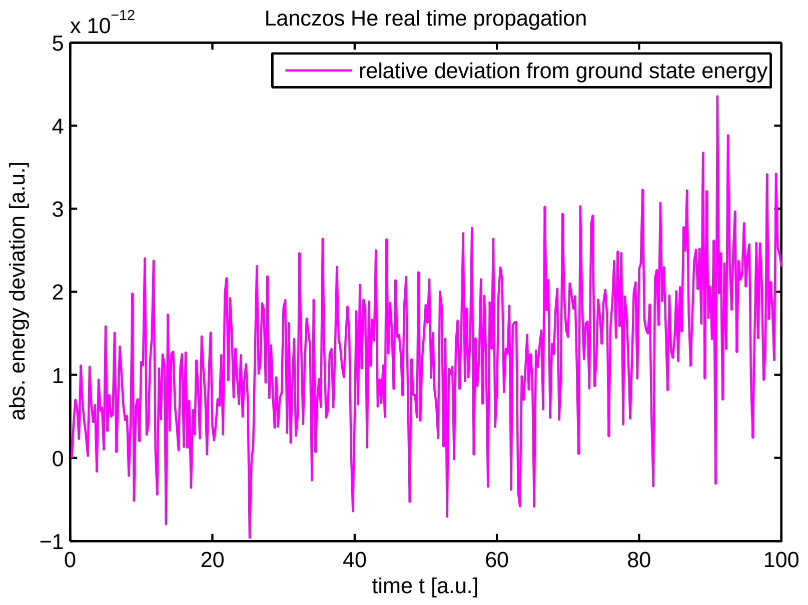

Figure 6.28: Lanczos energy deviation for a helium atom with "Predictor-Corrector" Lanczos method propagator with fixed step size and without external fields, time interval $[0, 100]$, $\Delta t = h = \frac{1}{128}$, $r_{\max} = 100$, $\Delta r = \frac{1}{400}$.

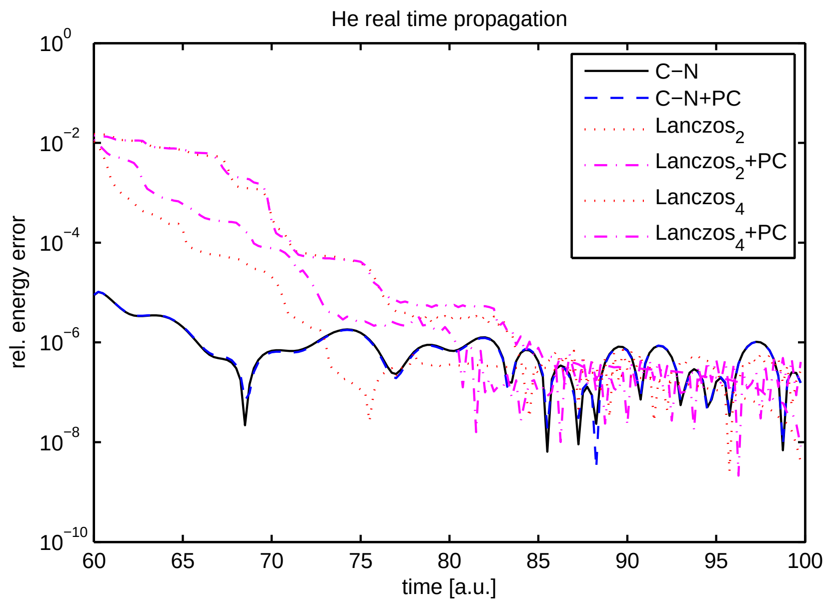

Figure 6.29: Energy deviation for a helium atom after laser pulse for different propagators, time interval $[60, 100]$, $\Delta t = h = \frac{1}{128}$, $r_{\max} = 100$, $\Delta r = \frac{1}{100}$.

algorithm is affected by round-off errors, such that one can only expect very low errors of order of magnitude *eps*, where *eps* denotes the relative machine accuracy. For all algorithms (C-N and Lanczos, with and without “Predictor-Corrector” step) the relative energy error as well as the error of the wavefunction norm for hydrogen and neon stays below 10^{-14}.

Chapter 7

Conclusions

7.1 Theory and Documentation

The theory and implementation details of QPROP are presented and analysed in its entirety. The documentation within this work on the one hand gives an idea how to handle QPROP and apply it to a given problem, on the other hand it shows how the QPROP library works and what its numerical properties are. Additionally the README file (Appendix A) provides a simple guideline for the installation of the QPROP software on personal computers and computer systems.

7.2 Front-end

The new QPROP front-end XMLQPROP is implemented and described in Chapter 3. It enables to apply the QPROP library to a wide range of problems without neither re-compilation nor new implementation. Therefore the front-end and its easily understandable XML-parameter file interface facilitate getting into the use of QPROP. But there are also advantages for the experienced user because the code can be easily extended to further problems due to the object oriented implementation of QPROP and its front-end.

7.3 Improvements and optimizations

Not only a front-end is added to QPROP, the library is also improved by adding additional classes for an object-oriented treatment of external laser field potentials, for the timing of the programs parts as well as for the pre-computation and storage of several coefficients that are often used by the library. The existing classes are further optimized, besides the reduction of numerical effort and error correction the readability of the source code was improved and comments for better understanding were added at several points in the source code.

7.4 Numerical properties

The numerical results of Qprop show lots of interesting effects. The first is the order reduction in time discretization (from order 2 to 1) when going from the single electron atom (hydrogen) to many-electron atoms like neon due to the non-linearity of the many-body problem. This is resolved by the introduction of a "Predictor-Corrector" step bringing the order back from 1 to 2 while increasing the numerical effort by a factor of 2.
Imaginary time propagation is only an additional option of the code to compute initial ground state wavefunctions. Here the orbitals converge very fast and with high accuracy to the ground state energy level, even if they are initialized with random numbers. While for spherically symmetric electron shells the choice of ℓ_{max} is very simple, the computation of the carbon ground state corresponding to Hund's rules shows exponential error decay for increasing ℓ_{max}.
The implementation of Lanczos method enables further the comparison of two different propagators. Being nearly equivalent in imaginary time propagation where the numerical effort and energy error levels are very low in general as compared to real time propagation, many differences can be found for real time propagation and different parameter settings. For standard applications the standard C-N propagator seems to be preferable, but the Lanczos method is advantageous in norm- and partially in energy-conservation. The flexibility of the Lanczos method in adding higher multipole terms and different differential operator discretizations as well as several electromagnetic field gauges should also be mentioned in this context.
On the bottom line it cannot be decided in general which propagator is the best for all applications, it depends on the posed problem and parameter sets (the latter depends on required accuracy and available computational power). But the detailed results in Chapter 6 should help to choose the best-suited propagator for many individual applications.

7.5 Outlook

This work represents one step in the development of Qprop. Although it might be a big step, it will not be the last one, there is a lot of place left for further development. One future step may be the inclusion of higher order exchange correlation potential multipole terms, which is facilitated by the Lanczos method implementation. Additionally the spin of electrons may also be treated as well as further laser field polarizations. To increase the performance for modern multi-core CPUs and other parallel computing systems the propagate algorithm can be parallelized. These and other further upgrades would enable to extend the field of possible applications.

List of Figures

List of Tables

Appendix A

Readme

The following `README.txt` file was added to QPROP to facilitate the installation of the package by a step-by-step installation and compilation guide:

```
README and quick installation guide for the qprop version as implemented
in the diploma thesis of Josef Kamleitner

1.  Install libraries

First the libraries "lapack", "blas" and "xerces-c" have to be installed.
If possible, install the "f2c" or "g2c" library!


2.  Unpack the qprop zip file or check out the repository

After this step there should be a "trunk" directory (maybe inside "da"
directory) that contains at least directories named "expokit", "libf2c"
and "qprop".


3.  Make supporting libraries

Change to the "libf2c" library and type "make".

Change to the "expokit/fortran" directory and type "make".
Now the file "expokit.o" should be in this directory.


4.  Edit qprop library makefile

Change to the "qprop" directory and edit the ROOT entry in "GNUmakefile.tmpl"
such that it points to the current (the "qprop" directory).


5.  Compile qprop and the front-end xmlqprop
```

```
Change to the "qprop/src/xmlqprop" directory and type "make xmlqprop_c"


6.  The front-end xmlqprop and the qprop library should now work properly.

To start computations using the xml parameter file "default.xml" type
"./xmlqprop default".
To start the computations in background, type "sh ./xqs default".
```

Appendix B

C++ code-lets and XML parameter files

B.1 The default parameter file

The XML parameter file `default.xml` shows the structure of the XML parameter files. The parameter values below describe imaginary time propagation for the neon atom:

```
<!-- this is the default parameter file ↩
↪ (values for a neon atom)-->
<params>
   <static_params>
     <realtimemode>0</realtimemode>
     <hydrogenlike>0</hydrogenlike>
     <extrapolate>0</extrapolate>
     <numtisranget>0</numtisranget>
     <lanczos>0</lanczos>
     <i_v>1</i_v>
     <i_slateronly>0</i_slateronly>
     <dimens>34</dimens>
     <n_orb>3</n_orb>
     <n_ang_mom>2</n_ang_mom>
     <degener>
       <d1>2</d1>
       <d2>2</d2>
       <d3>6</d3></degener>
     <ms>
       <d1>0</d1>
       <d2>0</d2>
       <d3>0</d3></ms>
     <ells>
       <d1>0</d1>
       <d2>0</d2>
       <d3>1</d3></ells>
     <really_propagate>
       <d1>1</d1>
       <d2>1</d2>
       <d3>1</d3></really_propagate>
     <nuclear_charge>10.0</nuclear_charge>
     <imagpot_ampl>100.0</imagpot_ampl>
     <imagpot_inner_border>0.0</imagpot_inner_border>
   </static_params>
   <dynamic_params>
     <range_x>100.0</range_x>
     <range_t>100.0</range_t>
     <n_outputs>200</n_outputs>
     <dx>
       <i>0.16</i>
       <f>0.0025</f>
       <a>0</a>
       <b>0.5</b></dx>
     <dt>
       <i>1.0</i>
       <f>1.0e-4</f>
       <a>0</a>
       <b>0.5</b></dt>
     <intens>
       <i>0.0</i>
       <f>0.0</f>
       <a>2.0</a>
       <b>2.0</b></intens>
   </dynamic_params>
   <init_params>
     <i_initmode>2</i_initmode>
     <filenamepart_ini>np_ground_neon</filenamepart_ini>
     <multiinit>0</multiinit>
     <n_orb_ini>3</n_orb_ini>
     <n_ang_mom_ini>2</n_ang_mom_ini>
     <range_x_ini>100.0</range_x_ini>
     <dx_ini>0.0025</dx_ini>
   </init_params>
   <lanczos_params>
     <adaptive>0</adaptive>
     <hermitian>1</hermitian>
     <dx_order>2</dx_order>
     <max_dim_krylov>16</max_dim_krylov>
     <loc_tol>1.0e-14</loc_tol>
     <mxstep>500</mxstep>
     <mxreject>1000</mxreject>
     <pade_deg>6</pade_deg>
     <lanczos_delta>1.2</lanczos_delta>
     <lanczos_gamma>0.9</lanczos_gamma>
   </lanczos_params>
</params>
```

B.2 XML parsing source code examples

The following source code examples show the most important cases how to parse parameters of different types like doubles, booleans, strings, arrays of integers and parsing with default values for backwards compatibility. Other possible cases like parsing floats, for example, can be easily derived from the provided examples.
The types of the read-in parameters are self-explanatory, the doc is of type DOMDocument*, aNode and rootNode of type DOMNode* and aNodeList of type DOMNodeList*. Furthermore the parser parser of type XercesDOMParser* is used as well as static functions belonging to the XMLString-class.
All these special classes and types are defined and implemented within the Xerces-C library.

B.2.1 Parsing a double precision number parameter

The following example illustrates how a double precision parameter is parsed.

```
aNode=doc->getElementsByTagName(XMLString::transcode("range_x")) ↩
↪ ->item(0);
sscanf(XMLString::transcode(aNode->getTextContent()), ↩
↪ "%lf",&rangex);
cout << "rangex = " << rangex << endl;
```

In the first line the first element of the document doc with tag name range_x is assigned to the node aNode. In the second line the text content of this node is transcoded from the XMLString to char type and scanned as double precision number into the variable rangex. Finally the value of the read-in parameter is displayed on the screen as information to the user.

B.2.2 Parsing a boolean parameter with default value

The following example illustrates how a boolean parameter is parsed. Additionally this implementation shows how to treat XML-file parameters that are added in the future without losing compatibility to older parameter files. Therefore it is checked whether the parameter exists and, if not, a default value is used. Otherwise the standard procedure is applied.

```
aNodeList=doc->getElementsByTagName( ↩
↪ XMLString::transcode("lanczos"));
if(aNodeList->getLength()==0){
   cout << "WARNING: error reading lanczos from XML-file, ↩
  ↪ setting default value!" << endl;
   lanczos=false;
   cout << "standard propagation mode selected (no Lanczos)" ↩
  ↪ << endl;
}else{
   aNode=aNodeList->item(0);
   sscanf(XMLString::transcode(aNode->getTextContent()), ↩
  ↪ "%d",&tempint);
```

```
   if(tempint){
     lanczos=true;
     cout << "Lanczos propagation mode selected" << endl;
   }else{
     lanczos=false;
     cout << "standard propagation mode selected (no Lanczos)" ↩
    ↪ << endl;
   }
}
```

In the first line, all elements of the document `doc` with tag name `lanczos` are assigned to the node list `aNodeList`. In the if-statement it is checked whether this node list is empty (length equal to 0). For an empty node list there is no Lanczos element in the document and therefore a warning is displayed and the default value `lanczos=false` is set. Otherwise, if the Lanczos method parameter is defined within the XML-file, its value is scanned by the `sscanf` statement out of the transcoded text content of the node into the temporary integer variable `tempint`. Depending on the value of tempint, the variable `lanczos` is set (`true` for 1 and `false` for 0). Corresponding user information is displayed for each case.

B.2.3 Parsing a string parameter

The following example illustrates how a string parameter is parsed.

```
aNode=doc->getElementsByTagName( ↩
↪ XMLString::transcode("filenamepart_ini"))->item(0);
sprintf(cstr_fname_wf_ini, ↩
↪ "./res/%s.dat",XMLString::transcode(aNode->getTextContent()));
cout << "cstr_fname_wf_ini = '" ↩
↪ << cstr_fname_wf_ini << "'" << endl;
```

The first element of the document `doc` with tag name `filenamepart_ini` is assigned to the node `aNode` in the first line. In the second line the text content of this node is transcoded from the `XMLString` to `char` type and printed into the `cstr_fname_wf_ini` variable as defined by the format string `./res/%s.dat`. Finally the value of the read-in parameter is displayed on the screen as information to the user.

B.2.4 Parsing an array of integer parameters

By the example of the `really_propagate`-parameter, consisting of `norb` integer values `really_propagate[i]`, the parsing of an array of integers is illustrated.

```
aNode=doc->getElementsByTagName( ↩
↪ XMLString::transcode("really_propagate"))->item(0);
aNodeList=aNode->getChildNodes();
cout << "really_propagate:  ( ";
for(i=0;i<norb;i++){
   sscanf(XMLString::transcode(aNodeList->item(2*i+1) ↩
 ↪ ->getTextContent()), "%d",&really_propagate[i]);
   cout << really_propagate[i]<< " ";
}
cout << ")" << endl;
```

The first element of the document `doc` with tag name `really_propagate` is assigned to the node `aNode` in the first line. Then the child nodes of this node are put into the node list `aNodeList` in the second line, before the output and the for-loop starts. The `cout` statements printing brackets and spaces are only used to format the output for the user. Important is the for-loop over all indices `i` from 0 to `norb`−1 in which the items, corresponding to the $\texttt{i}^{\text{th}}$ parameter array element, are selected from the list and their content is transformed via the `transcode` and `sscanf` function into a corresponding integer variable.

B.3 Lanczos method source code parts

The main parts of the Lanczos method are explained in detail in Section 5.1.3, the adaptive time stepping and Padé approximation in Section 5.1.4. There the computation of the adaptive time step size and the local error estimate was only roughly described and will subsequently be explained in more detail including source code parts.

B.3.1 Adaptive step size

The first step size is computed by the function `init_step(...)` from `beta` (`beta` is the norm of v), `anorm` (`anorm` is the norm of H), the tolerance `tol` and the dimension `m` of the Krylov subspace.

```
template<class M> double Expokit<M>::init_step(double beta){
   return this->trans_step(1/anorm*pow(tol*pow((m+1)/M_E,m+1) ↩
 ↪ *sqrt(2*M_PI*(m+1))/4.0/beta/anorm,1.0/(double)m));
}
```

The next proposed step size depends on the previous one (`t_step`), the local error estimate `err_loc`, the shrinking factor `gamma` and the tolerance `tol`.

```
template<class M> double Expokit<M> ↩
↪ ::next_step(double t_step,double err_loc){
   return this->trans_step(gamma*t_step ↩
 ↪ *pow(t_step*tol/err_loc,1.0/(double)m));
}
```

Inside the adaptive step size functions given above, the originally estimated step size is transformed using the following function `trans_step`. This is a rounding function whose result is a decimal number with only the first two digits being non-zero. It is used to avoid very small remaining step sizes.

```
template<class M> double Expokit<M>::trans_step(double tau){
   double p ↩
```

```
 ↪ =exp((rint( log( tau )/M_LN10-sqrt(0.1) )-1.0)*M_LN10);
   tau=floor( tau/p + 0.55 ) * p;
   return tau;
}
```

B.3.2 Local error estimate

The following codelet implements computation of the local truncation error estimate for the Padé approximation within the Lanczos method.

```
double avnorm=this->norm2(V_m[m],n);
double p1 = abs( f[m] ) * beta;
double p2 = abs( f[mp1] ) * beta * avnorm;
if ( p1>10.0*p2 ){
   err_loc = p2;
}else{
   if ( p1>p2 ){
     err_loc = (p1*p2)/(p1-p2);
   }else{
     err_loc = p1;
   }
}
```

The three cases (selected by the if clauses) are, according to [10]:

1. small step size, quick convergence.
2. slow convergence.
3. asymptotic convergence.

Bibliography

[1] J. S. Parker, L.R. Moore, and K.T. Taylor. Accurate computational methods for two-electron atom-laser interactions. *Optics-Express*, 8:436–441, 2001.

[2] A. Bandrauk and A. Chelkowski. On laser Coulomb explosion imaging of proton motion. *Chem. Phys. Letters*, 336:518–522, 2001.

[3] P. Hohenberg and W. Kohn. Inhomogeneous electron gas. *Phys. Rev.*, 136(3B):B864, 1964.

[4] E. Runge and E. K. U. Gross. Density-functional theory for time-dependent systems. *Phys. Rev. Lett.*, 52(12):997, Mar 1984.

[5] H. G. Muller. An efficient propagation scheme for the time-dependent schrdinger equation in the velocity gauge. *Laser Physics*, 9(1):138–148, 1999.

[6] D. Bauer. Qprop manual I. *www.qprop.de*, 2005.

[7] D. Bauer and P. Koval. QPROP: A Schrödinger-solver for intense laser-atom interaction. *Comput. Phys. Commun.*, 174(5):396, 2006.

[8] L. Schimka. Interaction of atoms with laser fields; Time-Dependent Density Functional Theory and QPROP software, 2008.

[9] R. Hammerling. TDDFT-Generalized kick perturbations and monitoring observables for calculation of excitation energies. *Philosophical Magazine*, 88:2817, 2008.

[10] R. B. Sidje. Expokit: A software package for computing matrix exponentials. *ACM Trans. Math. Software*, 24(1):130–156, 1998.

[11] G. H. Golub and C. F. Van Loan. *Matrix Computations.* The John Hopkins University Press, Baltimore, 3rd edition, 1996.

[12] C. Lubich. *From Quantum to Classical Molecular Dynamics: Reduced Models and Numerical Analysis.* Zurich Lectures in Advanced Mathematics. European Mathematical Society, Zurich, 2008.

[13] N. J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM J. Matrix Anal. Appl.*, 26(4):1179–1193, 2005.

[14] E. Hairer, C. Lubich, and G. Wanner. *Geometric Numerical Integration.* Springer-Verlag, Berlin–Heidelberg–New York, 2002.