UNIVERSITY OF LINCOLN

SCHOOL OF MATHEMATICS AND PHYSICS

Optimisation of Time-Dependent Density Functional Theory methods in CP2K

Scientific Report

Matthew Thompson
THO19699057

Supervised by Professor Matt Watkins

24th May 2023

1 Abstract

A theoretical and computational method, in the form of a Python program entitled 'PyRTP', was determined to emulate the real-time time-dependent density functional theory method as implemented in CP2K. The creation of this program allowed for the computational cost and accuracy implications of a variety of elements of the RT-TDDFT process to be evaluated and compared, most notably the effect of different propagators, grid spacings and absorption spectra calculations. This method allowed for suggestions to optimise the RT-TDDFT process in CP2K to be made. Using this method, it was found that the CFM4 propagator was the most suitable for optimising the 'Real-Time Propagation' function within CP2K, given the minimal computational cost and superior accuracy over other methods. Equations to determine suitable values for propagation timestep and number of steps of any given RT-TDDFT process were also determined, allowing for minimised computational cost and sufficient accuracy when determining the absorption spectra of molecules. The possible uses of entropy in RT-TDDFT calculations were discussed and implemented into PyRTP. A comparison of the accuracy of propagation methods in PyRTP and CP2K could not be made, due to the inability to reproduce a Gaussian envelop electric pulse in CP2K.

Table of Contents

List of Tables

List of Figures

2 Declaration

"*I confirm that the content of this report is my own work and that all references and quotations from both primary and secondary sources have been fully identified and acknowledged appropriately.*"

[illegible]

Matthew A. Thompson
24th May 2023

3 Acknowledgements

Firstly, I wish to express my sincerest thanks to Professor Matt Watkins for his unrelenting support as my project supervisor. His approachability at any time of the day and his rigorous (and at points testing) project goals have pushed my abilities and knowledge over the last few months; I am exceptionally grateful to him for this. I also wish to give my thanks to Sergey Chulkov for providing technical support, as well as providing access to the Young HPC system for me. Further to this, I wish to thank the MMM Hub team for providing access to the Young HPC system, which has saved me enormous amounts of time throughout this project. I wish to thank Christopher Dickens at the University of Lincoln for his pointers in the right direction with regard to using CP2K, as well as sections of code which saved me many hours of time. I wish to extend my thanks to Adam Kinsella, Ben Darvill and Grace Smith for their very kind offers of proofreading; it was most helpful in the lead up to submission. Finally, I wish to thank the developers of CP2K for the fabulous program they have written - it has been a pleasure to work with.

4 Introduction

Atomistic simulation has been used for many years now to both predict theoretical phenomena, and arguably more importantly, verify experimental data. These methods have been in use since the early 1940s, starting first as a somewhat laborious task of using analog computers and manually computing each timestep [1]. The use of computers to perform these calculations "automatically" was first proposed by Enrico Fermi and first performed at the Los Alamos National Laboratory in New Mexico, USA in 1953 [2]. These simulations were one-dimensional and featured 64 simulated particles on the MANIAC-1 computer, which was a very early general purpose computer [3]. Since then, owing to constant improvements in computational capability, the complexity of systems under study has also consistently increased, to such a point where simulations may now accurately reflect real-world systems [4]. Given modern computational performance, there has been a great need by the experimental physics and chemistry communities for accurate programs and packages for use in research performed by these communities for many years, starting as far back as 1970, with the release of Gaussian '70 [5]. CP2K is one of many modern quantum chemistry/solid state physics program used around the world for simulations of various atomic-scale systems, using a variety of methods. It is favoured by many for its open-source, community-developed code and its applicability to a wide variety of systems, given the large array of methods featured in the program. One such method featured in CP2K is 'Time-Dependent Density Functional Theory', first implemented into the program by Florian Schiffmann as part of his PhD thesis at the University of Zurich in 2010 [6].

This method differs from conventional Density Functional Theory (DFT) calculations, as while conventional DFT calculations provide information on the ground state of an atomistic system, very little (if anything) can be learned about excited states [7]. By comparison, by using time-dependent density functional theory (TDDFT), we may gather information both about excited states of an atomistic system, but also on time-dependent phenomena affecting the system, such as the influence of an external electric or magnetic field, among a variety of other things [8]. The implementation of TDDFT in CP2K currently allows for a range of information to be acquired, including emission and absorption spectra. However, further information may be obtained during TDDFT calculation methods, which presently, are not obtainable in CP2K. The collection of further information during the process of quantum materials simulation would be useful in a wide variety of research, and as CP2K is one of the most popular molecular modelling programs in use today, any improvements to the program would benefit research around the world.

CP2K utilises a variety of methods to reduce computational cost, most significantly via heavy use of OpenMP multi-threading and MPI. CP2K also takes advantage of GPU acceleration via Nvidia CUDA for a small number of operations [9]. These ground-level programming optimisations are very scalable to the system, however further performance may be obtained by optimising the process undertaken. In the case of TDDFT, various processes in the method may be optimised, such as the ground-state DFT calculation, the propagator, etc. Thus, it is beneficial to reduce the computational cost of this calculation, whilst maintaining a similar level of accuracy.

The aim of this project is to determine a computational method to accurately model time-dependent phenomena in quantum materials. In order to achieve this, the objectives of this project are to formulate a theoretical method to perform TDDFT calculations, to determine any further information that may be obtained about the system during the process of calculation, to produce a computational method based on the proposed theoretical TDDFT method to compare to the implementation of TDDFT in CP2K, and to propose changes to the CP2K source code based on the findings of the comput -ational method.

5 Methodology

5.1 Theoretical TDDFT Method

In principle, TDDFT revolves around solving the Time-Dependent Schrödinger Equation (TDSE), which is given as the following:

$$i\hbar\frac{\partial\Psi}{\partial t} = \hat{H}\Psi \tag{1}$$

Such that Ψ is the many-body wavefunction and $\hat{H}$ is the Hamiltonian of the system. Besides TDDFT, there are a variety of methods to solve the TDSE, however in this formalism, the time-dependent Kohn-Sham equation is used, for reasons which will be discussed at greater length in Section 5.1.1.

$$i\frac{\partial}{\partial t}\Psi_n^{KS}(\boldsymbol{r},t) = \hat{H}^{KS}[\rho(t),t]\Psi_n^{KS}(\boldsymbol{r},t) \tag{2}$$

All TDDFT methods start with a determination of the ground-state energy and electron density, which uses conventional DFT. This method is preferred over other self-consistent field (SCF) methods, as DFT generally produces greater accuracy in many calculations compared to Hartree-Fock (HF) methods, which do not consider electron correlation [10]. Exchange correlation calculations may be performed during post-HF methods, and thus there is no explicit reason why post-HF methods could not be used in place of DFT for the initial ground-state calculation, besides the computational cost. For the purposes of this report, we take the view that DFT is the most suitable method to determine the ground-state energies and electron density and will not describe other SCF methods besides DFT[1].

5.1.1 DFT

Density functional theory was developed as somewhat of a continuation of the work of Llewellyn Thomas and Enrico Fermi, who via the Thomas-Fermi model, were able to describe the total energy of a quantum mechanical system as a function of electron density exclusively [12]. This total energy is described by the Thomas-Fermi energy functional, given as the following:

$$T_{TF}[\rho(\boldsymbol{r})] = C_F\int\rho^{\frac{5}{3}}(\boldsymbol{r})d\boldsymbol{r} - Z\int\frac{\rho(\boldsymbol{r})}{\boldsymbol{r}}d\boldsymbol{r} + \frac{1}{2}\iint\frac{\rho(\boldsymbol{r})\rho(\boldsymbol{r})}{|\boldsymbol{r}_1-\boldsymbol{r}_2|}d\boldsymbol{r}_1 d\boldsymbol{r}_2 \tag{3}$$

[1]In general, it is simple to replace the initial DFT calculation with any SCF method, so long as the end result is an electron density of similar accuracy to DFT. A paper published in 2017 discusses coupled TDHF and TDDFT methods with some reference to using this combination [11]. It is beyond the scope of this project, however comparative testing of real-time propagation TDDFT using post-HF methods for the ground-state calculations compared to the conventionally used DFT may be interesting to see if there is any tangible benefit.

Such that $\rho(\boldsymbol{r})$ is the electron density, C_F is a constant equal to $C_F = \frac{3}{10}(3\pi^2)^{\frac{2}{3}}$ and Z is the atomic number. This functional gives a reasonable, but not wholly accurate, picture of the total energy [13], and laid the foundation for Walter Kohn and Pierre Hohenberg to formulate the Hohenberg-Kohn theorems [14]. These two theorems, which are widely considered to be the basis of DFT (and have proofs given in Appendix A.1), are given as the following:

Theorem 1 *For non-degenerate ground states, two different Hamiltonians cannot have the same ground-state electron densities.*

Theorem 2 *The functional $E = E[\rho(\boldsymbol{r})]$ is at a minimum when the electron density $\rho(\boldsymbol{r})$ is equal to the true ground state electron density.*

These theorems allow for an N-body system with $3N$ spatial coordinates to be reduced to three spatial coordinates, which is more easily described. However, determination of the electron density using this alone for interacting systems (of which the majority of systems are) is largely inaccurate, due to the difficulty of obtaining suitable approximations of the kinetic energy functional. Walter Kohn and Lu Jeu Sham proposed a solution to remedy this, by constructing a non-interacting system with the same electron density. This allows for the kinetic energy functional to be exactly known, and the ground-state energy and electron density may be determined by minimising the Kohn-Sham equation [15], given as the following:

$$E_{KS}[\rho(\boldsymbol{r})] = T_S[\rho(\boldsymbol{r})] + E_H[\rho(\boldsymbol{r})] + E_{xc}[\rho(\boldsymbol{r})] + \int d\boldsymbol{r}\, v_{ext}(\boldsymbol{r})\rho(\boldsymbol{r}) \tag{4}$$

Such that E_{KS} is the total energy, T_S is the Kohn-Sham kinetic energy of the system, E_H is the Hartree energy (also commonly referred to as the Coulomb energy, given that this value is determined via the Coulomb interaction), E_{xc} is the exchange-correlation energy and v_{ext} is the external potential acting on the system. This expression may also be written in the following form to better illustrate the interacting and non-interacting parts:

$$E_{KS}[\rho(\boldsymbol{r})] = T_S[\rho(\boldsymbol{r})] + \int d\boldsymbol{r}\, v_{eff}(\boldsymbol{r})\rho(\boldsymbol{r}) \tag{5}$$

Such that v_{eff} is the effective external potential which moves the non-interacting particles in a similar manner to a similar system of interacting particles:

$$v_{eff} = v_{ext}(\boldsymbol{r}) + e^2 \int \frac{\rho(\boldsymbol{r}')}{|\boldsymbol{r} - \boldsymbol{r}'|}\, d\boldsymbol{r}' + \frac{\delta E_{xc}[\rho]}{\delta \rho(\boldsymbol{r})} \tag{6}$$

The last term, $\frac{\delta E_{xc}[\rho]}{\delta \rho(\boldsymbol{r})}$, is the exchange-correlation potential and is unknown by definition in this method, as it takes the form of the difference between the true energy functional and the sum of the remaining terms until this point. As such, this term is always approximated through a variety of methods.

5.1.1.1 Exchange-correlation energy approximations

Approximating the exchange-correlation energy remains an area of active research, however the methods used to approximate this value can be split into a few broader groups:

- Local Density Approximations (LDA)
- Generalised Gradient Approximations (GGA)
- Meta-Generalised Gradient Approximations (Meta-GGA)
- Hybrid XC Functionals

These methods may be viewed in some sense as evolutionary steps, as in general, the XC energy calculations are more accurate further down the list. Starting with LDA, this approach assumes that the XC energy functional can be wholly described by the electron density $\rho(\boldsymbol{r})$ exclusively, such that the XC energy functional is given as:

$$E_{xc}^{LDA}[\rho] = \int \rho(\boldsymbol{r})\epsilon_{xc}(\rho(\boldsymbol{r}))\, d\boldsymbol{r} \tag{7}$$

Such that ϵ is the XC energy density. Many of these approximations are based on homogeneous electron gases (HEG, also referred to as Jellium), for which the electron density is exactly known. From this, we may determine the XC energy by adding the exchange and correlation parts. The exchange energy density is known exactly, and thus can be given exactly as:

$$E_x^{LDA} = -\frac{3}{4}\left(\frac{3}{\pi}\right)^{\frac{1}{3}} \int \rho(\boldsymbol{r})^{\frac{4}{3}}\, d\boldsymbol{r} \tag{8}$$

Determination of the correlation energy density is the cause of ongoing research into LDAs, as only the upper and lower bounds are known analytically:

$$\epsilon_{c,\text{upper}} = A\ln(r_s) + B + r_s(C\ln(r_s) + D) \tag{9}$$

$$\epsilon_{c,\text{lower}} = \frac{1}{2}\left(\frac{g_0}{r_s} + \frac{g_1}{r_s^{3/2}} + \cdots\right) \tag{10}$$

Such that r_s is the Wigner-Seitz parameter. Thus, a variety of methods (quantum Monte Carlo simulations, among others) are used for determining values between these bounds. This approximation may also be extended to spin-polarised systems, however this is beyond the scope of this report. A discussion on the application of spin-polarised systems to LDA and thus DFT can be found in Appendix A.2. LDAs, however, suffer from an underestimation of the exchange energy density and an overestimation of the correlation energy density. As such, generalised gradient approximations are used to provide greater accuracy, by assuming that the XC energy functional is described by the electron density $\rho(\boldsymbol{r})$ and the gradient of the electron density, such that:

$$E_{xc}^{GGA}[\rho] = \int \rho(\boldsymbol{r})\epsilon_{xc}\left(\rho(\boldsymbol{r}), \nabla\rho(\boldsymbol{r})\right)\, d\boldsymbol{r} \tag{11}$$

The inclusion of the gradient of $\rho(\boldsymbol{r})$ in GGA compared to LDA allows us to describe GGA as a semi-local approximation, rather than a local approximation. This property is useful in ensuring that the energy density approximations (specifically the correlation energy density) is accurate when interpolating. In general, for the majority of calculations, GGAs are perfectly suitable, however in certain cases, further accuracy is required. For these cases, a further expansion in terms of the second derivative (the Laplacian) of $\rho(\boldsymbol{r})$ in the XC energy functional is used, generally referred to as Meta-GGAs.

Hybrid XC functionals represent a different approach, in using the exact exchange energy determined by Hartree-Fock theory[2]. This exact exchange energy is given by the following equation:

$$E_x^{\mathrm{HF}} = -\frac{1}{2}\sum_{i,j}\iint \psi_i^*(\boldsymbol{r}_1)\psi_j^*(\boldsymbol{r}_2)\frac{1}{r_{12}}\psi_j(\boldsymbol{r}_1)\psi_i(\boldsymbol{r}_2)\, d\boldsymbol{r}_1\, d\boldsymbol{r}_2 \tag{12}$$

Where ψ is a one-particle wavefunction. Using this, a very accurate approximation of the XC energy may be determined. For the purposes of this report, the 'LibXC' library will be used, which contains the majority of the commonly used XC energy approximations for use in DFT, which the user may select.

[2]The majority of hybrid XC functionals (some notable examples being PBE0, B3LYP and SCAN) use the exact exchange energy from Hartree-Fock theory *in conjunction with* the exchange energy determined by some form of approximation.

5.1.2 Time-dependence

In a similar manner as ground-state DFT, time dependent DFT has its foundation in the Runge-Gross theorem, which can be viewed as an extension of the Hohenberg-Kohn theorems to time-dependent systems. The first Runge-Gross theorem is given as the following [8]:

Theorem 3 *For every single-particle potential $v(\boldsymbol{r}, t)$ which can be expanded into a Taylor series with respect to the time coordinate around $t = t_0$, a map $G : v(\boldsymbol{r}, t) \rightarrow \rho(\boldsymbol{r}, t)$ is defined by solving the time-dependent Schrödinger equation with a fixed initial state $\Phi(t_0) = \Phi_0$ and calculating the corresponding densities $\rho(\boldsymbol{r}, t)$. This map can be inverted up to an additive merely time-dependent function in the potential.*

This theorem, which is proven in Appendix A.3, implies that there is a one-to-one mapping between the many-body wavefunction $\Psi(\boldsymbol{r}, t)$ and the electron density $\rho(\boldsymbol{r})$ over all time, such that:

$$\rho(\boldsymbol{r}, t) = \sum_{n=1}^{N} |\psi_n^{KS}(\boldsymbol{r}, t)|^2 \tag{13}$$

As such, the system may be entirely described by the electron density and time and thus, when solving the time-dependent Schrödinger equation, we can expect the Hamiltonian operator to be a functional of these variables exclusively. As shown in Equation (2), the Hamiltonian operator used in the Time-dependent Kohn-Sham (TDKS) equation has either an explicit or implicit dependance on time. Thus, we may construct the Hamiltonian used in the TDKS equation in the following form:

$$\hat{H}^{KS}[\rho(t), t] = \hat{T}_{el} + \hat{V}_{ext}(\boldsymbol{r}, t) + \hat{V}_H[\rho(t)] + \hat{V}_{xc}[\rho(t)] \tag{14}$$

Where the external potential $\hat{V}_{ext}(\boldsymbol{r}, t)$ may or may not have some explicit dependance on time. While it may be technically possible to evaluate each of these terms with respective time dependance, there are some approximations which may be made to dramatically simplify the determination of the Hamiltonian.

We first turn out attention to the XC potential term, which has an implicit dependance on time via the electron density $\rho(t)$. This term changes very slowly given a change in t, and thus, can be considered to be approximately adiabatic, based on the following definition of an adiabatic process:

Definition 4 *If process exchanges no heat, and therefore energy, within a given system, then that process may be described as **adiabatic**.*

We therefore apply the adiabatic approximation to the XC potential, given the short timescales used in TDDFT calculations. Thus, we may write the approximated XC potential as depending exclusively on the instantaneous electron density, such that:

$$\hat{V}_{xc}^{adia}[\rho](t) = \hat{V}_{xc}^{approx}[\rho(t)] \tag{15}$$

We may now consider the external potential $\hat{V}_{ext}(\boldsymbol{r}, t)$, which is commonly a small electric field pulse, and thus can be described by the following:

$$\hat{V}_{ext} = \hat{\boldsymbol{r}} \cdot \mathbf{E}(\boldsymbol{r}, t) \tag{16}$$

Such that $\mathbf{E}(\boldsymbol{r}, t)$ has a plane wave solution:

$$\mathbf{E}(\boldsymbol{r}, t) = E_0 e^{i(\boldsymbol{k}\cdot\boldsymbol{r} - \omega t)} \tag{17}$$

The modulus of the wavevector $\boldsymbol{k}$ is related to the wavelength of the electric field by the following function:

$$|\boldsymbol{k}| = \frac{2\pi}{\lambda} \tag{18}$$

It can be shown that if the wavelength of the field is significantly larger than the size of the molecule, $|\boldsymbol{k}|$ becomes very small, and given that $|\boldsymbol{r}|$ is small, $\boldsymbol{k} \cdot \boldsymbol{r} \approx 0$. As such, we may then find that the equation for $\mathbf{E}(\boldsymbol{r}, t)$ becomes:

$$\mathbf{E}(\boldsymbol{r}, t) = E_0 e^{i\boldsymbol{k}\cdot\boldsymbol{r}} e^{-i\omega t} \approx E_0 e^{-i\omega t} \tag{19}$$

Thus, we may assume that:

$$\mathbf{E}^{approx}(\boldsymbol{r}, t) \approx \mathbf{E}(t) \tag{20}$$

This approximation is referred to as the electric dipole approximation and removes any spatial dependance from the electric field equation. The Kohn-Sham Hamiltonian may thus be approximated as:

$$\hat{H}^{KS}[\rho(t), t] \approx \hat{T}_{el} + \hat{V}_{ext}(t) + \hat{V}_H[\rho(t)] + \hat{V}_{xc}[\rho](t) \tag{21}$$

It is now easy to see that the remaining time-dependance of each of the terms in $\hat{H}^{KS}$ is explicit in $\hat{V}_{ext}$ and implicit in $\hat{V}_H$. These approximations now allow for a more simple computational method to be described, with the purpose of comparing the implementation of TDDFT in CP2K to a purpose-built Python script. In doing this, the intention is to find improvements that may be applicable to CP2K.

5.2 Computational TDDFT Method

The Python script written to perform TDDFT calculations, henceforth referred to as 'PyRTP', is given in Appendix A.5. In this section, the computational method and justifications for programming decisions will be given.

For the purposes of this report, a focus has been put specifically on real-time TDDFT (RT-TDDFT), as opposed to linear response TDDFT (LR-TDDFT). This decision has a few motivations, but primarily that RT-TDDFT is applicable to a wider variety of systems and has the capability of providing a greater amount of information, given the calculation of the electron density at all timesteps. Thus, we shall save discussions of LR-TDDFT to Appendix A.6. An extremely simplified overview of the RT-TDDFT method can be described as the following steps:

1. Compute the ground state energy and density matrix of the system using conventional DFT
2. Apply a small perturbation to the system to ensure evolution in time
3. Propagate the density matrix to the next timestep
4. Determine and record energy and any required information
5. Repeat steps 3 and 4 for all timesteps
6. Post-process the recorded data

These steps will be explained in the following sections.

5.2.1 DFT calculations

The DFT function in PyRTP is adapted from the DFT function written by Dr Warren Lynch [16], which is adapted from the method described in Modern Quantum Chemistry by Szabo and Ostlund [17], as well as inspection of the CP2K DFT method [9]. This method begins by defining the atomic position vector $\boldsymbol{R_I}$, the atomic numbers $\boldsymbol{Z_I}$ and an initial guess for the density matrix $\boldsymbol{P}$. We then seek to define a set of atomic basis functions ϕ_i to approximate the true atomic orbitals. In the case of this program, STO-3G basis functions are used as a low computational cost and reasonably accurate approximation, which is formed using a linear combination of 3 Gaussian-type 1s orbitals (GTOs), as described in the following equations:

$$\phi_{1s}^{GTO} = \left(\frac{2\alpha}{\pi}\right)^{\frac{3}{4}} e^{-\alpha|\boldsymbol{r}-\boldsymbol{R_I}|^2} \tag{22}$$

Such that α is the Gaussian orbital exponent (which is related to the relative atomic charge). The STO-3G orbital is determined using the following function:

$$\phi_{1s}^{STO-3G} = \sum_{\lambda=1}^{\lambda=3} C_\lambda \phi_{1s,\lambda}^{GTO} \tag{23}$$

Such that C is the contribution of each GTO to the STO-3G approximation. This function is evaluated over a grid of dimensions L^3, containing N points per unit length. From then, the integrals required to compute the Kohn-Sham (KS) matrix can be determined, which is given by the following:

$$KS_{\mu\nu} = \frac{\delta T_{\mu\nu}}{\delta \boldsymbol{P}_{\mu\nu}} + \boldsymbol{V}_{\mu\nu}^{\mathrm{H}} + \boldsymbol{V}_{\mu\nu}^{\mathrm{XC}} + \boldsymbol{V}_{\mu\nu}^{\mathrm{SR}} \tag{24}$$

Such that $\frac{\delta T_{\mu\nu}}{\delta \boldsymbol{P}_{\mu\nu}}$ is the derivative of the kinetic energy with respect to the density matrix $\boldsymbol{P}$, $\boldsymbol{V}_{\mu\nu}^{\mathrm{H}}$ is the Hartree potential, $\boldsymbol{V}_{\mu\nu}^{\mathrm{XC}}$ is the exchange-correlation potential and $\boldsymbol{V}_{\mu\nu}^{\mathrm{SR}}$ is the short range local pseudopotential. Many of these calculations involve the use of the reciprocal space, which is more easily described with a plane wave basis function, as opposed to a Gaussian basis function. We may determine these plane wave representations using the following formula:

$$\phi^{PW}(\boldsymbol{G}) = \sum_{\boldsymbol{r}} \phi^{STO-3G}(\boldsymbol{r})\, e^{i\boldsymbol{G}\cdot\boldsymbol{r}} \tag{25}$$

Such that $\boldsymbol{G}$ is the reciprocal space vector[3]. Given this, $\frac{\delta T_{\mu\nu}}{\delta \boldsymbol{P}_{\mu\nu}}$ may now be determined using the following equation:

$$\frac{\delta T_{\mu\nu}}{\delta \boldsymbol{P}_{\mu\nu}} = \frac{L^3}{2N^2} \sum_{\boldsymbol{G}} \boldsymbol{G}^2\, \phi_\mu^{*\,PW}(\boldsymbol{G})\, \phi_\nu^{PW}(\boldsymbol{G}) \tag{26}$$

A full derivation of this function is given in Appendix A.7. Calculation of the next term, the Hartree potential, requires determination of the charge density $n(\boldsymbol{r})$, which is given as the sum of the following two terms:

$$n_{el}(\boldsymbol{r}) = \sum_{\mu\nu} \boldsymbol{P} \phi_\mu^{STO-3G}(\boldsymbol{r}) \phi_\nu^{STO-3G}(\boldsymbol{r}) \tag{27}$$

$$n_c(\boldsymbol{r}) = \sum_I -\frac{\boldsymbol{Z}_I}{(\boldsymbol{R}_I^c)^3} \pi^{-3/2} \exp\left[-\left(\frac{\boldsymbol{r} - \boldsymbol{R}_I}{R_I^c} \right)^2 \right] \tag{28}$$

Such that $n_{el}(\boldsymbol{r})$ is the electron charge density, $n_c(\boldsymbol{r})$ is the nuclear charge density and R_I^c is a pseudopotential parameter which varies depending on the molecule under

[3]The reciprocal lattice vectors are determined using $\boldsymbol{b}_1 = \frac{2\pi}{V} \boldsymbol{a}_2 \times \boldsymbol{a}_3$, $\boldsymbol{b}_2 = \frac{2\pi}{V} \boldsymbol{a}_3 \times \boldsymbol{a}_1$ and $\boldsymbol{b}_3 = \frac{2\pi}{V} \boldsymbol{a}_1 \times \boldsymbol{a}_2$, such that $\boldsymbol{G} = h\, \boldsymbol{b}_1 + k\, \boldsymbol{b}_2 + l\, \boldsymbol{b}_3$ and V is the scalar triple product.

study. The Hartree potential most easily determined in reciprocal space, and thus, the reciprocal charge density is found by computing the Fourier transform of $n(\boldsymbol{r})$, such that:

$$n_c(\boldsymbol{G}) = \mathscr{F}\{n(\boldsymbol{r})\} \tag{29}$$

The Hartree potential is reciprocal space is then determined using the following equation, with a derivation given in Appendix A.8:

$$V_H(\boldsymbol{G}) = 4\pi \frac{\rho(\boldsymbol{G})}{\boldsymbol{G}^2} \tag{30}$$

We may then finally find the Hartree potential in real space by computing the inverse Fourier transform of $V(\boldsymbol{G})$, such that:

$$V(\boldsymbol{r}) = \mathscr{F}^{-1}\{V(\boldsymbol{G})\} \tag{31}$$

As stated in the theoretical section, the 'LibXC' library is used to determine $V^{\mathrm{XC}}(\boldsymbol{r})$, with the XC functional being selectable by the user. The final term, the short range local pseudopotential, is determined using the following equation:

$$V^{SR}(\boldsymbol{r}) = \sum_{I=0}^{1} \sum_{i=1}^{2} C_{I,i}^{PP} (R_I^c |\boldsymbol{r} - \boldsymbol{R}_I|)^{2i-2} \tag{32}$$

Such that $C_{I,i}^{pp}$ is a predefined pseudopotential parameter matrix.

Each of these potential terms (all besides the kinetic energy derivative) are evaluated at all points of the grid, and thus are not *total* potentials. Thus, to convert from an arbitrary potential to a total potential, we may integrate over all points on the grid using the following formula:

$$\boldsymbol{V}_{\mu\nu} = \sum_{\boldsymbol{r}} V(\boldsymbol{r}) \phi_\mu^{STO-3G}(\boldsymbol{r}) \phi_\nu^{STO-3G}(\boldsymbol{r}) \tag{33}$$

As we now know the Kohn-Sham matrix, for a given initial guess of $\boldsymbol{P}$, we may now perform an SCF method to converge upon the the correct $\boldsymbol{P}$. For this, we must determine the overlap matrix $\boldsymbol{S}$, which is given by the following equation:

$$\boldsymbol{S}_{\mu\nu} = \langle \phi_\mu^{STO-3G}(\boldsymbol{r}) \mid \phi_\nu^{STO-3G}(\boldsymbol{r}) \rangle \tag{34}$$

An approximation that the integral $\int \phi_\mu^{*\,STO-3G}(\boldsymbol{r})\, \phi_\nu^{STO-3G}(\boldsymbol{r})\, d\boldsymbol{r}$ is approximately equal to the sum of the function evaluated at all grid points. This allows us to determine a transformation matrix $\boldsymbol{X}$ to orthogonalise the Kohn-Sham matrix, turning the problem into an eigenvalue problem, using the following:

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{s}^{-1/2} \tag{35}$$

Such that $\boldsymbol{s}$ is a diagonal matrix of eigenvalues and $\boldsymbol{U}$ is a unitary matrix, such that $\boldsymbol{S} = \boldsymbol{U}\boldsymbol{s}\boldsymbol{U}^{-1}$. These matrices are determined by diagonalising the overlap matrix in SymPy. We can now determine the transformed Kohn-Sham matrix to be the following:

$$KS' = \boldsymbol{X}^{\dagger} KS\, \boldsymbol{X} \tag{36}$$

We may then repeat the process of diagonalisation, this time acting on the Kohn-Sham matrix, in order to retrieve the transformed matrix of coefficients $\boldsymbol{C'}$, such that $KS' = \boldsymbol{C'}\boldsymbol{\varepsilon}\,\boldsymbol{C'}^{-1}$. The $\boldsymbol{C'}$ matrix is then multiplied by the transformation matrix $\boldsymbol{X}$ to give the true matrix of coefficients $\boldsymbol{C}$. The adjusted density matrix is then obtained using the following equation:

$$\boldsymbol{P}_{\mu\nu} = 2\sum_{a=0}^{1} \boldsymbol{C}_{\mu a}\boldsymbol{C}^{*}_{\nu a} \tag{37}$$

This process is then repeated until convergence is reached. The total energy E_0 may also be determined during each step of the SCF loop using the following function:

$$E_0 = T + E_H + E_{XC} + E_{SR} + E_{self} + E_{II} \tag{38}$$

Such that T is the kinetic energy, E_H is the Hartree energy, E_{XC} is the exchange-correlation energy, E_{SR} is the short range pseudopotential energy, E_{self} is the nuclear core self interaction energy correctio and E_{II} is the ion-ion interaction energy. The terms T, E_H, E_{XC} and E_{SR} can be determined from the potentials calculated for the Kohn-Sham matrix using the following equation:

$$E = \sum_{\mu\nu} \boldsymbol{P}^{\mu\nu}\boldsymbol{V}_{\mu\nu} \tag{39}$$

This leaves two terms, E_{self} and E_{II}, which can both be calculated using the following equations:

$$E_{self} = -\sum_{I} \frac{1}{\sqrt{2\pi}} \frac{\boldsymbol{Z}_I^2}{R_I^c} \tag{40}$$

$$E_{II} = \sum_{j}^{N}\sum_{i\neq j}^{N} \left(\frac{Z_i Z_j}{|R_i - R_j|} \operatorname{erfc}\left[\frac{|R_i - R_j|}{\sqrt{2R_I^{c\,2}}} \right] \right) \tag{41}$$

Such that erfc(z) is the complementary error function, such that erfc$(z) = \frac{2}{\sqrt{\pi}}\int_z^{\infty} e^{-t^2}\, dt$.

Following this method, the density matrix and energy for the ground state is determined simply, allowing for propagation of $\boldsymbol{P}$. The next step in this method is to apply a small external perturbation, as described in Section 5.2.2.

5.2.2 Applying a small perturbation

As described by the Runge-Gross theorem, there is a one-to-one mapping between the applied external potential causing the system to evolve and the electron density $\rho(\boldsymbol{r}, t)$. Thus, to enforce this evolution in time, we apply a small 'kick' in the form of a short electric field pulse with a Gaussian envelop. As proven in the theoretical section, the applied electric field $E(t)$ has no spatial dependance, based on the dipole approximation, and thus, we use the following formula to describe the perturbing field:

$$\boldsymbol{E}(t) = E_0 e^{-\frac{(t-t_0)^2}{2t_w^2}} \cdot \hat{d} \tag{42}$$

Such that E_0 is the electric field strength, t_0 is the centre of pulse, t_w is the pulse width and $\hat{d}$ is the polarisation of the pulse. We may then determine the applied potential $\boldsymbol{V}_{app}$ using the following:

$$\boldsymbol{V}_{app} = -\boldsymbol{D}_{\mu\nu} \cdot \boldsymbol{E}(t) \tag{43}$$

Such that $\boldsymbol{D}_{\mu\nu}$ is the transition dipole tensor, which is equal to the following integral:

$$\boldsymbol{D}_{\mu\nu} = \sum_{x,y,z} \left(\int \phi_\mu^*(\boldsymbol{r})\, q_i\, \phi_\nu(\boldsymbol{r})\, d\boldsymbol{r} \right) \tag{44}$$

Such that q_i is a generalised coordinate. By evaluating the function at each point of the grid, we may evaluate the integral as a sum of the following form:

$$D_x = \sum_{\boldsymbol{r}} \phi_\mu^{STO-3G}(\boldsymbol{r})\, x\, \phi_\nu^{STO-3G}(\boldsymbol{r}) \tag{45}$$

This applied potential may then be added to the Kohn-Sham matrix to give the perturbed Kohn-Sham matrix, in the following form:

$$\boldsymbol{KS}' = \boldsymbol{KS} + \boldsymbol{V}_{app} \tag{46}$$

The following script was written to compute the transition dipole tensors in $\{x, y, z\}$, as well as the total transition dipole tensor, which may be used in later calculations:

```
def transition_dipole_tensor_calculation(r_x,r_y,r_z,CGF_He,CGF_H,dr)
 ↪  :

    D_x = [[0.,0.],[0.,0.]]
    for i in range(0,len(r_x)) :
        for j in range(0,len(r_x)) :
            for k in range(0,len(r_x)) :

                #Integrate over grid points
                D_x[0][0] += r_x[i]*CGF_He[i][j][k]**2*dr
                D_x[0][1] += r_x[i]*CGF_He[i][j][k]*CGF_H[i][j][k]*dr
                D_x[1][0] += r_x[i]*CGF_H[i][j][k]*CGF_He[i][j][k]*dr
                D_x[1][1] += r_x[i]*CGF_H[i][j][k]**2*dr

    D_y = [[0.,0.],[0.,0.]]
    for i in range(0,len(r_y)) :
        for j in range(0,len(r_y)) :
            for k in range(0,len(r_y)) :

                #Integrate over grid points
                D_y[0][0] += r_y[i]*CGF_He[i][j][k]**2*dr
                D_y[0][1] += r_y[i]*CGF_He[i][j][k]*CGF_H[i][j][k]*dr
                D_y[1][0] += r_y[i]*CGF_H[i][j][k]*CGF_He[i][j][k]*dr
                D_y[1][1] += r_y[i]*CGF_H[i][j][k]**2*dr

    D_z = [[0.,0.],[0.,0.]]
    for i in range(0,len(r_z)) :
        for j in range(0,len(r_z)) :
            for k in range(0,len(r_z)) :

                #Integrate over grid points
                D_z[0][0] += r_z[i]*CGF_He[i][j][k]**2*dr
                D_z[0][1] += r_z[i]*CGF_He[i][j][k]*CGF_H[i][j][k]*dr
                D_z[1][0] += r_z[i]*CGF_H[i][j][k]*CGF_He[i][j][k]*dr
                D_z[1][1] += r_z[i]*CGF_H[i][j][k]**2*dr

    D_tot=D_x+D_y+D_z

    return D_x,D_y,D_z,D_tot
```

The following Python function was used to compute the Gaussian pulse and apply this to the Kohn-Sham matrix, with the electric field strength and pulse parameters suggested in the Lopata and Govind paper, published in 2011 [18]:

```
def GaussianKick(KS,scale,direction,t,r_x,r_y,r_z,CGF_He,CGF_H,dr):
    t0=3
    w=0.2
    Efield=np.dot(scale*np.exp((-(t-t0)**2)/(2*(w**2))),direction)
```

```
D_x,D_y,D_z,D_tot=transition_dipole_tensor_calculation(r_x,r_y,
 ↪  r_z,CGF_He,CGF_H,dr)
V_app=-(np.dot(D_x,Efield[0])+np.dot(D_y,Efield[1])
 ↪  +np.dot(D_z,Efield[2]))
KS_new=KS+V_app
return KS_new
```

The use of a Gaussian pulse is significantly more simple than the more commonly used ‘delta pulse’ method, in which the electric field $\boldsymbol{E}(t)$ is given by the following equation:

$$\boldsymbol{E}(t) = E_0\delta(t)\hat{d} \tag{47}$$

Such that $\delta(t)$ is the Dirac delta distribution. This requires a slightly more complicated input file when replicating in CP2K, however this reduces the complexity of the PyRTP program significantly, as using the Dirac delta distribution requires solving the following equation numerically:

$$\delta(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{ikx}\, dk \tag{48}$$

In general, we require the strength of this perturbation E_0 to be small enough such that the Taylor expansion of the perturbed electron density $\rho(\boldsymbol{r},t)$ is expressed only up to the first order, such that [19]:

$$\rho(\boldsymbol{r},t) = \rho_{GS}(\boldsymbol{r}) + \rho_1(\boldsymbol{r},t) + \cdots \tag{49}$$

Such that:

$$\rho_1(\boldsymbol{r},t) = \int_0^{\infty} dt' \int d^3r'\chi(\boldsymbol{r}t,\boldsymbol{r}'t')\delta v_{ext}(\boldsymbol{r}'t') \tag{50}$$

Where:

$$\chi(\boldsymbol{r}t,\boldsymbol{r}'t') = \left.\frac{\delta\rho(\boldsymbol{r},t)}{\delta v_{ext}(\boldsymbol{r}'t')}\right|_{v_{ext},\,0} \tag{51}$$

This is for the purpose of determining useful information in Section 7, as the response produced by perturbing the system will not be smooth given a large field strength. This consideration is more critical in linear response TDDFT (LR-TDDFT), however the consideration should still be made for RT-TDDFT. A method to determine whether the perturbation strength is suitable will be described in Section 7.4.

5.2.3 Propagation methods

As we seek to solve the time-dependent Kohn-Sham equation numerically, we may first take some time to write the TDKS equation in a useful form. As a reminder, from Equation 2, we have that:

$$i\frac{\partial}{\partial t}\Psi_n^{KS}(\boldsymbol{r},t) = \hat{H}^{KS}[\rho(t),t]\Psi_n^{KS}(\boldsymbol{r},t)$$

We now seek some link between the current time t and the initial time t_0. We thus fix the wavefunction on the right hand side to be at $t = 0$. We may then integrate with respect to t between t and t_0 for n timesteps between them. We then find the equation to be of the following form [20]:

$$\Psi^{KS}(\boldsymbol{r},t) = \sum_{n=0}^{\infty}\frac{(-i)^n}{n!}\int_{t_0}^{t}\mathrm{d}t_1\int_{t_0}^{t}\mathrm{d}t_2\ldots\int_{t_0}^{t}\mathrm{d}t_n\,\hat{T}\left\{\hat{H}^{KS}[\rho(t_1),t_1]\hat{H}^{KS}[\rho(t_2),t_2]\ldots \hat{H}^{KS}[\rho(t_n),t_n]\right\}\Psi^{KS}(\boldsymbol{r},t_0) \quad (52)$$

We may now simplify this using a change of variable, such that the operator acting on $\Psi^{KS}(\boldsymbol{r},t_0)$, $\hat{U}(t,t_0)$ is given by the following:

$$\hat{U}(t,t_0) = \sum_{n=0}^{\infty}\frac{(-i)^n}{n!}\int_{t_0}^{t}\mathrm{d}\tau\,\hat{T}\left\{\hat{H}^{KS}[\rho(\tau),\tau]\right\} \quad (53)$$

The following definition of the exponential function is used to simplify this further:

$$\exp x = \sum_{n=0}^{\infty}\frac{x^n}{n!} \quad (54)$$

Using this, we may write $\hat{U}(t,t_0)$ as the following:

$$\hat{U}(t,t_0) = \exp\left(-i\int_{t_0}^{t}\mathrm{d}\tau\,\hat{T}\left\{\hat{H}^{KS}[\rho(\tau),\tau]\right\}\right) \quad (55)$$

Such that $\hat{T}$ is the time ordering operator, which is often written in from of the exponential in the following form:

$$\hat{U}(t,t_0) = \hat{T}\exp\left(-i\int_{t_0}^{t}\mathrm{d}\tau\,\left\{\hat{H}^{KS}[\rho(\tau),\tau]\right\}\right) \quad (56)$$

This is the exact propagator, which we now seek to approximate as closely as possible, given that we cannot know the exact form of the function $\hat{U}(t,t_0)$ [7]. In this case, we have a few specific requirements of a proposed approximate propagator, some of which are necessary, and some of which are for computational ease.

5.2.3.1 Requirements of suitable propagators

The requirements we seek for a proposed propagator are as follows:

- **The propagator must be unitary**
  Given that we seek a propagator such that $\Psi^{KS}(\boldsymbol{r}, t) = \hat{U}(t, t_0)\Psi^{KS}(\boldsymbol{r}, t_0)$, and given that we assume that there is a one-to-one mapping between the wavefunction and the electron density, we seek a propagator to act on the electron density, such that:

  $$\boldsymbol{P}(t + \Delta t) = \boldsymbol{U}(t + \Delta t, t)\, \boldsymbol{P}(t)\, \boldsymbol{U}^{\dagger}(t + \Delta t, t) \tag{57}$$

  As such, we seek a propagator that is unitary, such that [20]:

  $$\boldsymbol{U}^{\dagger}(t + \Delta t, t) = \boldsymbol{U}(t + \Delta t, t) \tag{58}$$

  Given that the dagger operator † denotes the Hermitian conjugate.

- **The propagator should have time-reversal symmetry**
  In the case of the majority of physical systems, a given system should behave identically regardless of the flow of time. A system of particles perturbed by an electric field preserves time-reversal symmetry, and thus we seek a propagator which does this also. If this is the case, the inverse of the propagator matrix is equal to the propagator matrix itself, and may be described by the following [20]:

  $$\boldsymbol{U}^{-1}(t + \Delta t, t) = \boldsymbol{U}(t + \Delta t, t) \tag{59}$$

  There are many cases in which this requirement is not necessary (such as if the system is perturbed by a magnetic field), and thus generally quantum chemistry programs give a range of options for propagators, in order to pick one best suited to the system being modelled. However, for long timescale simulations, time reversal symmetry is significantly more important.

- **The propagator must be symplectic**
  Liouville's theorem states that [21]:

  Definition 5 *The Hamiltonian flow in phase space is incompressible, such that:*

  $$div\,(\boldsymbol{v}) = 0 \tag{60}$$

  Such that $\boldsymbol{v} = (\dot{x}, \dot{p})$.

  From this, for a simulated system, we expect the volume of phase space $\boldsymbol{dx} \wedge \boldsymbol{dp}$ to not change. For this, we require a propagator that changes canonical coordinates without changing Hamilton's equations. This requirement removes the possibility of using integrators such as the Euler method or Runge-Kutta methods.

- **The propagator should be computationally affordable**
  This suggestion allows for suggestions of suitable propagators for given systems, such as propagators which do not preserve time-reversal symmetry (and thus are more computationally affordable) being best suited to systems perturbed by a magnetic field.

5.2.3.2 Commonly used propagators

As we have now considered the requirements of a proposed propagator, we may now discuss some common choices:

- **Crank-Nicholson propagator**
  The Crank-Nicholson propagator is given by the following equation [22]:

$$\hat{U}^{CN}(t+\Delta t, t) = \frac{1 - i\frac{\Delta t}{2}\hat{H}(t+\Delta t/2)}{1 + i\frac{\Delta t}{2}\hat{H}(t+\Delta t/2)} \tag{61}$$

  Such that the Hamiltonian at time $t + \Delta t/2$ is determined via extrapolation. The method used to determine this in PyRTP will be described in Section 5.2.3.3. Computationally, this method is particularly efficient, given the relative simplicity of the equation. Time-reversal symmetry is not, however, preserved, limiting the usefulness in many cases.

- **Exponential Midpoint propagator**
  The EM propagator is given as the following [7]:

$$\hat{U}^{EM}(t+\Delta t, t) = \exp\Big(-i\Delta t\hat{H}(t+\Delta t/2)\Big) \tag{62}$$

  This propagator is relatively popular, as if a quantum chemistry program has good methods to calculate exponentials, it tends to be more computationally efficient than Crank-Nicholson. Time reversal symmetry is also not necessarily preserved, and requires $\hat{H}(t + \Delta t/2)$ to be exact.

- **Enforced Time-Reversal Symmetry (ETRS) propagator**
  Rather than requiring the Hamiltonian at $t+\Delta t/2$ to be exact in order to preserve time-reversal symmetry, we may enforce it using the ETRS propagator [23]:

$$\hat{U}^{ETRS}(t+\Delta t, t) = \exp\left\{-t\frac{\Delta t}{2}\hat{H}(t+\Delta t)\right\}\exp\left\{-t\frac{\Delta t}{2}\hat{H}(t)\right\} \tag{63}$$

  It should be noted that the ETRS propagator contains the Hamiltonian at the next timestep, which often must be determined exactly, and thus via another propagator, such as EM or CN. As such, this comes with a significant increase in computational cost. There are several methods to reduce this cost, mainly revolving around extrapolating the Hamiltonian at the next timestep.

- **Approximated and Corrected-Approximated ETRS propagators**
  As expressed above, approximating the Hamiltonian at the next timestep is generally done by extrapolation. In this case, we approximate the Hamiltonian using Lagrange interpolation, although other methods may be used. This significantly reduces the computational cost of ETRS, with preserved time-reversal symmetry, at the cost of some accuracy. As such, methods to correct this approximation are referred to as corrected-approximated ETRS (or CAETRS). One such algorithm, the 'predictor-corrector' algorithm, will be discussed in Section 5.2.3.3.

- **Magnus propagators**
  Magnus propagators revolve around various order expansions of the Magnus operator, which was first described in 1954 by Wilhelm Magnus as an exact solution to the following equation [24]:

  $$\hat{U}(t+\Delta t, t) = \exp\left(\hat{\Omega}(t+\Delta t, t)\right) \tag{64}$$

  Such that:

  $$\hat{\Omega}(t+\Delta t, t) = \sum_{m=1}^{\infty} \hat{\Omega}_m(t+\Delta t, t) \tag{65}$$

  Deriving each order of the Magnus operator is beyond the scope of this report, however some common expansions shall be listed. The second order Magnus operator (often shortened to $M(2)$) is equivalent to the Exponential Midpoint propagator, and thus is rather simple to describe. Another commonly used method is the commutator-free 4th order Magnus expansion (shortened to CFM4), which takes the following form:

  $$\begin{aligned}\hat{U}^{CFM4}(t+\Delta t, t) = \exp&\left(-i\Delta t\alpha_1\hat{H}_{t_1} - i\Delta t\alpha_2\hat{H}_{t_2}\right)\\ &\times \exp\left(-i\Delta t\alpha_2\hat{H}_{t_1} - i\Delta t\alpha_1\hat{H}_{t_2}\right)\end{aligned} \tag{66}$$

  Such that t_1 and t_2 are carefully chosen intermediate times , generally $t_1 = t + \left(\frac{1}{2} - \frac{\sqrt{3}}{6}\right)$ and $t_2 = t + \left(\frac{1}{2} + \frac{\sqrt{3}}{6}\right)$, and α_1 and α_2 are well selected constants. Further expansions may be found in the 2018 paper on the topic by Auer et al. [25].

This is not a comprehensive list of propagators, but rather a summary of commonly used methods. Each of these propagators has most suitable applications, and thus a robust RT-TDDFT program should have the ability to switch between them. As such, this requirement was considered heavily during the design of PyRTP.

5.2.3.3 Implementation of propagators in PyRTP

PyRTP currently has 6 selectable propagators: Crank-Nicholson, Exponential Midpoint, ETRS, AETRS, CAETRS and CFM4. In the case of CN, EM and CAETRS, to further improve the accuracy, a modified predictor-corrector regime is used, as is relatively common [26]. It is reasonable to omit this step for EM and CN, however a predictor-corrector regime does increase the accuracy, whilst impacting computational cost minimally. Using a method adapted from the Hekele et al. paper published in 2021 [27], the following process was used:

1. Predict the Hamiltonian at $t + a\Delta t/2$
   For this, Lagrange interpolation is used on the set of Hamiltonians at previous timesteps, via the '*scipy.interpolate.lagrange*' function:

   ```
   def LagrangeExtrapolate(t,H,tnew):
           f=lagrange(t,H)
           return f(tnew)
   ```

   Further information on the Lagrange interpolation method can be found in Appendix A.9. Using this, a reasonable approximation of $H(t + a\Delta t/2)$ can be made. We refer to this as the predictor Hamiltonian, as we may now use this to determine the corrector Hamiltonian via the DFT method. Before this, we must calculate some other variables.

2. Determine the predictor C matrix
   We may find the updated $\boldsymbol{C}$ matrix by applying the chosen propagator to the current $\boldsymbol{C}$ matrix, such that:

$$\boldsymbol{C}^p = \boldsymbol{U}^p(t + \Delta t, t)\boldsymbol{C}(t) \tag{67}$$

3. Determine the predictor density matrix
   As shown by Equation 37, the new density matrix $\boldsymbol{P}^p$ may now be determined from $\boldsymbol{C}^p$.

4. Use $\boldsymbol{P}^p$ to compute the corrector energy
   The following function was used to compute the energy at $t + \Delta t$ for any given $\boldsymbol{P}$:

   ```
   def computeE_0(R_I,Z_I,P,N_i,Cpp,r_x,r_y,r_z,N,dr,CGF_He,CGF_H,
    ↪  G_u,G_v,G_w,delta_T,E_self,E_II,L):
       n_el_r, n_el_r_tot  =
        ↪  calculate_realspace_density(CGF_He,CGF_H,N,N_i,P,dr)
       n_c_r = calculate_core_density(N,N_i,Z_I,r_x,r_y,r_z,R_I)
   ```

```
    n_r = n_el_r + n_c_r
    T = energy_calculation(delta_T,P)
    n_G = np.fft.fftn(n_r)
    V_G, E_hart_G = calculate_hartree_reciprocal(n_G,N,N_i,r_x,
     ↪  r_y,r_z,G_u,G_v,G_w,L)
    V_r = np.fft.ifftn(V_G)
    V_hart = grid_integration(V_r,dr,CGF_He,CGF_H)
    E_hart_r = calculate_hartree_real(N_i,V_r,n_r,dr)
    V_XC_r = calculate_XC_potential(N,N_i,n_el_r)
    V_XC = grid_integration(V_XC_r,dr,CGF_He,CGF_H)
    E_XC = calculate_XC_energy(N_i,n_el_r,dr)
    V_SR_r = calculate_V_SR_r(N,N_i,r_x,r_y,r_z,Z_I,Cpp,R_I)
    V_SR = grid_integration(V_SR_r,dr,CGF_He,CGF_H)
    E_SR = energy_calculation(V_SR,P)
    E_0 = E_hart_r + E_XC + E_SR + T + E_self + E_II

    return E_0
```

From this, we may now take a more accurate approximation of $H(t + a\Delta t/2)$ using the following function:

$$H^c(t + a\Delta t/2) = H[\boldsymbol{P}, t] + \frac{1}{2}\left(H[\boldsymbol{P}^p, t + \Delta t] - H[\boldsymbol{P}, t]\right) \tag{68}$$

5. Determine the corrected $\boldsymbol{P}$ matrix
   As before, we may now use the corrected Hamiltonian at $t+a\Delta t/2$ in the selected propagator to determine $\boldsymbol{C}(t + \Delta t)$ and thus $\boldsymbol{P}(t + \Delta t)$.

For the other propagators used in PyRTP (ETRS, AETRS and CFM4), only step 5 of the process is performed. AETRS, CAETRS and CFM4 are different, in that extrapolation is used to determine $\hat{H}$ at times other than $t + \Delta t/2$. There are two distinct approaches to this problem: the Lagrange interpolation method, as used here, or the following equation:

$$\hat{H}(t + f\Delta t) = (1 + f)\hat{H}(t) - f\hat{H}(t - \Delta t) \tag{69}$$

The decision was made to use Lagrange interpolation to improve accuracy, as Equation 69 depends only on one previous point, however both are valid approaches.

Given we now have the density matrix $\boldsymbol{P}(t + \Delta t)$, we may now put it back into the SCF loop to converge upon a ‘true’ density matrix. From then, we may determine the properties of the system.

5.2.4 Information collection

After the have determined the energy and density matrix of the system at $t + \Delta t$, we may now find any further information on the system. The primary focus of this project, as it is the most commonly used with respect to RT-TDDFT, is determining the absorption spectrum of a given molecule. This information may be obtained by computing the Fourier transform of the dipole moment, such that:

$$S(\omega) = \mathscr{F}[\mu(t)] \tag{70}$$

Where $\mu(t)$ is the total dipole moment and $S(\omega)$ is the dipole strength function. The code used to perform this calculation and post-processing can be found in Appendix A.10. From there, using the Planck relation, the frequency ω may be scaled, giving an absorption spectrum in terms of the wavelength λ. The dipole moment, thus, is given by the following relation:

$$\mu(t) = \mathrm{Tr}[\boldsymbol{D}\boldsymbol{P}(t)] \tag{71}$$

Where $\boldsymbol{D}$ is the transition dipole tensor (as described in Equation 44).

Another form of information that may be useful in some calculations is the change in electronic charge on each of the atoms, and thus, we may determine this via Mulliken population analysis:

$$\mathrm{MP}_{total} = \mathrm{Tr}(\boldsymbol{P}\boldsymbol{S}) \tag{72}$$

With the diagonal elements of the $\boldsymbol{P}\boldsymbol{S}$ matrix representing the electron charge distribution on each atom.

A link may be further provided to quantum statistical mechanics with the determination of the von Neumann entropy, which is determined as the following:

$$S = -\mathrm{Tr}(\boldsymbol{P}\ln(\boldsymbol{P})) \tag{73}$$

Which, if $\boldsymbol{P}$ is in a basis of eigenvectors, is equivalent to the Shannon entropy, which has a range of uses in statistical mechanics. This is not a comprehensive list, however this illustrates that information may be gathered at each iteration of the RTP loop.

In order to determine what we may seek to change in CP2K, we must now look closely at the procedure it uses to perform RT-TDDFT calculations.

5.3 TDDFT Implementation in CP2K

CP2K is a very comprehensive program, and thus discussing all possible variations of TDDFT calculations that may be performed in it is beyond the scope of this report. As such, we shall focus on specifically on the '*RT Propagation*' RT-TDDFT method, which the user may select [9]. The aim of this section is to identify methods similar to those featured in PyRTP, with the view of comparing them. As such, the process of analysing the CP2K method will take the same form as the description of the method implemented in PyRTP.

- **Ground-state DFT calculations**
  CP2K uses the 'Quickstep' DFT process, which focuses heavily on the use of augmented and non-augmented Gaussian plane waves. The purpose of this is to reduce the number of basis functions per atom, giving simpler representations of the wavefunction, alongside creating less dense density matrices, allowing for 'sparce' matrix proceedures, which reduce computational cost. Given that this area is one of the most well researched sections of the CP2K source code, this calculation as a part of this method is very well optimised, however the key information that the user provides here is the number of points on the integration grid. More points on the grid provides greater accuracy, however drastically increases the computational cost of the DFT calculation. As such, it is useful to determine the minimum number of grid points to converge upon an accurate result.

- **Electric field perturbations**
  Perturbations in CP2K's method are generally performed via a 'delta kick', as previously described in Equation 47, however the 'EFIELD' section may be included in the input file, which allows for a Gaussian pulse to be specified. This is useful in comparative benchmarking of methods, as the same process can be replicated in CP2K and PyRTP.

- **Propagators**
  By default, RT-TDDFT calculations in CP2K are propagated via the ETRS propagator, however the Crank-Nicholson and Exponential Midpoint propagators can also be selected. A predictor corrector algorithm is not used in CP2K, and thus, an accuracy comparison, alongside an analysis of computational cost of the different methods, would be beneficial information.

- **Information collection**
  A variety of information may be obtained, including dipole moment and Mulliken population (among many other things), however there is no method to calculate

the von Neumann entropy implemented in CP2K, despite this being relatively simple to determine after the fact.

- **Computational optimisations**
  CP2K makes heavy use of multi-threading via OpenMP on the CPU and CUDA in certain operations on the GPU. Currently, this feature is not present in PyRTP, meaning direct timings of processes are not useful when comparing the implementation of CP2K and PyRTP, unless the single threaded version is used.

Given this, the following areas to compare have been identified:

- Optimising the number of grid points for DFT calculations in CP2K
- Computational cost of all propagator methods in PyRTP (particularly those not featured in CP2K)
- Accuracy between predictor-corrector algorithm in PyRTP and alternate method in CP2K for EM and CN propagators
- Ease of information collection in CP2K against PyRTP

6 Results

6.1 Optimising the number of DFT grid points

In order to optimise the number of points used in the DFT calculation in CP2K, we run a series of DFT calculations on the same molecule with only 1 SCF step, increasing the ‘cutoff’ value (which corresponds to the ‘coarseness’ of the grid) [9]. CP2K uses a multi-level integration grid, and thus, by increasing the cutoff value, we increase the number of points on the grid at each level, with the lower grids being progressively coarser. In general, we seek to have an even distribution of plane-wave Gaussians on each level of the grid, with a converged energy, whilst minimising the cutoff. This, of course, could be done by creating a CP2K input file, running and recording the total energy and multigrid properties, whilst progressively increasing the cutoff on each run, however using a *bash* script is more efficient. These scripts, which construct the files to run the input file with a range of specified cutoffs is adjusted from the scripts on the CP2K website [28] and can be found in Appendices A.11, A.12 and A.13. Helium hydride ($HeH+$) was selected to simulate, as the molecule is very simple, thereby reducing computational cost. The following table of values was produced:

# Grid cutoff vs total energy					
# Date: Tue 2 May 2023 11:59:55 BST					
# PWD: /Users/matt/cp2k/data					
# Cutoff (Ry)	Energy (Ha)	Grid 1	Grid 2	Grid 3	Grid 4
50	-0.208725286	13392	1461	0	0
100	-2.57961377	11560	3221	72	0
150	-2.636059034	10981	2411	1461	0
200	-2.64044996	10087	3281	1413	72
250	-2.641126788	9508	2946	2327	72
300	-2.641132721	8614	2946	3221	72
350	-2.641132871	7890	3525	1977	1461
400	-2.641132076	7890	3236	2266	1461
450	-2.64113207	7601	3380	2411	1461
500	-2.641132001	7577	3380	2411	1485

Table 1: A table showing the recorded total energies and distribution of plane-wave Gaussian functions of a helium hydride molecule ($HeH+$) for a range of cutoff values. Total energies are given in Hartree units and cutoffs are given in Rydberg units.

Using this data, we can determine that a cutoff of 350 Ry is most suitable, given that the energy is converged up to 6 significant figures, and the coarsest grid, Grid 4, is well used, giving a good distribution of plane-wave Gaussian functions across all grid levels. Of course, further optimisation may be achieved by reducing the range of cutoff values, however the returns from this are diminishing.

6.2 Comparing the computational costs of various propagators

In order to compare the efficiency of the propagators implemented in PyRTP, the average time to propagate the density matrix was recorded for each propagator. The computational time should scale with speed of the processor used, so for continuity, the system used to run the code had the following specifications:

CPU: Intel Core i5-7500 @ 3.40GHz
Memory: DDR4 Dual Channel @ 2666 MHz
GPU: Nvidia Geforce GTX 1060 6GB
OS: Ubuntu 22.04.2 LTS
Python version: 3.11.3
NumPy version: 1.24.2
SciPy version: 1.10.1
SymPy version: 1.11.1
PySCF version: 2.2.1

The PyRTP code was set to do only one SCF loop (given that we are not interested in how long this process takes and wish to minimise the time taken to run the code for each propagator) and set to run for 100 timesteps, in order to determine if there is any correlation between number of previous timesteps and the propagation time. The following plot was produced:

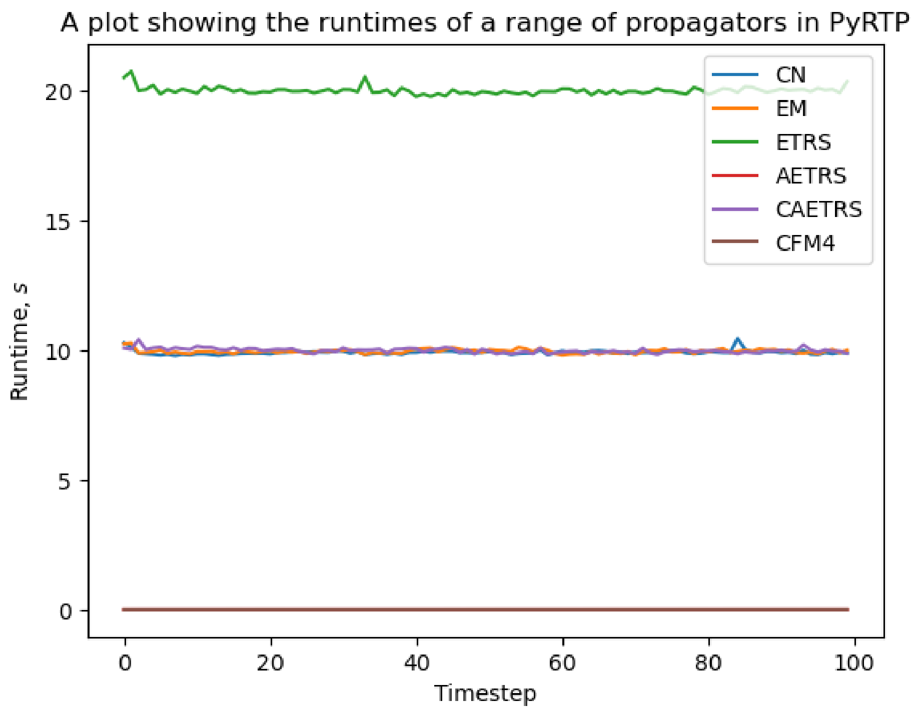


Figure 1: A plot showing the runtimes of a range of propagators for 100 timesteps.

A full list of plots for each propagator can be found in Appendix A.14. From this plot, it can be shown that the runtimes of all of the propagators are approximately constant, and thus, the following average runtimes were found to be:

Propagator	Average Runtime, s
Crank-Nicholson	9.92583
Exponential Midpoint	9.96984
ETRS	20.02365
AETRS	0.00167
CAETRS	10.00078
CFM4	0.00241

Table 2: A table showing the average runtimes of a set of propagators in PyRTP

From this table, it can be shown that the AETRS propagator was the most computationally efficient, however in order to determine the effectiveness of each of the propagators, we must evaluate the accuracy of each of them.

6.3 Comparing the accuracy of various propagators

In order to compare the accuracy of the propagators used in PyRTP and CP2K, it was determined that a comparison of absorption spectra from each propagator is a suitable method. As a point of reference, LR-TDDFT was used to determine the exact absorption spectrum of $HeH+$, with the following spectrum:

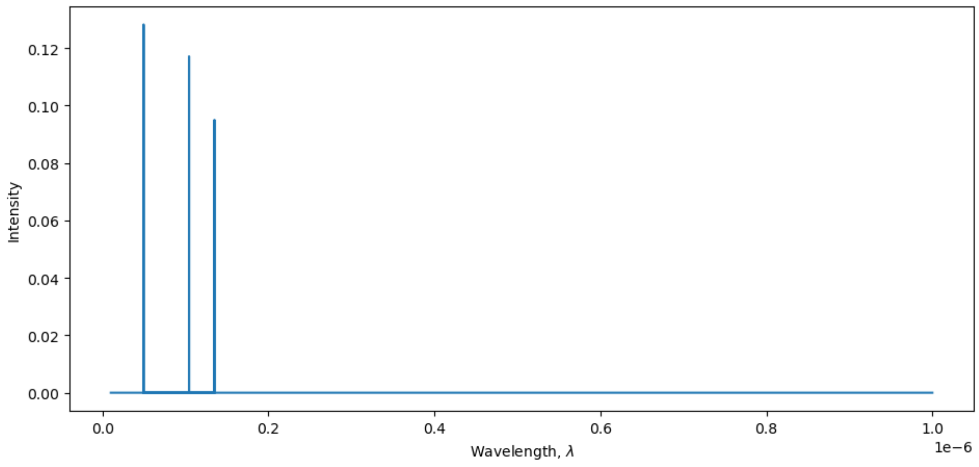


Figure 2: A plot showing the absorption spectrum of helium hydride ($HeH+$), computed using LR-TDDFT.

The LR-TDDFT code is given in Appendix A.15. Using 2000 timesteps of 0.1 au_t and an applied Gaussian electric pulse of 0.001 D , the following absorption spectra were produced for each of the propagators:

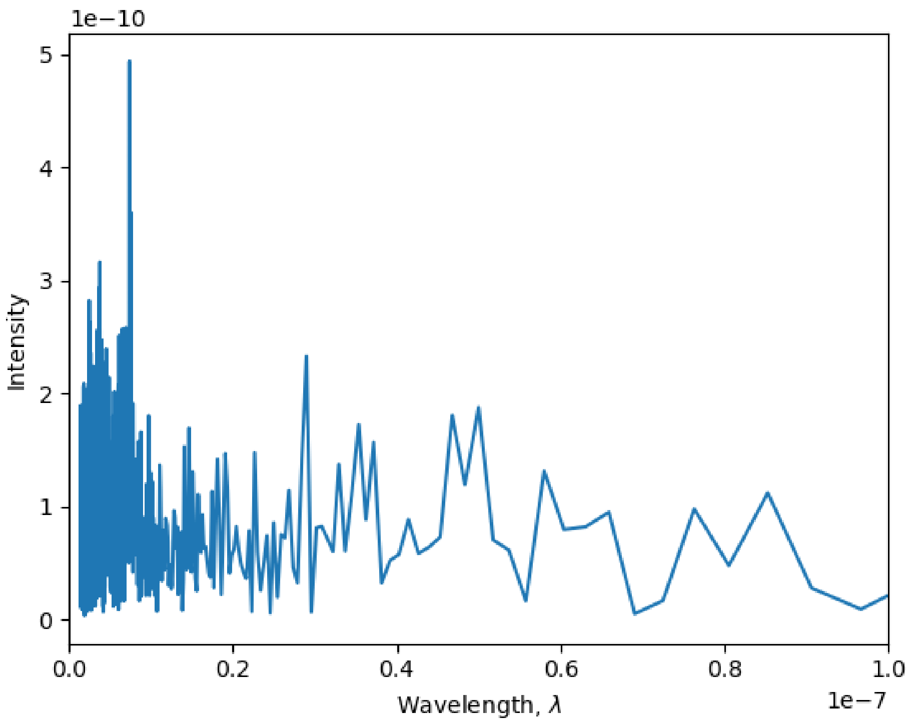


Figure 3: A plot showing the absorption spectrum of helium hydride ($HeH+$), computed using the Crank-Nicholson propagator in PyRTP.

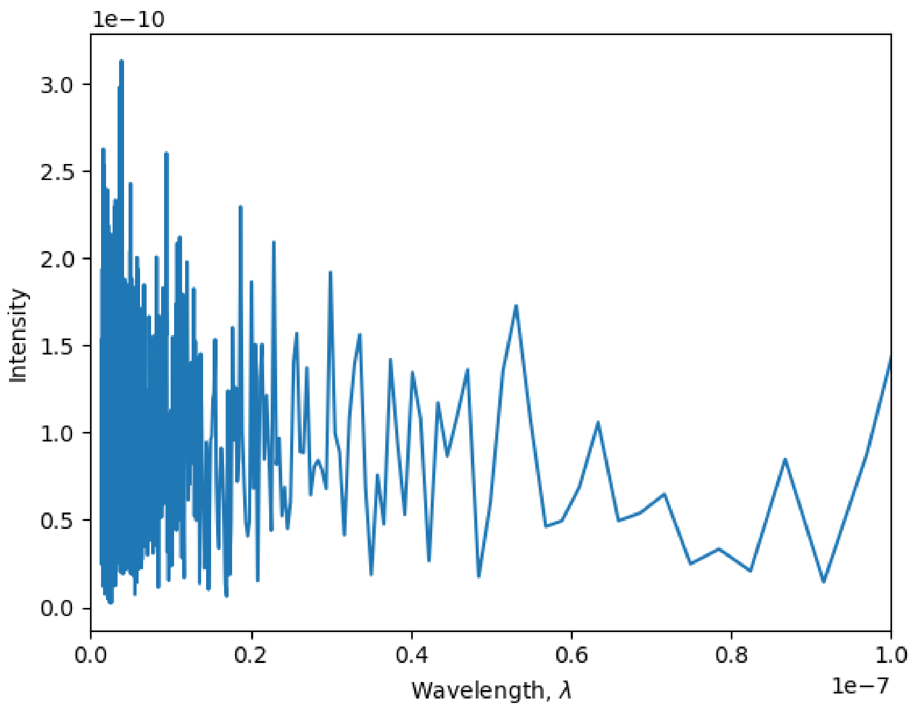


Figure 4: A plot showing the absorption spectrum of helium hydride ($HeH+$), computed using the exponential midpoint propagator in PyRTP.

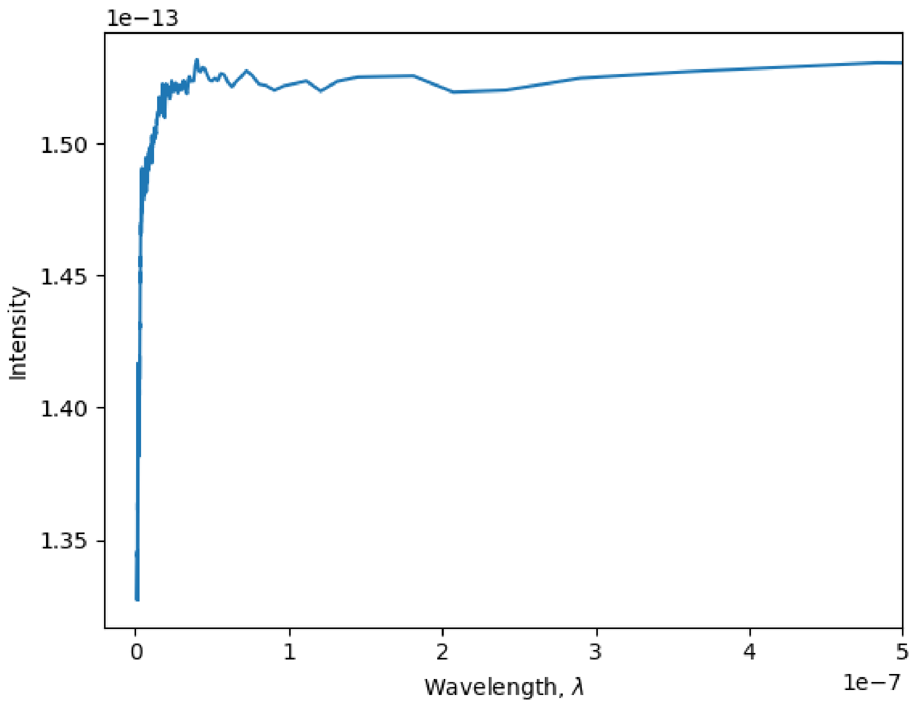


Figure 5: A plot showing the absorption spectrum of helium hydride ($HeH+$), computed using the ETRS propagator in PyRTP.

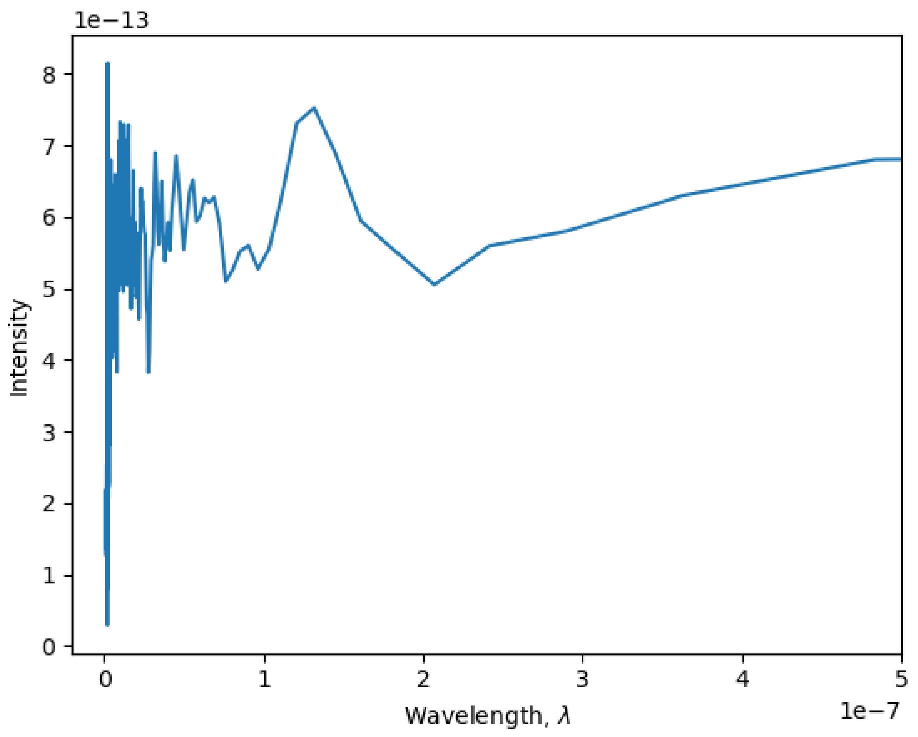


Figure 6: A plot showing the absorption spectrum of helium hydride ($HeH+$), computed using the AETRS propagator in PyRTP.

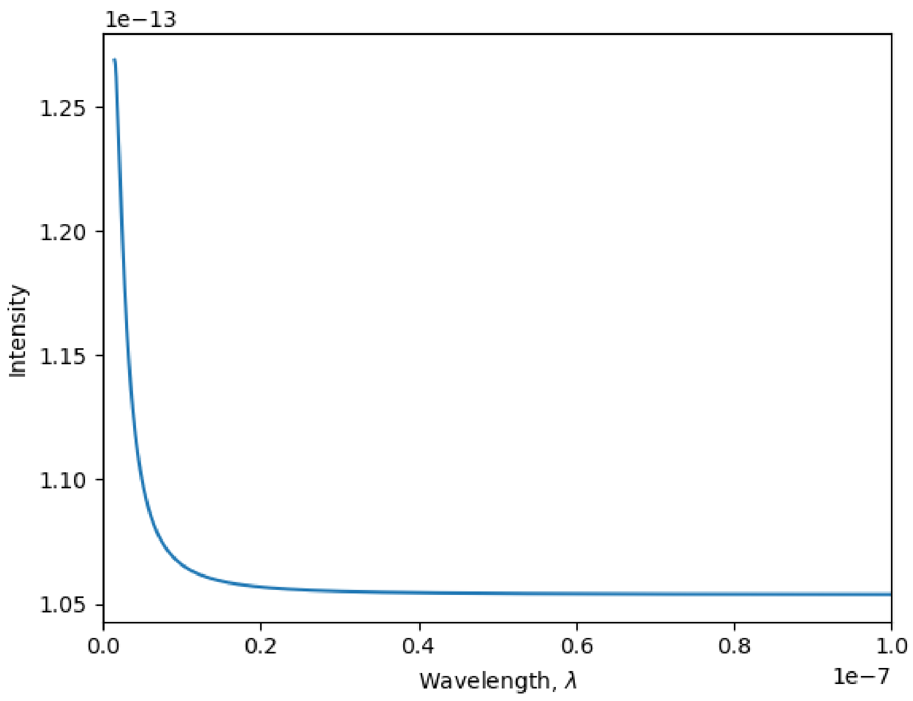


Figure 7: A plot showing the absorption spectrum of helium hydride ($HeH+$), computed using the CAETRS propagator in PyRTP.

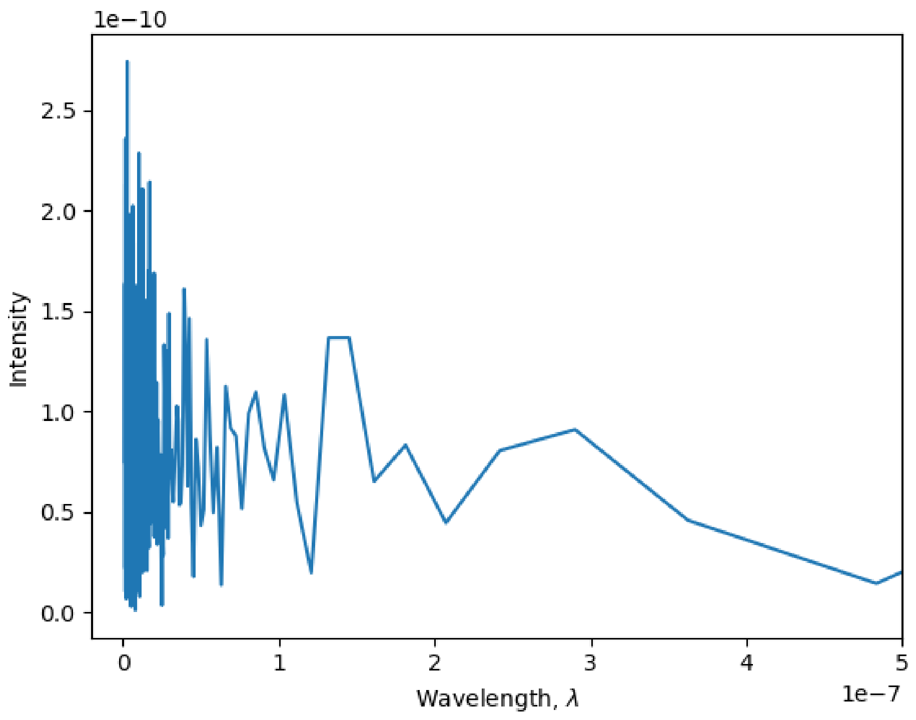


Figure 8: A plot showing the absorption spectrum of helium hydride ($HeH+$), computed using the CFM4 propagator in PyRTP.

In order to compare the implementation of propagators in PyRTP and CP2K, the following absorption spectra were computed using CP2K:

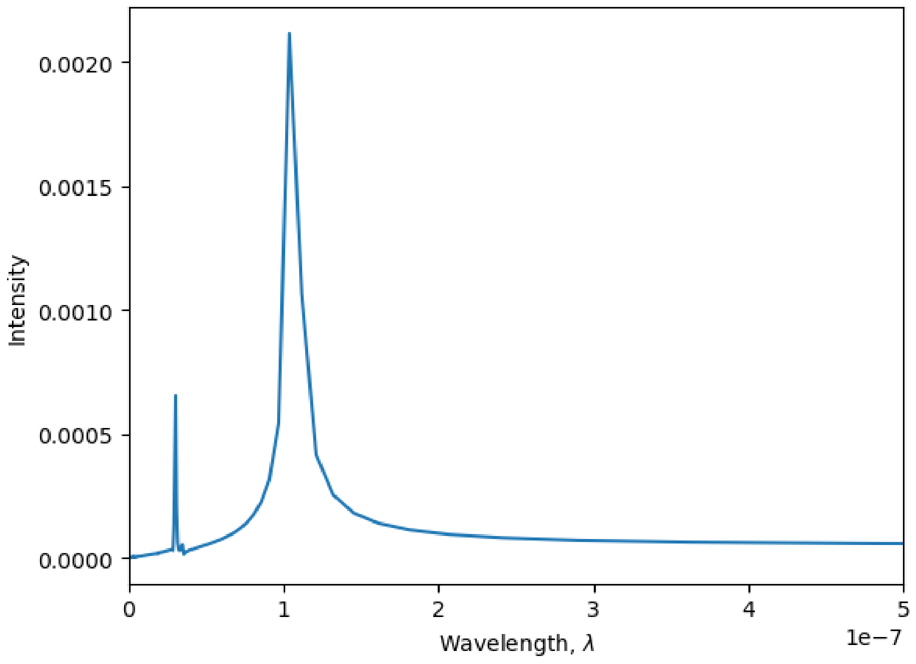


Figure 9: A plot showing the absorption spectrum of helium hydride ($HeH+$), computed using the Crank-Nicholson propagator in CP2K.

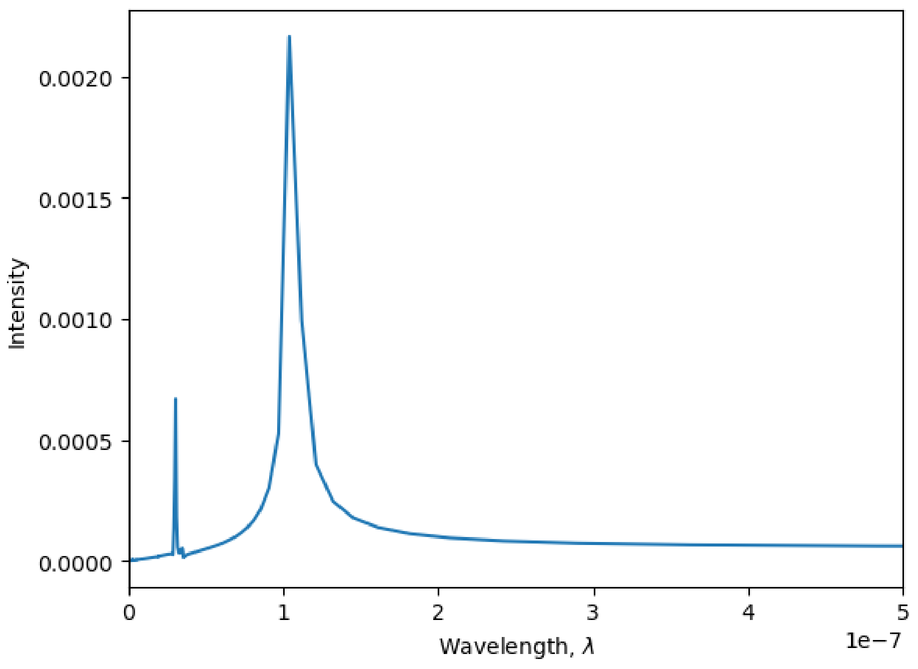


Figure 10: A plot showing the absorption spectrum of helium hydride ($HeH+$), computed using the exponential midpoint propagator in CP2K.

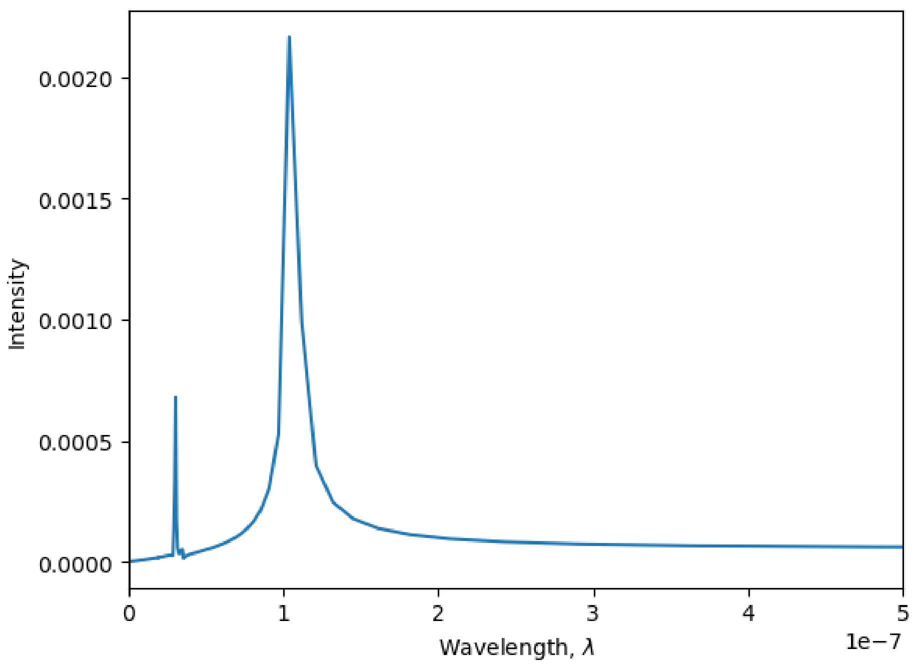


Figure 11: A plot showing the absorption spectrum of helium hydride ($HeH+$), computed using the ETRS propagator in CP2K.

7 Discussion

7.1 Propagator Accuracy

From visual inspection alone, it is clear to see that the ETRS and CAETRS propagators produced wildly inaccurate absorption spectra. It is suspected that there is some underlying error in these propagators, especially given that they both use the same equation to determine the unitary matrix. As such, it was not possible to fully assess the accuracy of these propagators for the purposes of this report.

All of the propagators tested (besides CAETRS) correctly identified points very close to the true absorption spectrum peaks, meaning the propagation methods used were all reasonable methods. Determining which propagator is '*best*' also requires a discussion on the amount of noise generated by the propagator when producing the absorption spectra, which is discussed in Section 7.4.

During testing, a variety of Gaussian pulse strengths were used, and for certain propagators, the system would return to the ground state within a few timesteps if the pulse was not strong enough. This was seen most often with the 'ETRS' propagators, which produced the following energy plots:

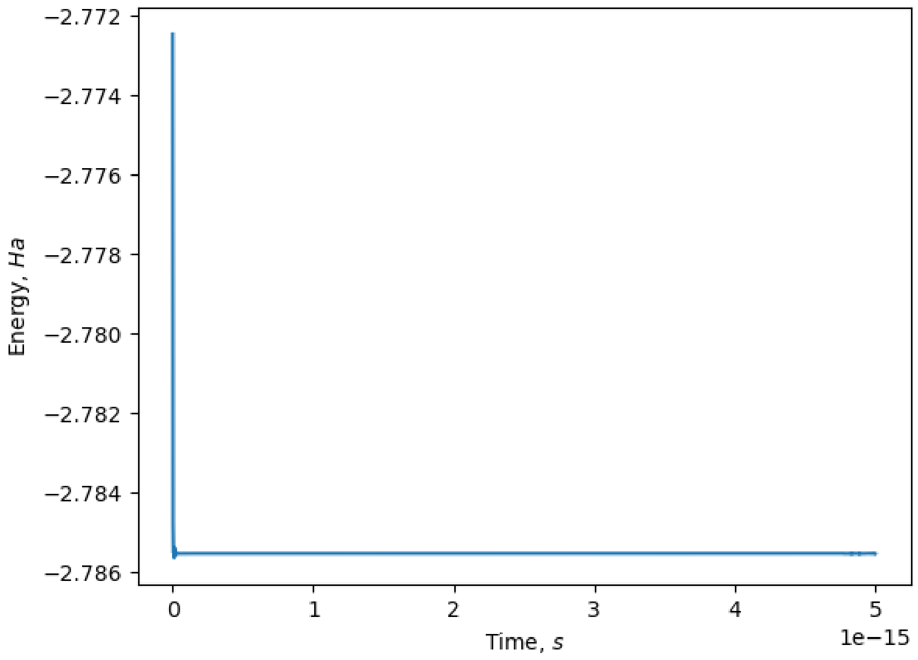


Figure 12: A plot showing the calculated energy of a helium hydride ($HeH+$) molecule against time, calculated using the ETRS propagator in PyRTP with an insufficiently large electric pulse ($2 \times 10^{-5}\, D$).

The energy stays stable after a small number of timesteps, and thus it is useful to look at the initial timesteps to determine the issue:

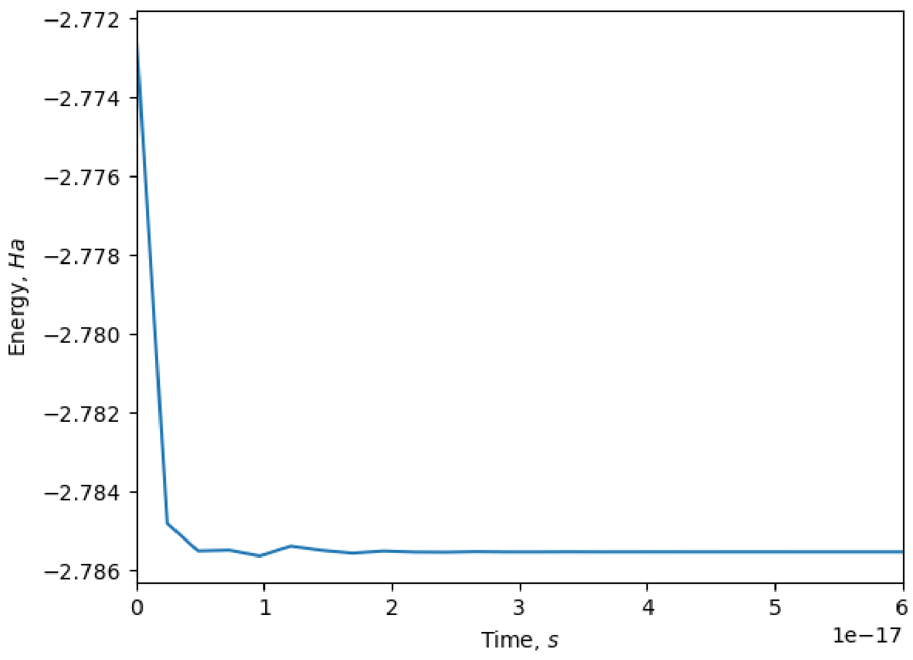


Figure 13: A plot showing the calculated energy of a helium hydride ($HeH+$) molecule against time, calculated using the ETRS propagator in PyRTP with an insufficiently large electric pulse ($2 \times 10^{-5}\ D$). The propagation time is restricted to a short interval to allow for the fast return to a constant energy to be easily seen.

For the first few timesteps, there is some small oscillation in energy, but it quickly becomes constant. It is assumed that the cause of this is the electric field pulse being insufficient in energy to put the electrons into an excited state, and thus adds to the kinetic energy of the system, which is quickly dissipated. To correct this, a larger Gaussian pulse of 0.001 D was used going forward to ensure the system was provided with sufficient energy.

7.2 Propagator Stability

The propagators were tested for stability by setting the strength of the applied Gaussian pulse to either zero or sufficiently small enough that the system returned to the ground state quickly. From this, it was found that all of the propagators had sufficient stability to complete a usable absorption spectrum, with AETRS and CAETRS having the most significant instability towards the end of the simulation:

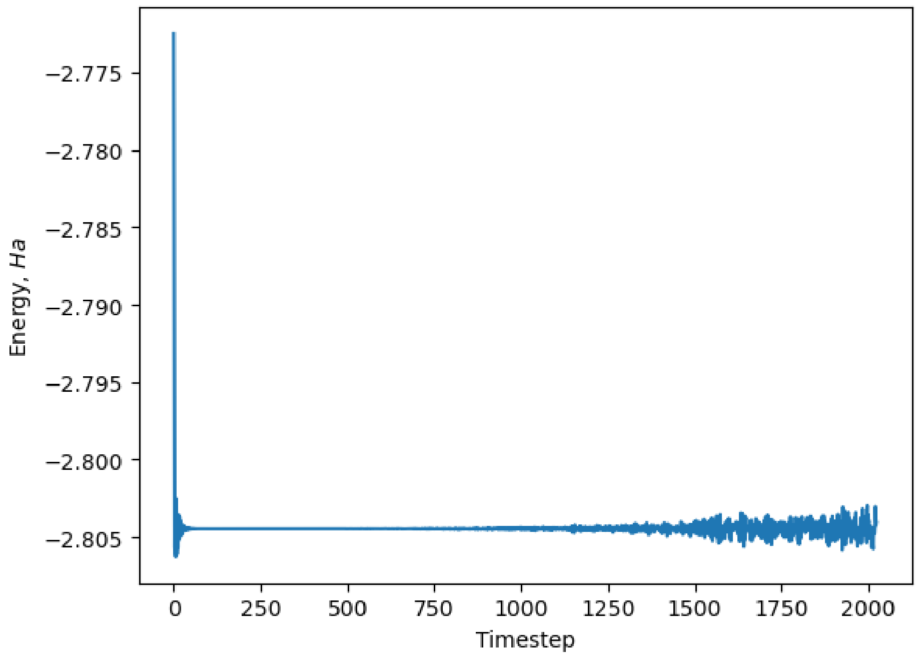


Figure 14: A plot showing the instability of the energy calculated by the AETRS propagator in PyRTP as the number of timesteps increases.

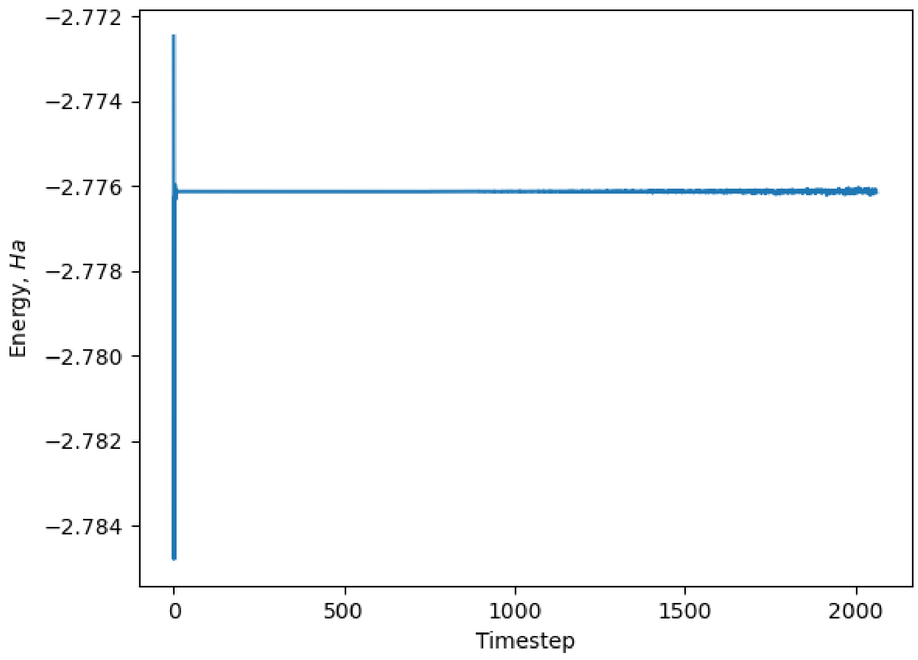


Figure 15: A plot showing the instability of the energy calculated by the CAETRS propagator in PyRTP as the number of timesteps increases.

7.3 Ease of Information Collection

Currently, any information determined or produced by CP2K is given in the '.out' file once the process has completed. This file is a plain text file, and thus, adds an extra step before post-processing, which may confuse less experienced users. By contrast, all of the calculated and useable data is given as arrays in PyRTP, which allows the user to begin post-processing immediately.

One method to mitigate this is to create a Python user interface, which works as an 'in-between' of CP2K and the user, while not requiring the user to write input files in the *Fortran* programming language. This is in development, under the title of 'PyCP2K' [29]. This package works well as an interface, however it could benefit from some built-in functions to perform various post-processing methods, which will be discussed in the outlook section. The ability of PyCP2K to build the input files, whilst not necessary for the majority of users, dramatically reduces the 'learning curve' for new users of the program, and arguably is a more effective teaching tool in what each input is doing than a well described tutorial in using the input files.

7.4 Signal Processing of $\mu(t)$

In order to appropriately and accurately determine the absorption spectrum of a simulated molecule, whilst minimising the simulation time, it is important to consider the timestep and total number of iterations. In general, we require the timestep to be sufficiently small in order to capture the required maximum frequency. This condition is described by the Nyquist-Shannon sampling theorem, which states that [30]:

Theorem 6 *If a function $f(t)$ contains no frequencies higher that W Hz, it is completely determine by giving its ordinates at a series of points spaced $1/(2W)$ seconds apart.*

Given that we are interested in the wavelength λ, we may determine a function relating the minimum timestep of the RT-TDDFT calculation and the required minimum wavelength. The wavelength is connected to the frequency and energy via the Planck relation, such that:

$$E = hf = \frac{hc}{\lambda} \tag{74}$$

We may then find that $f = c/\lambda$, which we may substitute into the Nyquist-Shannon theorem, giving:

$$\Delta t = \frac{\lambda}{2c} \tag{75}$$

Where Δt is in seconds. In general, most quantum chemistry programs give the timestep in Hartree units ($1\ au_t = 2.418 \times 10^{-17}$), thus, we find that the minimum timestep to capture the smallest wavelengths within the absorption spectrum of a simulated molecule is given by the following function:

$$\Delta t = \frac{\lambda}{(4.836 \times 10^{-17})c} \tag{76}$$

In the majority of cases, the user will be mostly interested in the visible light spectrum, with a minimum wavelength of $3.8 \times 10^{-7}\, m$, thus, using Equation 76, we find the minimum timestep for visible light to be approximately $26.18\, au_t$. We now must consider the largest wavelengths, as we require at least one full wavelength to be captured to accurately describe this frequency. As such, we use the following relation between timestep and the peak frequency:

$$\Delta t = \frac{1}{f_{max}} \tag{77}$$

As previously stated, we may use the relation between f and λ to write Equation 77 in terms of λ:

$$\Delta t = \frac{\lambda_{min}}{c} \tag{78}$$

This allows for determination of the highest frequency, and thus, smallest wavelength. The frequencies between zero and the highest frequency, in the majority of Fast Fourier transform operations, occurs via creation of a linearly spaced array of frequencies, such that the length of the array is equal to the number of timesteps. In order to ensure that the lowest frequencies, and thus highest wavelengths, are captured, the following relation may be used:

$$f_{min} = \frac{f_{max}}{N} = \frac{1}{N \cdot \Delta t} \tag{79}$$

Converting from f_{min} into λ_{max}, we find that:

$$\lambda_{max} = \frac{c}{f_{min}} = cN\Delta t \tag{80}$$

Thus, given out maximum timestep to capture the smallest visible wavelength, in order to capture the largest wavelength of visible light (7.5×10^{-7} m), we find that the minimum number of timesteps to capture the maximum visible light frequency is approximately 3.947 steps. Of course, it is unlikely that an accuracy intensity will be captured after this timescale, however we may use a short number of timesteps (on the order of hundreds) and use padding to produce a usable plot. The following function was used to produce a padded dipole moment plot:

```
mu_padded=np.append(np.array(mu),np.ones(2000-nsteps)*(np.mean(mu)))
t_padded=np.arange(0,2000*dt,dt)*2.418e-17
plt.plot(t_padded,mu_padded)
plt.xlabel('Time, $s$')
plt.ylabel('Dipole moment, $\mu$')
```

This produced the following padded plot:

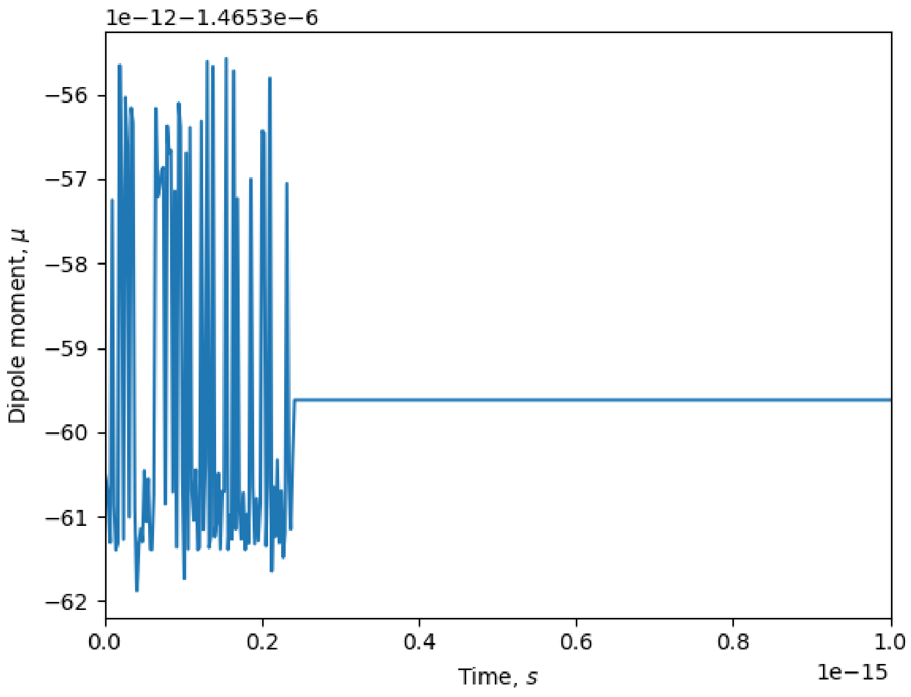


Figure 16: A plot of the calculated dipole moment against time, computed using the CFM4 propagator in the PyRTP program. The program was run for 100 timesteps and padded for 1900 timesteps after to produce a useable plot.

This data was then processed using a Fourier transform to produce an absorption spectrum:

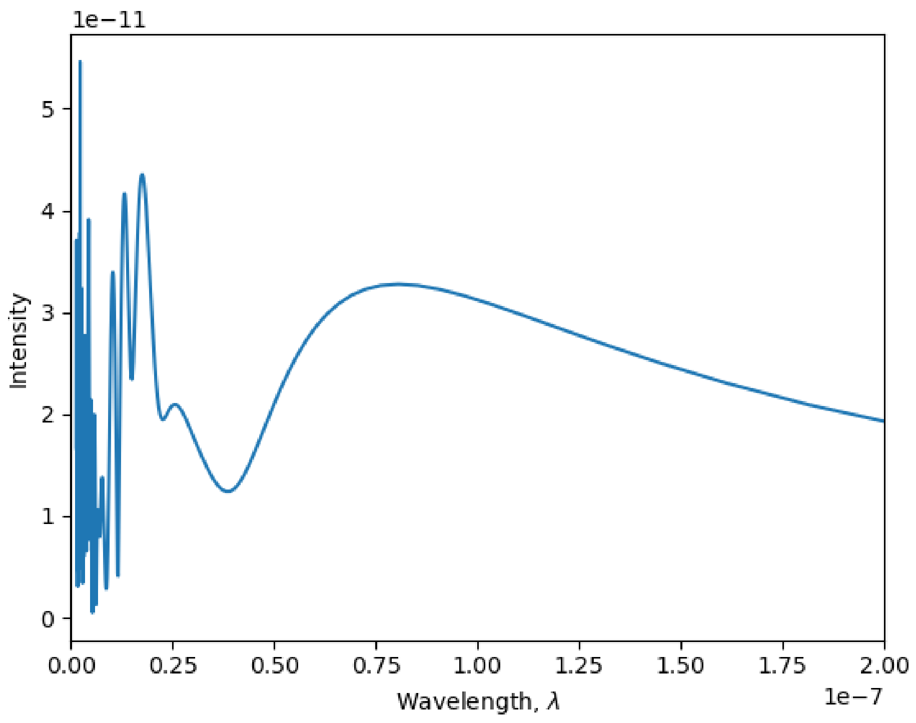


Figure 17: A plot showing the absorption spectrum of helium hydride ($HeH+$), computed using the CFM4 propagator in PyRTP. The program was run for 100 timesteps and padded for 1900 timesteps after to produce a useable plot.

This provides a significant reduction in computational cost, whilst preserving a good degree of accuracy. It is also worth considering the use of filtering to remove noise, which is performed using the following code:

```
filterpercentage=20
sp=scipy.fft.rfft(mu)
freq = scipy.fft.rfftfreq(mu.size,(dt*2.4188843265857e-17))
ld=c/freq
indexes=np.where(sp<np.percentile(sp,filterpercentage))[0]
print(indexes)
sp[indexes]=0
plt.plot(ld,np.abs(sp))
plt.xlabel('Wavelength, $\lambda$')
plt.ylabel('Intensity')
```

Using this, the absorption spectra can be filtered to give plots in the following form:

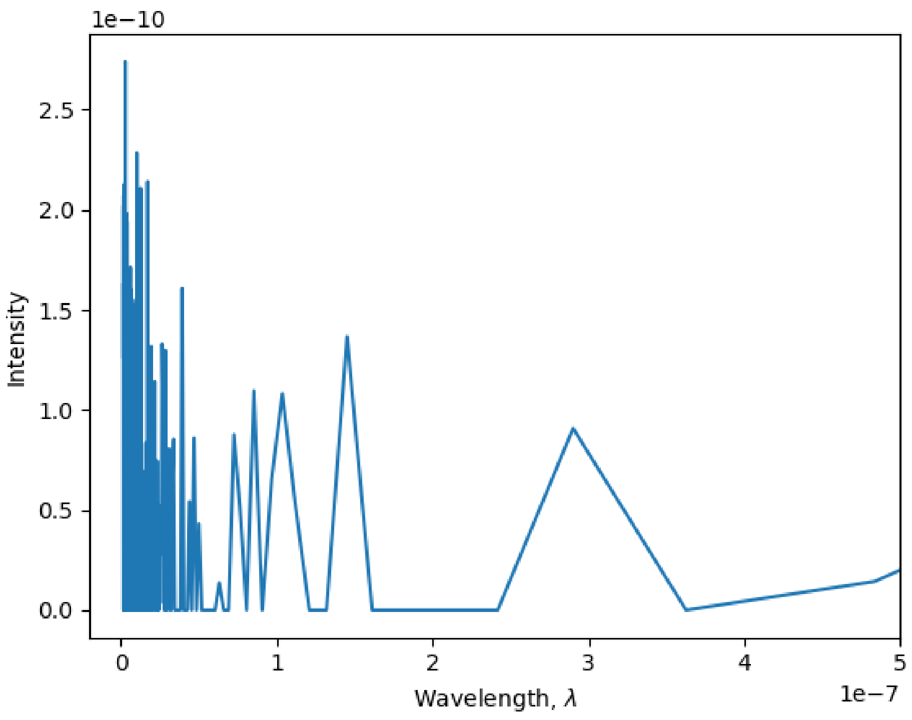


Figure 18: A plot of the calculated absorption spectrum of $HeH+$, computed using the CFM4 propagator in the PyRTP program. The spectrum plot was filtered such that all frequencies with intensities in the lower 50th percentile had intensities set to zero.

A further point to consider is whether the Gaussian pulse applied is suitable. In order to do this, we must look closely at the waveform produced by the dipole moment:

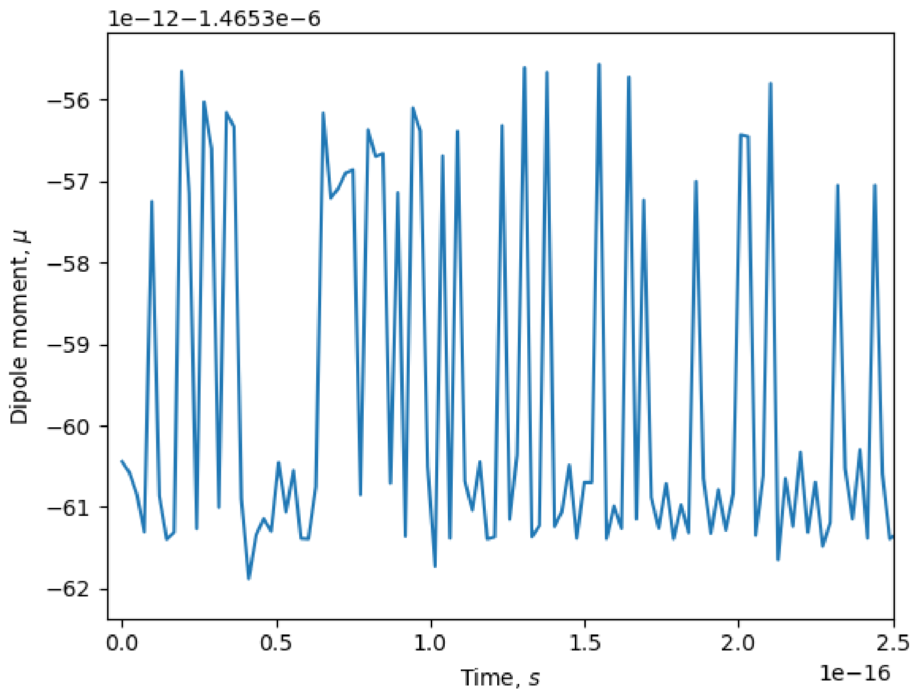


Figure 19: A plot of the calculated dipole moment against time, computed using the CFM4 propagator in the PyRTP program, such that the applied Gaussian electric pulse was of suitable strength.

It is shown that the dipole moment μ forms an reasonably regular waveform, which is not completely destroyed. It is clear that there is some additional 'noise', however given that one can easily identify a waveform in this plot, we may assume that the field strength was suitable. If the field strength were too strong, we would expect the noise to be stronger than the waveform, with no obvious wave to be seen. Conversely, if the field were too weak, we would expect to see little to no change in the dipole moment, resulting in a flat plot.

7.5 Shannon Entropy and the Notion of Equilibrium

Given the relative uncertainty some users of the RTP functions in CP2K may have in regard to setting the appropriate number of timesteps, it is important to consider what can be done to reduce any uncertainty a user may have. One possible method is to make use of the Shannon entropy (sometimes referred to as the information entropy) to end the loop when the simulated system has reached statistical equilibrium. As shown in Section 5.2.4, it is trivially easy to determine the Shannon entropy from the density matrix, and thus, the following code was used to calculate the Shannon entropy:

```
def ShannonEntropy(P):
    P_eigvals=np.linalg.eigvals(P)
    P_eigvals_corrected=[x for x in P_eigvals if x>0.00001]
    P_eigvecbasis=np.diag(P_eigvals_corrected)
    SE=np.trace(np.dot(P_eigvecbasis,np.log(P_eigvecbasis)))

    return SE
```

Given that a matrix in a basis of its own eigenvectors is equal to a diagonal matrix of its eigenvalues, we use the 'np.linalg.eigvals' function (rather than the 'np.linalg.eig' function, which is slower computationally) and use 'np.diag' to construct this. For the purposes of evaluating the Shannon entropy, the program may run into issues when the eigenvalues (λ_i) become very small, as $\ln(x)$ sharply tends towards negative infinity as x tends to zero. As such, we may the following approximation:

$$\forall(\lambda_i \ll 1) \in \boldsymbol{P}_{eigvecbasis}, \ -(\lambda_i \ln(\lambda_i)) \approx 0 \tag{81}$$

Using the method described in PyRTP, the following plot of Shannon entropy against time was produced:

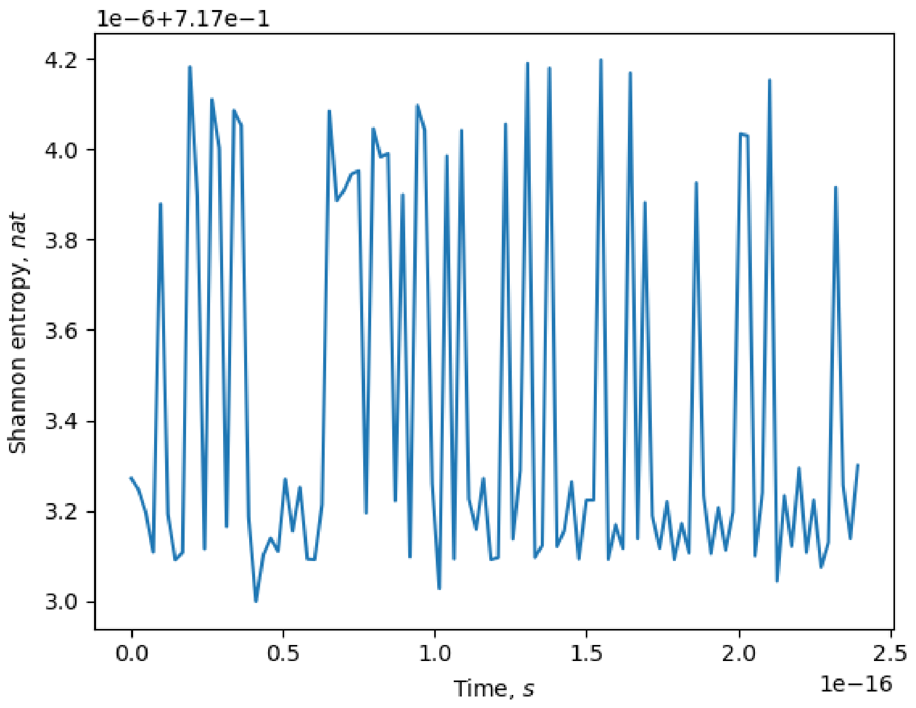


Figure 20: A plot of the calculated Shannon entropy against time, computed using the CFM4 propagator in the PyRTP program.

It is shown in this plot that the Shannon entropy oscillates at a relatively constant frequency. This behaviour is expected, given that DFT is an 'NVE' simulation, meaning no energy is radiated away, and thus the energy applied to the system via the electric pulse is kept within the system. As shown in the Sabirov and Shepelevich paper [31], the Shannon entropy is a measure of 'the degree of delocalization of electron density', and thus, we may use this to determine which states over the course of propagation are highly delocalised, and thus, excited. For a system where energy may be lost, i.e. a canonical ensemble (NVT), this method of using the Shannon entropy to equilibrate the system is viable.

It is also worth considering additional 'entropy-like' measures, such as the von Neumann entropy and the entropy from the density matrix in a basis of the initial density matrix. These measures were calculated using the following functions:

```
def vonNeumannEntropy(P):
    vNE=np.trace(np.dot(P,np.log(P)))
    return vNE

def PGSEntropy(P,Pinit):
    PPinitBasis=np.matmul(np.linalg.inv(Pinit),np.matmul(P,Pinit))
    EPinitBasis=np.trace(np.dot(PPinitBasis,np.log(PPinitBasis)))
    return EPinitBasis
```

Using these functions, the following plots were produced:

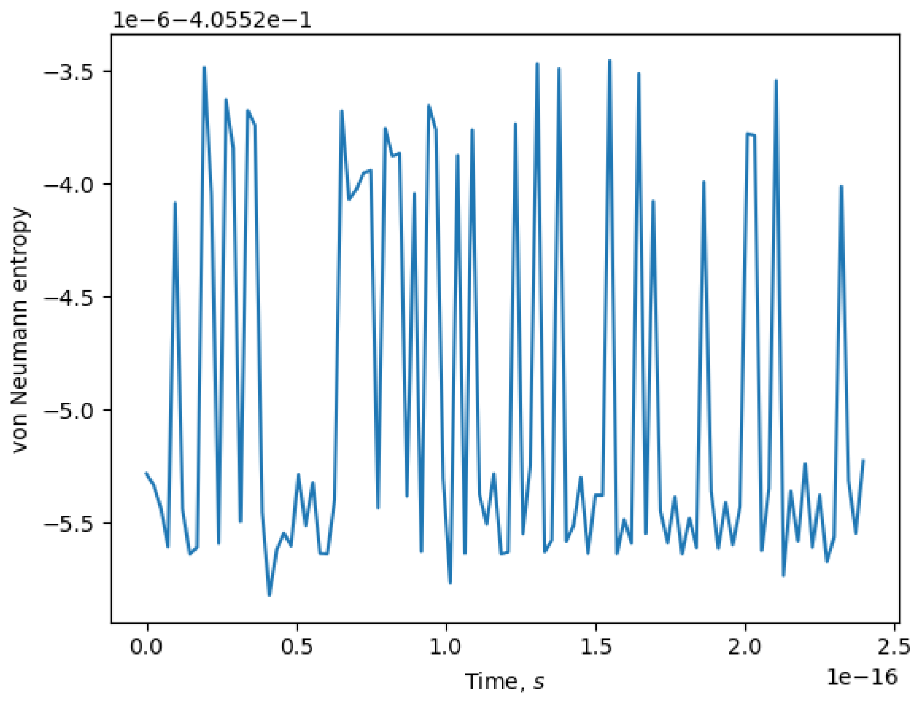


Figure 21: A plot of the calculated von Neumann entropy against time, computed using the CFM4 propagator in the PyRTP program.

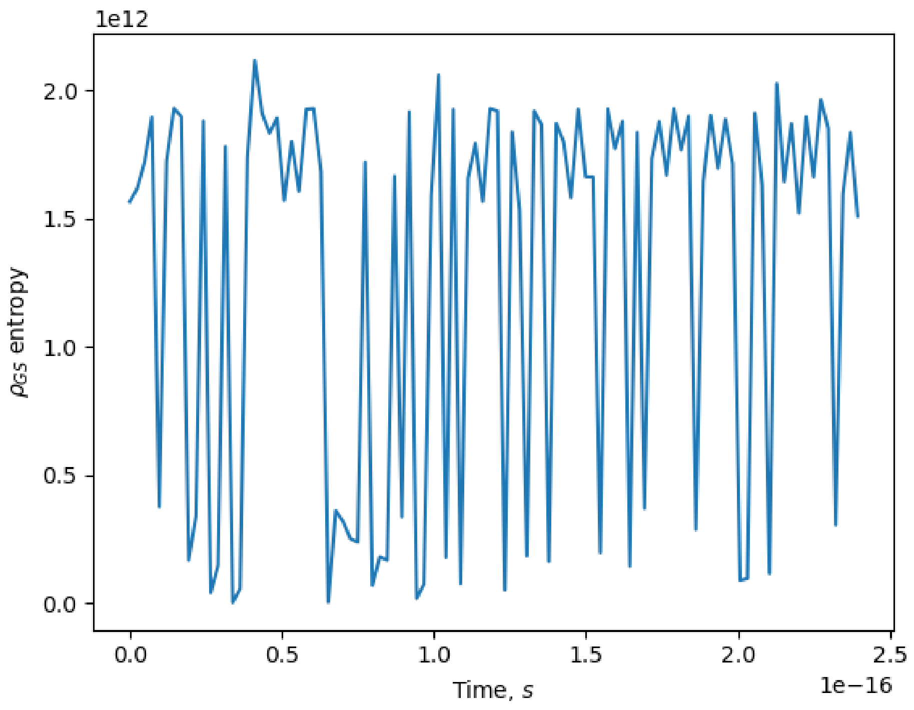


Figure 22: A plot of the calculated ground state density matrix entropy against time, computed using the CFM4 propagator in the PyRTP program.

7.6 Determining the most suitable propagator

Based on the information collected on computational cost and accuracy, it was determined that the CFM4 propagator was the most suitable propagator in the majority of cases, given its minimal computational cost and suitable accuracy. The accuracy was deemed to be the best of all of the propagators tested, as the filtering percentage to produce a useable plot was minimal. The next most suitable propagator was determined to be the exponential midpoint propagator, given the slight increase in accuracy over the Crank-Nicholson propagator. The AETRS propagator was not suitable to determine energies, however it did prove to be useful in determining the absorption spectrum of $HeH+$, so long as the applied electric pulse is sufficiently strong. Further improvements to this propagation method could produce a particularly efficient regime which preserves time-reversal symmetry, should this feature be necessary for a user. It is worth considering, however, that the AETRS propagator has significant instability beyond 1000 timesteps, and thus this propagator is only suitable for short timescale simulations, as shown in Figure 14.

7.7 Comparing propagators in CP2K and PyRTP

It is clear to see from Figures 9 and 10 that the accuracy of the propagators in CP2K is greatly superior to the accuracy in PyRTP, given the almost non-existent noise in the plots produced by CP2K. This would imply that the predictor-corrector algorithm is less accurate than not using it, however further inspection shows that CP2K was only able to identify 2 of the 3 main absorption wavelengths. As such, there is a possibility that despite producing significantly more noise, using the predictor-corrector algorithm may produce results with superior accuracy.

The lack of noise in the CP2K plots is presumed to be as a result of the DFT calculation using multiple orbitals, whereas PyRTP uses only the $1s$ orbital. Using multiple orbitals provides greater accuracy when determining the density matrix, which as a result, provides a more accurate dipole moment value. As such, it is not currently possible to evaluate the accuracy of propagators in CP2K against the accuracy of propagators in PyRTP.

8 Conclusion

The objectives of this project were to reproduce the RT-TDDFT method implemented in CP2K in another programming language, both theoretically and computationally, and to compare the accuracy and efficiency of possible optimisations between CP2K and the computational method determined. This was done by first considering the theoretical RT-TDDFT process and how it can be 'streamlined' wherever possible. Next, the PyRTP program was produced in Python through an analysis of common TDDFT methods, in order to compute the absorption spectra of helium hydride ($HeH+$), which could then be compared to similar absorption spectra computed using CP2K [32]. The computational cost was then considered for each of the propagators within the PyRTP and CP2K programs, in order to minimise the computational cost whilst preserving sufficient accuracy. The ease of information collection and the use of possible information was then considered, most notably the determination and use of the Shannon entropy in TDDFT, in order to determine if there are any changes which can be made to improve the implementation of RT-TDDFT in CP2K. Finally, an analysis and discussion of the signal processing of the dipole moment to produce the absorption spectrum was performed, in order to determine the criteria to produce an accurate spectrum, as well as reduce computational cost as much as possible.

From this method, it was found that the 'Commutator-Free 4th Order Magnus' (CFM4) propagator was the most suitable for use in RT-TDDFT calculations, of the propagators tested. It was also determined that the Shannon entropy was suitable to determine how delocalised the electrons are at a given time within a simulation, but was not a suitable measure of how far a system is from equilibrium. The von Neumann entropy was found to be useful in determining whether a state is pure, which should generally be constant in all simulations (and thereby is useful in determining if a simulation is physically accurate). The entropy determined from the density matrix in a basis of the ground state density matrix was found to be useful in determining how close a given state is from being equal to the ground state, however it was determined that all of the proposed entropy measures would be not be particularly useful in TDDFT, and would be better suited to a 'NVT' simulation method. It was found that using Nyquist-Shannon sampling theorem, in order to accurately capture the smallest wavelength required, the timestep should be set to $\frac{\lambda}{2c}\,au_t$. Given this timestep, it was then found that in order to capture the largest required wavelength, the simulation should run for $\frac{\lambda}{c\Delta t}$ steps. Finally, using an iterative method, the grid cutoff in CP2K for DFT calculations in $HeH+$ was found to be optimal at $350\,Ry$. It was not possible to evaluate whether the predictor-corrector algorithm in PyRTP was more or less accurate than the method used by CP2K, given the different molecular orbitals used. Given these results, a recommendation can be made to improve the ‘real-time propagation’ calculation in CP2K via the inclusion of the CFM4 propagator and using the provided cutoff optimisation code to converge upon the most suitable cutoff for a given molecule.

The most significant issue with this report and project is the failure to consider further propagators, such as higher order Magnus propagators. Further expansions of the Magnus operator would, in theory, provide greater accuracy, with some expected increase in computational cost. It would be interesting to consider which expansion of the Magnus operator provides the most suitable balance of computational cost and accuracy. Another issue was the inability to give a consensus on the suitability of the ETRS and CAETRS propagators, due to inaccurate data. This could be corrected by repeating this study using another program which features all of these propagators, such as ‘Octopus’.

9 Outlook

The process of completing this project has revealed a number of avenues in which this work could be extended, most notably in the development of PyRTP. Given that this program is written in Python and is relatively easy to understand (even by users with limited programming experience), it may be beneficial to continue development into creating a useful teaching tool for undergraduate students learning about DFT and TDDFT. Of course, for this to happen, significant improvements to both computational cost and the user interface would need to occur, however if these changes were to happen, it may prove to be beneficial for students. This could be distributed as a Python package, which would make working with the program simple.

A change that would benefit users significantly is the replacement of the XC energy calculation method, as this currently uses the ‘PySCF’ package, which requires a Unix-based operating system to run. PySCF uses the ‘LibXC’ library, and there are standalone packages which interface this library with Python (PyLibXC), and thus the change would simply require a function to evaluate the XC energy at each point of the integration grid to be written. This change would negate the need for PySCF, and allow PyRTP to be run natively on Windows installations of Python.

Another possible avenue to follow is to develop PyRTP into a program for validating TDDFT methods and calculations. Whilst Python is a relatively computationally expensive programming language, the relatively intuitive syntax allows for efficient testing of methods. This would allow for new methods of obtaining information and new propagators (among other things) to be quickly compared, and provide a resource to compare new methods in other TDDFT programs to. For this purpose, it seems beneficial to make the GitHub page public and maintain it as an open-source program.

Given the results found in this project, proposals to change the CP2K RT-TDDFT source code will be made based on the findings of this report, allowing for comparisons between the implementations in PyRTP and CP2K. This was proposed as one of the initial objectives, however due to the significant time constraints of this project, this could not be completed in time for the submission of this report.

10 References

[1] J. D. Bernal, “The bakerian lecture, 1962. the structure of liquids,” *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, vol. 280, no. 1382, pp. 299–322, Jul. 1964. [Online]. Available: https://doi.org/10.1098/rspa.1964.0147

[2] E. Fermi, P. Pasta, S. Ulam, and M. Tsingou, “Studies of the Nonlinear Problems,” Los Alamos Scientific Laboratory, Tech. Rep., May 1955. [Online]. Available: https://doi.org/10.2172/4376203

[3] J. B. Jackson and N. Metropolis, *The Maniac documentation*, Los Alamos Scientific Laboratory, Los Alamos, NM 87545, USA, December 1951.

[4] M. Levitt and A. Warshel, “Computer simulation of protein folding,” *Nature*, vol. 253, no. 5494, pp. 694–698, Feb. 1975. [Online]. Available: https://doi.org/10.1038/253694a0

[5] *Gaussian 70*, Quantum Chemistry Program Exchange, 1970.

[6] F. Schiffmann, “An atomistic picture of the active interface in dye sensitized solar cells,” Ph.D. dissertation, University of Zurich, Faculty of Science, University of Zurich, Rämistrasse 71, CH-8006 Zürich, Switzerland, 2010.

[7] M. A. L. Marques and E. K. U. Gross, “Time-dependent density functional theory,” *Annu Rev Phys Chem*, vol. 55, pp. 427–455, June 2004.

[8] E. Runge and E. K. U. Gross, “Density-functional theory for time-dependent systems,” *Physical Review Letters*, vol. 52, no. 12, pp. 997–1000, March 1984.

[9] T. D. Kühne, M. Iannuzzi, M. D. Ben, V. V. Rybkin, P. Seewald, F. Stein, T. Laino, R. Z. Khaliullin, O. Schütt, F. Schiffmann, D. Golze, J. Wilhelm, S. Chulkov, M. H. Bani-Hashemian, V. Weber, U. Borštnik, M. Taillefumier, A. S. Jakobovits, A. Lazzaro, H. Pabst, T. Müller, R. Schade, M. Guidon, S. Andermatt, N. Holmberg, G. K. Schenter, A. Hehn, A. Bussy, F. Belleflamme, G. Tabacchi, A. Glöß, M. Lass, I. Bethune, C. J. Mundy, C. Plessl, M. Watkins, J. VandeVondele, M. Krack, and J. Hutter, “CP2K: An electronic structure and molecular dynamics software package - Quickstep: Efficient and accurate electronic structure calculations,” *The Journal of Chemical Physics*, vol. 152, no. 19, p. 194103, May 2020. [Online]. Available: https://doi.org/10.1063/5.0007045

[10] P. M. Gill and P. von Rague Schleyer, “Density functional theory (DFT), Hartree-Fock (HF), and the self-consistent field,” *J. Chem. Phys*, vol. 100, pp. 5066–5075, 1994.

[11] *Time-Dependent Coupled Perturbed Hartree–Fock and Density-Functional-Theory Approach for Calculating Frequency-Dependent (Hyper)Polarizabilities with Nonorthogonal Localized Molecular Orbitals*, vol. 13, no. 9, Aug. 2017. [Online]. Available: https://doi.org/10.1021/acs.jctc.7b00321

[12] L. H. Thomas, “The calculation of atomic fields,” *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 23, no. 5, pp. 542–548, 1927.

[13] E. H. Lieb and B. Simon, “The thomas-fermi theory of atoms, molecules and solids,” *Advances in Mathematics*, vol. 23, no. 1, pp. 22–116, 1977. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0001870877901086

[14] P. Hohenberg and W. Kohn, “Inhomogeneous electron gas,” *Phys. Rev.*, vol. 136, pp. B864–B871, Nov 1964. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRev.136.B864

[15] W. Kohn and L. J. Sham, “Self-consistent equations including exchange and correlation effects,” *Phys. Rev.*, vol. 140, pp. A1133–A1138, Nov 1965. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRev.140.A1133

[16] W. Lynch, “Private discussion,” March 2023, python DFT program.

[17] A. Szabo and N. S. Ostlund, *Modern Quantum Chemistry*, ser. Dover Books on Chemistry. Mineola, NY: Dover Publications, Jan. 1996.

[18] K. Lopata and N. Govind, “Modeling fast electron dynamics with real-time time-dependent density functional theory: Application to small molecules and chromophores,” *Journal of Chemical Theory and Computation*, vol. 7, no. 5, pp. 1344–1355, Apr. 2011. [Online]. Available: https://doi.org/10.1021/ct200137z

[19] K. Yabana, T. Nakatsukasa, J.-I. Iwata, and G. F. Bertsch, “Real-time, real-space implementation of the linear response time-dependent density-functional theory,” *physica status solidi (b)*, vol. 243, no. 5, pp. 1121–1138, Apr. 2006. [Online]. Available: https://doi.org/10.1002/pssb.200642005

[20] A. Castro, M. A. L. Marques, and A. Rubio, “Propagators for the time-dependent kohn–sham equations,” *The Journal of Chemical Physics*, vol. 121, no. 8, pp. 3425–3433, Aug. 2004. [Online]. Available: https://doi.org/10.1063/1.1774980

[21] F. Paillusson, “Statistical Mechanics: The Rosetta Stone of Physics,” February 2022, Lecture notes.

[22] J. Crank and P. Nicolson, “A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type,” *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 43, no. 1, pp. 50–67, 1947.

[23] X. Andrade, C. D. Pemmaraju, A. Kartsev, J. Xiao, A. Lindenberg, S. Rajpurohit, L. Z. Tan, T. Ogitsu, and A. A. Correa, “Inq, a Modern GPU-Accelerated Computational Framework for (Time-Dependent) Density Functional Theory,” *Journal of Chemical Theory and Computation*, vol. 17, no. 12, pp. 7447–7467, 12 2021. [Online]. Available: https://doi.org/10.1021/acs.jctc.1c00562

[24] W. Magnus, “On the exponential solution of differential equations for a linear operator,” *Communications on Pure and Applied Mathematics*, vol. 7, no. 4, pp. 649–673, Nov. 1954. [Online]. Available: https://doi.org/10.1002/cpa.3160070404

[25] N. Auer, L. Einkemmer, P. Kandolf, and A. Ostermann, “Magnus integrators on multicore CPUs and GPUs,” *Computer Physics Communications*, vol. 228, pp. 115–122, jul 2018. [Online]. Available: https://doi.org/10.1016%2Fj.cpc.2018.02.019

[26] Y. Zhu and J. M. Herbert, “Self-consistent predictor/corrector algorithms for stable and efficient integration of the time-dependent kohn-sham equation,” *The Journal of Chemical Physics*, vol. 148, no. 4, p. 044117, Jan. 2018. [Online]. Available: https://doi.org/10.1063/1.5004675

[27] J. Hekele, Y. Yao, Y. Kanai, V. Blum, and P. Kratzer, “All-electron real-time and imaginary-time time-dependent density functional theory within a numeric atom-centered basis function framework,” *The Journal of Chemical Physics*, vol. 155, no. 15, p. 154801, Oct. 2021. [Online]. Available: https://doi.org/10.1063/5.0066753

[28] CP2K, “howto:converging_cutoff [CP2K Open Source Molecular Dynamics] — cp2k.org,” https://www.cp2k.org/howto:converging_cutoff?do=, Weinheim, FRG, pp. 239–263, 2020, [Accessed 02-May-2023].

[29] SinGroup, “PyCP2K,” https://github.com/SINGROUP/pycp2k, 2022.

[30] C. Shannon, “Communication in the presence of noise,” *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, Jan. 1949. [Online]. Available: https://doi.org/10.1109/jrproc.1949.232969

[31] D. S. Sabirov and I. S. Shepelevich, “Information entropy in chemistry: An overview,” *Entropy*, vol. 23, no. 10, p. 1240, Sep. 2021. [Online]. Available: https://doi.org/10.3390/e23101240

[32] M. A. Thompson and M. Watkins, “PyRTP,” https://github.com/Matt-A-Thompson/PyRTP, May 2023.

[33] W. Koch and M. C. Holthausen, “Chemical reactivity: Exploring potential energy surfaces,” in *A Chemist’s Guide to Density Functional Theory.* Weinheim, FRG: Wiley-VCH Verlag GmbH, 2003, pp. 239–263.

[34] U. von Barth and L. Hedin, “A local exchange-correlation potential for the spin polarized case. i,” *Journal of Physics C: Solid State Physics*, vol. 5, no. 13, p. 1629, jul 1972. [Online]. Available: https://dx.doi.org/10.1088/0022-3719/5/13/012

[35] C. A. Ullrich, *Time-Dependent Density-Functional Theory: Concepts and Applications.* Oxford University Press, Dec. 2011.

[36] J. Ihm, A. Zunger, and M. L. Cohen, “Momentum-space formalism for the total energy of solids,” *Journal of Physics C: Solid State Physics*, vol. 12, no. 21, p. 4409, nov 1979. [Online]. Available: https://dx.doi.org/10.1088/0022-3719/12/21/009

[37] E. Meijering, “A chronology of interpolation: from ancient astronomy to modern signal and image processing,” *Proceedings of the IEEE*, vol. 90, no. 3, pp. 319–342, March 2002. [Online]. Available: https://doi.org/10.1109%2F5.993400

A Appendices

A.1 Proofs of the Hohenberg-Kohn Theorems

This proof is adapted from the Hohenberg and Kohn paper from 1964 [14], with updated, modern notation for ease of understanding.

Theorem 1

For Theorem 1, we first define the Hamiltonian of a non-degenerate ground state system (that being a system of electrons enclosed by some volume under some external potential v_{ext}) as:

$$\hat{H} = \hat{T} + \hat{V} + \hat{U} \tag{82}$$

Such that $\hat{T}$ is the kinetic energy of the system, $\hat{V}$ is the potential as a result of the external potential v_{ext} and $\hat{U}$ is the Coulombic interaction between the electrons within the system. The kinetic energy term $\hat{T}$ is given by the following relation:

$$\hat{T} = \frac{1}{2} \int \nabla \psi^*(\boldsymbol{r}) \nabla \psi(\boldsymbol{r}) \, \mathrm{d}\boldsymbol{r} \tag{83}$$

Where $\psi(\boldsymbol{r})$ is a single electron wavefunction. The potential term $\hat{V}$ is given by the following equation:

$$\hat{V} = \int v_{ext} \, \psi^*(\boldsymbol{r}) \psi(\boldsymbol{r}) \, \mathrm{d}\boldsymbol{r} \tag{84}$$

Finally, the Coulomb interaction term $\hat{U}$ is given by the following equation:

$$\hat{U} = \frac{1}{2} \int \frac{1}{|\boldsymbol{r} - \boldsymbol{r}'|} \psi^*(\boldsymbol{r}') \psi(\boldsymbol{r}') \psi^*(\boldsymbol{r}) \psi(\boldsymbol{r}) \, \mathrm{d}\boldsymbol{r} \tag{85}$$

We may now define the electron density $\rho(\boldsymbol{r})$ as the following[4]:

$$\rho(\boldsymbol{r}) = \langle \Psi | \psi^*(\boldsymbol{r}) \psi(\boldsymbol{r}) | \Psi \rangle \tag{86}$$

Given that Ψ has a unique solution to the time-dependent Schrödinger equation ($\hat{H}$), it should be clear that $\rho(\boldsymbol{r})$ is a functional of v_{ext}.

[4]The Hohenberg-Kohn paper (1964) uses slightly different notation, most notably the lack of bra-ket notation and the use of $n(\boldsymbol{r})$ as opposed to $\rho(\boldsymbol{r})$. Much of this proof was compared to proofs given in '*A Chemist's Guide to Density Functional Theory*' [33] and updated using modern notation.

We now seek to show that v_{ext} is a unique functional of $\rho(\boldsymbol{r})$ as well. We thus consider a new external potential v'_{ext} and corresponding wavefunction Ψ', with the same electron density. We may now determine the ground state energy as the following:

$$E_0 = \langle\Psi|\hat{H}|\Psi\rangle \tag{87}$$

From the variational principle, we know that the ground state energy E_0 is always less than or equal to the expectation value of the Hamiltonian for any trial wavefunction. Thus, the following holds:

$$E_0 < E'_0 = \langle\Psi'|\hat{H}|\Psi'\rangle = \langle\Psi'|\hat{H}'|\Psi'\rangle + \langle\Psi'|\hat{V} - \hat{V}'|\Psi'\rangle \tag{88}$$

This evaluates to equal:

$$E_0 < E'_0 + \int\{v_{ext} - v'_{ext}\}\rho(\boldsymbol{r})\,\mathrm{d}\boldsymbol{r} \tag{89}$$

By swapping the prime and non-prime energies, we find that:

$$E'_0 < E_0 - \int\{v_{ext} - v'_{ext}\}\rho(\boldsymbol{r})\,\mathrm{d}\boldsymbol{r} \tag{90}$$

Adding these two functions together gives that:

$$E_0 + E'_0 < E'_0 + E_0 \tag{91}$$

Thus, via ‘reductio ad absurdium’, we find that Theorem 1 is proven to be true.

Theorem 2

For the purposes of the proof of this theorem, we define a functional of the electron density alone, given as the following:

$$F[\rho(\boldsymbol{r})] = \langle\Psi|\hat{T} + \hat{U}|\Psi\rangle \tag{92}$$

We may now write the energy functional as the following:

$$E[\rho(\boldsymbol{r})] = \int\{v_{ext}\}\rho(\boldsymbol{r})\,\mathrm{d}\boldsymbol{r} + F[\rho(\boldsymbol{r})] \tag{93}$$

Or, in bra-ket notation:

$$E[\Psi] = \langle\Psi|\hat{V}|\Psi\rangle + \langle\Psi|\hat{T} + \hat{U}|\Psi\rangle \tag{94}$$

It can be easily shown that when Ψ is equal to the ground-state wavefunction, $E[\Psi]$ is minimised. We also know from Theorem 1 that there is a one-to-one mapping between external potentials and wavefunctions. Thus, we define Ψ' as the wavefunction associated with a different external potential v'_{ext}. From there, we find that:

$$E[\Psi'] > E[\Psi] \tag{95}$$

Given in Theorem 1 we have proven that there is a one-to-one mapping between wavefunctions and electron densities, we thus show that the functional $E[\rho(\boldsymbol{r}]$ is equal to $E[\Psi]$, and $E[\rho(\boldsymbol{r}]$ is minimised when ρ is equal to the true ground-state electron density.

A.2 XC energy approximation of spin-polarised systems

In certain systems where spin polarisation is significant, such as in ferromagnets, the XC energy approximation is affected, and thus a more suitable approximation should be used. In the case of the local density spin approximation, it takes the following form:

$$E_{xc}^{LDSA}[\rho_\uparrow, \rho_\downarrow] = \int \rho(\boldsymbol{r})\epsilon_{xc}(\rho_\uparrow, \rho_\downarrow)\, \mathrm{d}\boldsymbol{r} \tag{96}$$

As shown, the electron density is split, such that $\rho(r) = \rho_\uparrow + \rho_\downarrow$, with each spin treated simultaneously. One may treat the generalised gradient approximations in the same way, giving the following result:

$$E_{xc}^{GGSA}[\rho_\uparrow, \rho_\downarrow] = \int \rho(\boldsymbol{r})\epsilon_{xc}\left(\rho_\uparrow, \rho_\downarrow, \nabla\rho_\uparrow, \nabla\rho_\downarrow\right)\, \mathrm{d}\boldsymbol{r} \tag{97}$$

Further approximations deal with this in a similar fashion. Exact determination of the exchange term is simple, given that the simple relationship between the spin densities and the total electron density. Thus, the exchange term for the LDSA can be simply determined as:

$$E_{x}^{LDSA}[\rho_\uparrow, \rho_\downarrow] = \frac{1}{2}\left(E_x[2\rho_\uparrow], E_x[2\rho_\downarrow]\right) \tag{98}$$

Such that $E_x[2\rho_\uparrow]$ and $E_x[2\rho_\downarrow]$ are determined via Equation 8. The correlation energy is similarly determined via a range of approximation methods, some examples of which can be found in the von Barth and Hedin paper [34].

A.3 A proof of the first Runge-Gross theorem

Given that the first Runge-Gross theorem is essentially the foundation of TDDFT, it is of critical importance to prove that this theorem is true, and to acknowledge the steps in proving it to be true, so as to truly understand *why* TDDFT works. This proof is adapted from the original paper by Runge and Gross [8] and intends to put the proof in a concise and easy-to-read format.

We use the method of proof by contradiction (given we are proving a one-to-one mapping between the external potential $v_{ext}(\boldsymbol{r}, t)$ and the electron density $\rho(\boldsymbol{r}, t)$). We therefore let $v_{ext}(\boldsymbol{r}, t)$ and $v'_{ext}(\boldsymbol{r}, t)$ be two external potentials where there is a not insignificant difference between the two, i.e. $v_{ext}(\boldsymbol{r}, t) - v'_{ext}(\boldsymbol{r}, t) \neq c(t)$. Given we are proving that $\rho(\boldsymbol{r}, t)$ is unique for all values of t, we must first show that $v_{ext}(\boldsymbol{r}, t)$ and $v'_{ext}(\boldsymbol{r}, t)$ are unique for all values of t. The most logical exception one may see to this is that the two external potentials may be equal when $t = t_0$. This is the rational for the stipulation that $v_{ext}(\boldsymbol{r}, t)$ must be able to be expanded into a Taylor series around

$t = t_0$, as if this is true, then there must exist a non-negative integer such that the following relation holds:

$$\frac{\partial^k}{\partial t^k}\left[v_{ext}(\boldsymbol{r},t) - v'_{ext}(\boldsymbol{r},t)\right]\Big|_{t=t_0} \neq \text{constant} \tag{99}$$

Using this, we may show that the corresponding current densities $j(\boldsymbol{r},t)$ and $j'(\boldsymbol{r},t)$ are unique. We thus define the equation of motion for this system to be:

$$i\frac{d}{dt}\langle\Phi(t)|\hat{O}(t)|\Phi(t)\rangle = \langle\Phi(t)|i\frac{\partial}{\partial t}\hat{O}(t) + [\hat{O}(t),\hat{H}(t)]|\Phi(t)\rangle \tag{100}$$

Where $\Phi(t)$ is the time-dependent wavefunction and $\hat{O}(t)$ is any time-dependent observable. We may then substitute in $\hat{j}(\boldsymbol{r},t)$ as the following:

$$\boldsymbol{j}(\boldsymbol{r},t) = \langle\Phi(t)|\hat{j}(\boldsymbol{r})|\Phi(t)\rangle \tag{101}$$

Thus, we obtain:

$$i\frac{\partial}{\partial t}\boldsymbol{j}(\boldsymbol{r},t) = \langle\Phi(t)|[\hat{j}(\boldsymbol{r}),\hat{H}(t)]|\Phi(t)\rangle \tag{102}$$

We may now make a substitution in terms of the initial state (t_0, Φ_0, etc.), as both $\Phi(t)$ and $\Phi'(t)$ must have evolved from the initial state Φ_0, giving the following:

$$i\frac{\partial}{\partial t}\left[\boldsymbol{j}(\boldsymbol{r},t) - \boldsymbol{j}'(\boldsymbol{r},t)\right]\Big|_{t=t_0} = \langle\Phi_0|[\hat{j}(\boldsymbol{r}),\hat{H}(t_0) - \hat{H}'(t_0)]|\Phi_0\rangle \tag{103}$$

We may then solve the right-hand side, giving the following:

$$i\frac{\partial}{\partial t}\left[\boldsymbol{j}(\boldsymbol{r},t) - \boldsymbol{j}'(\boldsymbol{r},t)\right]\Big|_{t=t_0} = i\,\rho(\boldsymbol{r},t_0)\nabla[v_{ext}(\boldsymbol{r},t_0) - v'_{ext}(\boldsymbol{r},t_0)] \tag{104}$$

It can be shown simply that if $v_{ext}(\boldsymbol{r},t_0) \neq v'_{ext}(\boldsymbol{r},t_0)$ for $k = 0$, then, $\boldsymbol{j}(\boldsymbol{r},t) \neq \boldsymbol{j}'(\boldsymbol{r},t)$ shortly after t_0. For higher values of k, we may substitute in Equation 99 into the previous equation, giving:

$$\left(i\frac{\partial}{\partial t}\right)^{k+1}\left[\boldsymbol{j}(\boldsymbol{r},t_0) - \boldsymbol{j}'(\boldsymbol{r},t_0)\right]_{t=t_0} = i\,\rho(\boldsymbol{r},t_0)\nabla\left\{\left(i\frac{\partial}{\partial t}\right)^{k}\left[v_{ext}(\boldsymbol{r},t) - v'_{ext}(\boldsymbol{r},t)\right]_{t=t_0}\right\} \neq 0 \tag{105}$$

Using the above proof, if $v_{ext}(\boldsymbol{r},t_0) \neq v'_{ext}(\boldsymbol{r},t_0)$ for any value of k, then $\boldsymbol{j}(\boldsymbol{r},t) \neq \boldsymbol{j}'(\boldsymbol{r},t)$ shortly after t_0, as required. In order to connect this to the electron density, we make use of the continuity equation, given by the following:

$$\frac{\partial}{\partial t}\left[\rho(\boldsymbol{r},t) - \rho'(\boldsymbol{r},t)\right] = -\text{div}\left[\boldsymbol{j}(\boldsymbol{r},t_0) - \boldsymbol{j}'(\boldsymbol{r},t_0)\right] \tag{106}$$

We may now make a substitution of the above equation into Equation 105, giving the following:

$$\frac{\partial^{k+2}}{\partial t^{k+2}}\left[\rho(\boldsymbol{r},t)-\rho'(\boldsymbol{r},t)\right]_{t=t_0} = -\mathrm{div}\rho(\boldsymbol{r},t_0)\cdot\nabla\left\{\frac{\partial^k}{\partial t^k}\left[v_{ext}(\boldsymbol{r},t)-v'_{ext}(\boldsymbol{r},t)\right]_{t=t_0}\right\} \quad (107)$$

We now have a function directly relating $\rho(\boldsymbol{r},t)$ to $v_{ext}(\boldsymbol{r},t)$. We now must show that the above equation cannot equal zero if Equation 99 holds. We thus introduce the assumption that $\mathrm{div}[\rho(\boldsymbol{r},t_0)\nabla u(\boldsymbol{r})] = 0$, such that $u(\boldsymbol{r}) \neq \mathrm{const.}$. Thus, the above equation becomes:

$$\int \mathrm{d}^3r\, u(\boldsymbol{r})\,\mathrm{div}[\rho(\boldsymbol{r},t_0)\nabla u(\boldsymbol{r})] = -\int \mathrm{d}^3r\rho(\boldsymbol{r},t_0)[\nabla u(\boldsymbol{r})]^2 + \frac{1}{2}\oint \rho(\boldsymbol{r},t_0)[\nabla u^2(\boldsymbol{r})]\cdot\mathrm{d}\boldsymbol{f} = 0 \quad (108)$$

For $\rho(\boldsymbol{r},t_0)[\nabla u(\boldsymbol{r})]^2 = 0$, $\rho(\boldsymbol{r},t_0)$ must quickly become zero, which contradicts $u(\boldsymbol{r}) \neq$ const. (so long as $\rho(\boldsymbol{r},t_0)$ is well behaved). Thus, by reductio ad absurdum, the right hand side of Equation 108 cannot equal zero, meaning $\rho(\boldsymbol{r},t)$ may not equal $\rho'(\boldsymbol{r},t)$ shortly after t_0. This proves that there is a one-to-one mapping between external potentials $v_{ext}(\boldsymbol{r},t)$ and electron densities $\rho(\boldsymbol{r},t)$.

A.4 CP2K RT-TDDFT input file

```
&GLOBAL
  PROJECT HeH+efield
  RUN_TYPE  RT_PROPAGATION
  PRINT_LEVEL LOW
&END GLOBAL
&FORCE_EVAL
  METHOD Quickstep
  &DFT
    CHARGE +1
    LSD
    BASIS_SET_FILE_NAME BASIS_SET
    POTENTIAL_FILE_NAME POTENTIAL
    !RESTART_FILE_NAME HeH+_rtp-RESTART.rtpwfn
    &POISSON
      PERIODIC NONE
      PSOLVER  WAVELET
    &END POISSON
    &MGRID
      CUTOFF 500
      REL_CUTOFF 60
    &END MGRID
    &QS
    &END QS
    &SCF
      MAX_SCF 100
      SCF_GUESS ATOMIC
      EPS_SCF 1.0E-6
    &END SCF
    &XC
      &XC_FUNCTIONAL PADE
      &END XC_FUNCTIONAL
    &END XC

    &EFIELD
      INTENSITY 0.001
      POLARISATION 1.0 0.0 0.0
      WAVELENGTH 210
      ENVELOP GAUSSIAN
      &GAUSSIAN_ENV
        SIGMA 0.2
        TO  1.0
      &END
    &END
    &REAL_TIME_PROPAGATION
       PERIODIC .FALSE.
       COM_NL .FALSE.
       MAX_ITER 50
       MAT_EXP PADE
       EXP_ACCURACY 1.0E-10
       EPS_ITER 1.0E-9
       PROPAGATOR EM
       INITIAL_WFN SCF_WFN ! RT_RESTART
    &END
    &PRINT
      &MOMENTS
        PERIODIC .FALSE.
        FILENAME dipole
```

```
        COMMON_ITERATION_LEVELS 2
      &END
    &END
  &END DFT
  &SUBSYS
    &CELL
      ABC 10 10 10
    &END CELL
    &TOPOLOGY
      &CENTER_COORDINATES
      &END
    &END
    &COORD
    He    0.000000     0.000000    0.000000 HeH
    H    0.000000    0.000000     1.463200 HeH
    &END COORD
    &KIND H
      BASIS_SET DZVP-GTH-PADE
      POTENTIAL GTH-PADE-q1
    &END KIND
    &KIND He
      BASIS_SET DZVP-GTH-PADE
      POTENTIAL GTH-PADE-q2
    &END KIND
  &END SUBSYS
&END FORCE_EVAL
&MOTION
  &MD
    STEPS 2000
    TIMESTEP [au_t] 0.1
  &END
&END MOTION
!&EXT_RESTART
!   RESTART_FILE_NAME HeH+_rtp-1.restart
!&END
```

A.5 PyRTP source code

```
#%%
#Package imports
import numpy as np
from npy_append_array import NpyAppendArray #used to save data as
↪  external arrays regularly in case of failure to run
import math
import scipy
import matplotlib.pyplot as plt
from pyscf import gto, dft, lib
from sympy import Matrix
from scipy.interpolate import lagrange
import time
import os
# the number of threads can be specified by uncommenting one of the
↪  following functions,
# run 'np.show_config()' to determine which to use.
#os.environ['OMP_NUM_THREADS']='8'
#os.environ['OPENBLAS_NUM_THREADS']='8'
#os.environ['MPI_NUM_THREADS']='8'
#os.environ['MKL_NUM_THREADS']='8'

#%%
# FunctionCalls
def GridCreate(L,N_i):

        N = N_i**3
        dr = (L/N_i)**3

        r_x = np.linspace(-L/2.,L/2.,int(N_i),endpoint=False)
        r_y = np.linspace(-L/2.,L/2.,int(N_i),endpoint=False)
        r_z = np.linspace(-L/2.,L/2.,int(N_i),endpoint=False)

        return r_x,r_y,r_z,N,dr

#a_GTO = exponent, r_GTO = grid position vector, R_GTO = nuclei
↪  position vector
```

```
def GTO(a_GTO,r_GTO,R_GTO): return
 ↪  (2*a_GTO/np.pi)**(3/4)*np.exp(-a_GTO*(np.linalg.norm(r_GTO -
 ↪  R_GTO))**2)

def construct_GTOs(nuc,N,N_i,r_x,r_y,r_z,R_I,alpha) :

    #create a matrix of grid points for each of the three GTOs,
     ↪  initialised to zero.
    GTO_p = np.zeros(3*N).reshape(3,N_i,N_i,N_i)

    # Currently only working for HeH+, however this would be simple
     ↪  to change for other molecules
    if nuc == 'He' :
        r_n = R_I[0]
        alpha_n = alpha[0]
    if nuc == 'H' :
        r_n = R_I[1]
        alpha_n = alpha[1]

    #Loop through GTOs and grid points, calculate the GTO value and
     ↪  assign to GTO_p.
    for gto in range(0,3) :
        #for i,j,k in range(0,N_i) :
        for i in range(0,N_i) :
            for j in range(0,N_i) :
                for k in range(0,N_i) :
                    p = np.array([r_x[i],r_y[j],r_z[k]]) #Select
                     ↪  current grid position vector.

                    GTO_p[gto][i][j][k] = GTO(alpha_n[gto],p,r_n)
                     ↪  #calculate GTO value using GTO function call.

    return GTO_p

def construct_CGF(GTOs,N,N_i,Coef) :

    CGF = np.zeros(N).reshape(N_i,N_i,N_i) #create a matrix of grid
     ↪  points initialised to zero.
    for g in range(0,len(GTOs)) : CGF += Coef[g]*GTOs[g] #construct
    ↪  the CGF from the GTOs and coefficients, Eq. 2.

    return CGF

def calculate_realspace_density(CGF_0,CGF_1,N,N_i,P,dr) :

    n_el_r = np.zeros(N).reshape(N_i,N_i,N_i)
    n_el_total = 0

    for i in range(0,N_i) :
        for j in range(0,N_i) :
            for k in range(0,N_i) :

                n_el_r[i][j][k] = P[0][0]*CGF_0[i][j][k]**2 +
                ↪  P[0][1]*CGF_0[i][j][k]*CGF_1[i][j][k] +\
                    P[1][0]*CGF_1[i][j][k]*CGF_0[i][j][k] +
                    ↪  P[1][1]*CGF_1[i][j][k]**2

                n_el_total += n_el_r[i][j][k]*dr

    return n_el_r, n_el_total

def calculate_core_density(N,N_i,Z_I,r_x,r_y,r_z,R_I) :

    R_pp = np.sqrt(2.)/5.

    n_c_r = np.zeros(N).reshape(N_i,N_i,N_i)

    for i in range(0,N_i) :
        for j in range(0,N_i) :
            for k in range(0,N_i) :
                r = np.array([r_x[i],r_y[j],r_z[k]])

                for n in range(0,len(Z_I)) :
                    n_c_r[i][j][k] += -Z_I[n]/(R_pp**3)*np.pi**(-3/2)*
                    ↪  np.exp(-((r-R_I[n])/R_pp).dot((r-R_I[n])/R_pp))
    return n_c_r

def grid_integration(V_r,dr,CGF_He,CGF_H) :

    V = [[0.,0.],[0.,0.]]

    for i in range(0,len(V_r)) :
        for j in range(0,len(V_r)) :
            for k in range(0,len(V_r)) :

                #Integrate over grid points
                V[0][0] += V_r[i][j][k]*CGF_He[i][j][k]**2*dr
                V[0][1] +=
                ↪  V_r[i][j][k]*CGF_He[i][j][k]*CGF_H[i][j][k]*dr
                V[1][0] +=
                ↪  V_r[i][j][k]*CGF_H[i][j][k]*CGF_He[i][j][k]*dr
                V[1][1] += V_r[i][j][k]*CGF_H[i][j][k]**2*dr

    return V

def energy_calculation(V,P) :

    E = P[0][0]*V[0][0] + P[0][1]*V[0][1] + P[1][0]*V[1][0] +
    ↪  P[1][1]*V[1][1]

    return E

def calculate_overlap(N_i,dr,CGF_He,CGF_H) :

    S = [[0.,0.],[0.,0.]] #initialise a 2 by 2 matrix of zeros.

    #Loop through grid points
    for i in range(0,N_i) :
        for j in range(0,N_i) :
            for k in range(0,N_i) :
                #Sum the overlap contributions from each CGF as in
                ↪  Eq. 3
                S[0][0] += CGF_He[i][j][k]**2*dr
                S[0][1] += CGF_He[i][j][k]*CGF_H[i][j][k]*dr
                S[1][0] += CGF_H[i][j][k]*CGF_He[i][j][k]*dr
                S[1][1] += CGF_H[i][j][k]**2*dr

    return S

def calculate_kinetic_derivative(N,N_i,G_u,G_v,G_w,PW_He_G,PW_H_G,L) :

    delta_T = [[0.,0.],[0.,0.]]

    for i in range(0,N_i) :
        for j in range(0,N_i) :
            for k in range(0,N_i) :

                g = np.array([G_u[i],G_v[j],G_w[k]]) #Select current G
                ↪   vector

                #Calculate Kinetic energy matrix as per Eq. 7
                #np.real is required here as PWs are complex.
                #The complex component is ~zero after taking
                ↪   PW_He_G^2 but the result still contains a complex
                ↪   term

                delta_T[0][0] +=
                ↪   0.5*L**3/N**2*np.dot(g,g)*np.real(np.dot(
                ↪   np.conjugate(PW_He_G[i][j][k]),PW_He_G[i][j][k]))
                delta_T[0][1] +=
                ↪   0.5*L**3/N**2*np.dot(g,g)*np.real(np.dot(
                ↪   np.conjugate(PW_He_G[i][j][k]),PW_H_G[i][j][k]))
                delta_T[1][0] +=
                ↪   0.5*L**3/N**2*np.dot(g,g)*np.real(np.dot(
                ↪   np.conjugate(PW_H_G[i][j][k]),PW_He_G[i][j][k]))
                delta_T[1][1] +=
                ↪   0.5*L**3/N**2*np.dot(g,g)*np.real(np.dot(
                ↪   np.conjugate(PW_H_G[i][j][k]),PW_H_G[i][j][k]))


    return delta_T
```

```
def calculate_hartree_reciprocal(n_G,N,N_i,r_x,r_y,r_z,G_u,G_v,G_w,L)
 ↪  :

    nG = np.fft.fftshift(n_G) #n_G is shifted to match same frequency
     ↪  domain as G (-pi,pi) instead of (0,2pi)
    Vg = np.complex128(np.zeros(N).reshape(N_i,N_i,N_i))
    E_hart_G = 0. ## Hartree energy in reciprocal space

    for i in range(0,N_i) :
        for j in range(0,N_i) :
            for k in range(0,N_i) :

                R_vec = np.array([r_x[i],r_y[j],r_z[k]])
                G_vec = np.array([G_u[i],G_v[j],G_w[k]])

                if np.dot(G_vec,G_vec) < 0.01 :  continue #can't
                 ↪  divide by zero

                Vg[i][j][k] = 4*np.pi*nG[i][j][k]/np.dot(G_vec,G_vec)
                E_hart_G += np.conjugate(nG[i][j][k])*Vg[i][j][k]

    E_hart_G *= L**3/N**2*0.5

    return np.fft.ifftshift(Vg), E_hart_G #result is shifted back.

def calculate_hartree_real(N_i,V_r,n_r,dr) :

    E_hart_r = 0.

    for i in range(0,N_i) :
        for j in range(0,N_i) :
            for k in range(0,N_i) :
                E_hart_r += 0.5*V_r[i][j][k]*n_r[i][j][k]*dr

    return E_hart_r

def calculate_XC_potential(N,N_i,n_el_r):
    V_XC_r = np.zeros(N).reshape(N_i,N_i,N_i)

    for i in range(0,N_i) :
        for j in range(0,N_i) :
            #Loop through first two axis and obtain a list of rho(r)
            ↪  for the third axis
            V_XC_r[i][j] =
            ↪  dft.xcfun.eval_xc('LDAERF',n_el_r[i][j])[1][0] #get
            ↪  the XC term for each n(r) in the list

    return V_XC_r

def calculate_XC_energy(N_i,n_el_r,dr):

    E_XC = 0.

    for i in range(0,N_i) :
        for j in range(0,N_i) :
            XC_ene = dft.xcfun.eval_xc('LDAERF',n_el_r[i][j])[0]
            for k in range(0,N_i) :
                E_XC += XC_ene[k]*n_el_r[i][j][k]*dr

    return E_XC

def calculate_V_SR_r(N,N_i,r_x,r_y,r_z,Z_I,Cpp,R_I) :

    V_SR_r = np.zeros(N).reshape(N_i,N_i,N_i)
    alpha_pp = 5./np.sqrt(2.) #pseudopotential parameter, alpha_pp =
    ↪  1/R^{c}_{I}

    for i in range(0,len(V_SR_r)) :
        for j in range(0,len(V_SR_r)) :
            for k in range(0,len(V_SR_r)) :
                R_vec = np.array([r_x[i],r_y[j],r_z[k]])
                for n in range(0,len(Z_I)) : #loop over nuclei
                    r = np.linalg.norm(R_vec - R_I[n])
                for c in range(0,len(Cpp)) : #loop over values of
                ↪ Cpp
                    V_SR_r[i][j][k] +=
                    ↪ Cpp[n][c]*(np.sqrt(2.)*alpha_pp*r)**
                    ↪ (2*(c+1)-2)*np.exp(-(alpha_pp*r)**2)

    return V_SR_r

def calculate_self_energy(Z_I) :

    R_pp = np.sqrt(2.)/5. #R_{I}^{c}

    E_self = 0.

    for n in range(0,len(Z_I)) :
        E_self += -(2*np.pi)**(-1/2)*Z_I[n]**2/R_pp

    return E_self

def calculate_Ion_interaction(Z_I,R_I) :

    R_pp = np.sqrt(2.)/5. #R_{I}^{c}

    E_II = Z_I[0]*Z_I[1]/np.linalg.norm(R_I[0]-R_I[1])*
    ↪ math.erfc(np.linalg.norm(R_I[0]-R_I[1])/
    ↪ np.sqrt(R_pp**2+R_pp**2))

    return E_II

def dftSetup(R_I,alpha,Coef,L,N_i,Z_I):
    r_x,r_y,r_z,N,dr=GridCreate(L,N_i)
    GTOs_He = construct_GTOs('He',N,N_i,r_x,r_y,r_z,R_I,alpha)
    CGF_He = construct_CGF(GTOs_He,N,N_i,Coef)
    GTOs_H = construct_GTOs('H',N,N_i,r_x,r_y,r_z,R_I,alpha)
    CGF_H = construct_CGF(GTOs_H,N,N_i,Coef)
    G_u = np.linspace(-N_i*np.pi/L,N_i*np.pi/L,N_i,endpoint=False)
    G_v = np.linspace(-N_i*np.pi/L,N_i*np.pi/L,N_i,endpoint=False)
    G_w = np.linspace(-N_i*np.pi/L,N_i*np.pi/L,N_i,endpoint=False)
    PW_He_G = np.fft.fftshift(np.fft.fftn(CGF_He))
    PW_H_G = np.fft.fftshift(np.fft.fftn(CGF_H))
    S = calculate_overlap(N_i,dr,CGF_He,CGF_H)
    delta_T =
     ↪  calculate_kinetic_derivative(N,N_i,G_u,G_v,G_w,PW_He_G,PW_H_G,L)
    E_self = calculate_self_energy(Z_I)
    E_II = calculate_Ion_interaction(Z_I,R_I)

    return r_x,r_y,r_z,N,dr,GTOs_He,CGF_He,GTOs_H,CGF_H, G_u,G_v,G_w,
     ↪  PW_He_G,PW_H_G,S,delta_T,E_self,E_II

def computeE_0(R_I,Z_I,P,N_i,Cpp,r_x,r_y,r_z,N,dr,CGF_He,CGF_H,
 ↪  G_u,G_v,G_w,delta_T,E_self,E_II,L):

    n_el_r, n_el_r_tot  =
     ↪  calculate_realspace_density(CGF_He,CGF_H,N,N_i,P,dr)
    n_c_r = calculate_core_density(N,N_i,Z_I,r_x,r_y,r_z,R_I)
    n_r = n_el_r + n_c_r
    T = energy_calculation(delta_T,P)
    n_G = np.fft.fftn(n_r)
    V_G, E_hart_G =
     ↪  calculate_hartree_reciprocal(n_G,N,N_i,r_x,r_y,r_z,G_u,G_v,G_w,L)
    V_r = np.fft.ifftn(V_G)
    V_hart = grid_integration(V_r,dr,CGF_He,CGF_H)
    E_hart_r = calculate_hartree_real(N_i,V_r,n_r,dr)
    V_XC_r = calculate_XC_potential(N,N_i,n_el_r)
    V_XC = grid_integration(V_XC_r,dr,CGF_He,CGF_H)
    E_XC = calculate_XC_energy(N_i,n_el_r,dr)
    V_SR_r = calculate_V_SR_r(N,N_i,r_x,r_y,r_z,Z_I,Cpp,R_I)
    V_SR = grid_integration(V_SR_r,dr,CGF_He,CGF_H)
    E_SR = energy_calculation(V_SR,P)
    E_0 = E_hart_r + E_XC + E_SR + T + E_self + E_II

    return E_0

def computeDFT(R_I,alpha,Coef,L,N_i,P_init,Z_I,Cpp,iterations,r_x,r_y,
 ↪  r_z,N,dr,GTOs_He,CGF_He,GTOs_H,CGF_H,G_u,G_v,G_w,PW_He_G,PW_H_G,S,
 ↪  delta_T,E_self,E_II):
```

```
P = P_init
n_el_r, n_el_r_tot  =
↪  calculate_realspace_density(CGF_He,CGF_H,N,N_i,P,dr)
n_c_r = calculate_core_density(N,N_i,Z_I,r_x,r_y,r_z,R_I)
n_r = n_el_r + n_c_r
T = energy_calculation(delta_T,P)
n_G = np.fft.fftn(n_r)
V_G, E_hart_G =
↪  calculate_hartree_reciprocal(n_G,N,N_i,r_x,r_y,r_z,G_u,G_v,
↪  G_w,L)
V_r = np.fft.ifftn(V_G)
V_hart = grid_integration(V_r,dr,CGF_He,CGF_H)
E_hart_r = calculate_hartree_real(N_i,V_r,n_r,dr)
V_XC_r = calculate_XC_potential(N,N_i,n_el_r)
V_XC = grid_integration(V_XC_r,dr,CGF_He,CGF_H)
E_XC = calculate_XC_energy(N_i,n_el_r,dr)
V_SR_r = calculate_V_SR_r(N,N_i,r_x,r_y,r_z,Z_I,Cpp,R_I)
V_SR = grid_integration(V_SR_r,dr,CGF_He,CGF_H)
E_SR = energy_calculation(V_SR,P)
KS = np.array(delta_T)+np.array(V_hart)+np.array(V_SR)+
↪  np.array(V_XC)
KS = np.real(KS) #change data type from complex to float,
↪  removing all ~0. complex values
S=Matrix(S)
U,s = S.diagonalize()
s = s**(-0.5)
X = np.matmul(np.array(U,dtype='float64'),
↪  np.array(s,dtype='float64'))
X_dag = np.matrix.transpose(np.array(X,dtype='float64'))

err = 1.0e-6 #The error margin by which convergence of the P
↪  matrix is measured

P = P_init #reset P to atomic guess.
for I in range(0,iterations):
        n_el_r, n_el_r_tot=calculate_realspace_density(CGF_He,
        ↪  CGF_H,N,N_i,P,dr)
```

```
n_c_r =
 ↪  calculate_core_density(N,N_i,Z_I,r_x,r_y,r_z,R_I)
n_r = n_el_r + n_c_r
n_G = np.fft.fftn(n_r)
V_G, E_hart_G = calculate_hartree_reciprocal(n_G,N,N_i,
 ↪  r_x,r_y,r_z,G_u,G_v,G_w,L)
V_r = np.fft.ifftn(V_G)
V_hart = grid_integration(V_r,dr,CGF_He,CGF_H)
E_hart_r = calculate_hartree_real(N_i,V_r,n_r,dr)
V_XC_r = calculate_XC_potential(N,N_i,n_el_r)
V_XC = grid_integration(V_XC_r,dr,CGF_He,CGF_H)
E_XC = calculate_XC_energy(N_i,n_el_r,dr)
V_SR_r =
 ↪  calculate_V_SR_r(N,N_i,r_x,r_y,r_z,Z_I,Cpp,R_I)
V_SR = grid_integration(V_SR_r,dr,CGF_He,CGF_H)
E_SR = energy_calculation(V_SR,P)
E_0 = E_hart_r + E_XC + E_SR + T + E_self + E_II
KS = np.array(delta_T)+np.array(V_hart)+np.array(V_SR)+
 ↪  np.array(V_XC)
KS = np.real(KS)
KS_temp = Matrix(np.matmul(X_dag,np.matmul(KS,X)))
C_temp, e = KS_temp.diagonalize()
C = np.matmul(X,C_temp)
print("iteration : ", I+1, "\n")
print("total energy : ", np.real(E_0), "\n")
print("density matrix : ","\n", P, "\n")
print("KS matrix : ","\n", KS, "\n")
P_new=np.array([[0., 0.],[0., 0.]])
for u in range(0,2) :
        for v in range(0,2) :
                for p in range(0,1) :
                        P_new[u][v] += C[u][p]*C[v][p]
                P_new[u][v] *=2
if abs(P[0][0]-P_new[0][0]) <= err and
 ↪  abs(P[0][1]-P_new[0][1]) <= err and
 ↪  abs(P[1][0]-P_new[1][0]) <= err and
 ↪  abs(P[1][1]-P_new[1][1]) <= err :
        break
```

```
                P = P_new
        return P,np.real(E_0),C,KS

# GaussianKick - provides energy to the system in the form of an
↪  electric pulse
# with a Gaussian envelop (run each propagation step!)
def GaussianKick(KS,scale,direction,t,r_x,r_y,r_z,CGF_He,CGF_H,dr):
    t0=1
    w=0.2
    Efield=np.dot(scale*np.exp((-(t-t0)**2)/(2*(w**2))),direction)
    D_x,D_y,D_z,D_tot=transition_dipole_tensor_calculation(r_x,r_y,r_z,
    ↪  CGF_He,CGF_H,dr)
    V_app=-(np.dot(D_x,Efield[0])+np.dot(D_y,Efield[1])+
    ↪  np.dot(D_z,Efield[2]))
    KS_new=KS+V_app
    return KS_new

# Propagator - using predictor-corrector regime (if applicable)
def propagator(select,H1,H2,dt): #propagator functions
    match select:
        case 'CN':
            U=(1-(1j*(dt/2)*H1))/(1+(1j*(dt/2)*H1))
        case 'EM':
            U=np.exp(-1j*dt*H1)
        case 'ETRS':
            U=np.exp(-1j*(dt/2)*H2)*np.exp(-1j*(dt/2)*H1)
        case 'AETRS':
            U=np.exp(-1j*(dt/2)*H2)*np.exp(-1j*(dt/2)*H1)
        case 'CAETRS':
            U=np.exp(-1j*(dt/2)*H2)*np.exp(-1j*(dt/2)*H1)
        case 'CFM4':
            a1=(3-2*np.sqrt(3))/12
            a2=(3+2*np.sqrt(3))/12
            U=np.dot(np.exp(-1j*dt*a1*H1-1j*dt*a2*H2),
            ↪  np.exp(-1j*dt*a2*H1-1j*dt*a1*H2))
        case _:
            raise TypeError("Invalid propagator")
    return U

def propagate(R_I,Z_I,P,H,C,dt,select,N_i,Cpp,r_x,r_y,r_z,N,dr,CGF_He,
↪   CGF_H,G_u,G_v,G_w,delta_T,E_self,E_II,L,Hprev,t,energies,tnew,i):
    match select:
        case 'CN':
            st=time.time()
            # predictor step first
            if i==0:
                H_p=H
            elif i>0 and i<5:
             H_p=LagrangeExtrapolate(t[0:i],energies[0:i],t[i-1]+(1/2))
            else:
             H_p=LagrangeExtrapolate(t[i-5:i],energies[i-5:i],t[i-1]+
              ↪  (1/2))
            U=np.real(propagator(select,H_p,[],dt))
            C_p=np.dot(U,C)
            # update
            P_p=np.array([[0., 0.],[0., 0.]])
            for u in range(0,2) :
                for v in range(0,2) :
                    for p in range(0,1) :
                        P_p[u][v] += C_p[u][p]*C_p[v][p]
                    P_p[u][v] *=2
            E_0=computeE_0(R_I,Z_I,P_p,N_i,Cpp,r_x,r_y,r_z,N,dr,CGF_He,
             ↪  CGF_H,G_u,G_v,G_w,delta_T,E_self,E_II,L)
            # correct
            H_c=H+(1/2)*(E_0-H)
            U=np.real(propagator(select,H_c,[],dt))
            C_new=np.dot(U,C)
            P_new=np.array([[0., 0.],[0., 0.]])
            for u in range(0,2) :
                for v in range(0,2) :
                    for p in range(0,1) :
                        P_new[u][v] += C_new[u][p]*C_new[v][p]
                    P_new[u][v] *=2
            et=time.time()
```

```
case 'EM':
    st=time.time()
    # predictor step first
    if i==0:
        H_p=H
    elif i>0 and i<5:
     H_p=LagrangeExtrapolate(t[0:i],energies[0:i],t[i-1]+(1/2))
    else:
     H_p=LagrangeExtrapolate(t[i-5:i],energies[i-5:i],t[i-1]+
      ↪  (1/2))
    U=np.real(propagator(select,H_p,[],dt))
    C_p=np.dot(U,C)
    # update
    P_p=np.array([[0., 0.],[0., 0.]])
    for u in range(0,2) :
        for v in range(0,2) :
            for p in range(0,1) :
                P_p[u][v] += C_p[u][p]*C_p[v][p]
            P_p[u][v] *=2
    E_0=computeE_0(R_I,Z_I,P_p,N_i,Cpp,r_x,r_y,r_z,N,dr,CGF_He,
     ↪  CGF_H,G_u,G_v,G_w,delta_T,E_self,E_II,L)
    H_c=H+(1/2)*(E_0-H)
    U=np.real(propagator(select,H_c,[],dt))
    C_new=np.dot(U,C)
    P_new=np.array([[0., 0.],[0., 0.]])
    for u in range(0,2) :
        for v in range(0,2) :
            for p in range(0,1) :
                P_new[u][v] += C_new[u][p]*C_new[v][p]
            P_new[u][v] *=2
    et=time.time()
case 'ETRS':
    st=time.time()
    if i==0:
        H_p=H
    elif i>0 and i<5:
     H_p=LagrangeExtrapolate(t[0:i],energies[0:i],t[i-1]+(1/2))
    else:
```

```
             H_p=LagrangeExtrapolate(t[i-5:i],energies[i-5:i],t[i-1]+
              ↪  (1/2))
            U=np.real(propagator('EM',H_p,[],dt))
            C_p=np.dot(U,C)
            # update
            P_p=np.array([[0., 0.],[0., 0.]])
            for u in range(0,2) :
                for v in range(0,2) :
                    for p in range(0,1) :
                        P_p[u][v] += C_p[u][p]*C_p[v][p]
                    P_p[u][v] *=2
            E_0=computeE_0(R_I,Z_I,P_p,N_i,Cpp,r_x,r_y,r_z,N,dr,CGF_He,
             ↪  CGF_H,G_u,G_v,G_w,delta_T,E_self,E_II,L)
            H_c=H+(1/2)*(E_0-H)
            U=np.real(propagator('EM',H_c,[],dt))
            C_new=np.dot(U,C)
            P_new=np.array([[0., 0.],[0., 0.]])
            for u in range(0,2) :
                for v in range(0,2) :
                    for p in range(0,1) :
                        P_new[u][v] += C_new[u][p]*C_new[v][p]
                    P_new[u][v] *=2

             ↪  Hdt=computeE_0(R_I,Z_I,P_new,N_i,Cpp,r_x,r_y,r_z,N,dr,CGF_He,
             ↪  CGF_H,G_u,G_v,G_w,delta_T,E_self,E_II,L)
            U=np.real(propagator(select,H,Hdt,dt))
            C_new=np.dot(U,C)
            P_new=np.array([[0., 0.],[0., 0.]])
            for u in range(0,2) :
                for v in range(0,2) :
                    for p in range(0,1) :
                        P_new[u][v] += C_new[u][p]*C_new[v][p]
                    P_new[u][v] *=2
            et=time.time()
        case 'AETRS':
            st=time.time()
            if i<5:
                Hdt=LagrangeExtrapolate(t[0:i],energies[0:i],tnew)
            else:
                Hdt=LagrangeExtrapolate(t[i-5:i],energies[i-5:i],tnew)
            U=np.real(propagator(select,H,Hdt,dt))
            C_new=np.dot(U,C)
            P_new=np.array([[0., 0.],[0., 0.]])
            for u in range(0,2) :
                for v in range(0,2) :
                    for p in range(0,1) :
                        P_new[u][v] += C_new[u][p]*C_new[v][p]
                    P_new[u][v] *=2
            et=time.time()
        case 'CAETRS':
            st=time.time()
            if i<5:
                Hdt=LagrangeExtrapolate(t[0:i],energies[0:i],tnew)
            else:
                Hdt=LagrangeExtrapolate(t[i-5:i],energies[i-5:i],tnew)
            U=np.real(propagator(select,H,Hdt,dt))
            C_p=np.dot(U,C)
            P_p=np.array([[0., 0.],[0., 0.]])
            for u in range(0,2) :
                for v in range(0,2) :
                    for p in range(0,1) :
                        P_p[u][v] += C_p[u][p]*C_p[v][p]
                    P_p[u][v] *=2
            E_0=computeE_0(R_I,Z_I,P_p,N_i,Cpp,r_x,r_y,r_z,N,dr,CGF_He,
             ↪  CGF_H,G_u,G_v,G_w,delta_T,E_self,E_II,L)
            H_c=H+(1/2)*(E_0-H)
            Hdt=2*H_c-Hprev
            U=np.real(propagator(select,H_c,Hdt,dt))
            C_new=np.dot(U,C)
            P_new=np.array([[0., 0.],[0., 0.]])
            for u in range(0,2) :
                for v in range(0,2) :
                    for p in range(0,1) :
                        P_new[u][v] += C_new[u][p]*C_new[v][p]
                    P_new[u][v] *=2
            et=time.time()
        case 'CFM4':
            st=time.time()
            if i<5:
                Ht1=LagrangeExtrapolate(t[0:i],energies[0:i],
                 ↪  (t[i-1])+((1/2)-(np.sqrt(3)/6)))
                Ht2=LagrangeExtrapolate(t[0:i],energies[0:i],
                 ↪  (t[i-1])+((1/2)+(np.sqrt(3)/6)))
            else:
                Ht1=LagrangeExtrapolate(t[i-5:i],energies[i-5:i],
                 ↪  (t[i-1])+((1/2)-(np.sqrt(3)/6)))
                Ht2=LagrangeExtrapolate(t[i-5:i],energies[i-5:i],
                 ↪  (t[i-1])+((1/2)+(np.sqrt(3)/6)))

            U=np.real(propagator(select,Ht1,Ht2,dt))
            C_new=np.dot(U,C)
            P_new=np.array([[0., 0.],[0., 0.]])
            for u in range(0,2) :
                for v in range(0,2) :
                    for p in range(0,1) :
                        P_new[u][v] += C_new[u][p]*C_new[v][p]
                    P_new[u][v] *=2
            et=time.time()
        case _:
            raise TypeError("Invalid propagator")
    run=et-st
    return P_new,run

def transition_dipole_tensor_calculation(r_x,r_y,r_z,CGF_He,CGF_H,dr)
 ↪  :

    D_x = [[0.,0.],[0.,0.]]
    for i in range(0,len(r_x)) :
        for j in range(0,len(r_x)) :
            for k in range(0,len(r_x)) :

                #Integrate over grid points
                D_x[0][0] += r_x[i]*CGF_He[i][j][k]**2*dr
                D_x[0][1] += r_x[i]*CGF_He[i][j][k]*CGF_H[i][j][k]*dr
                D_x[1][0] += r_x[i]*CGF_H[i][j][k]*CGF_He[i][j][k]*dr
                D_x[1][1] += r_x[i]*CGF_H[i][j][k]**2*dr

    D_y = [[0.,0.],[0.,0.]]
    for i in range(0,len(r_y)) :
        for j in range(0,len(r_y)) :
            for k in range(0,len(r_y)) :

                #Integrate over grid points
                D_y[0][0] += r_y[i]*CGF_He[i][j][k]**2*dr
                D_y[0][1] += r_y[i]*CGF_He[i][j][k]*CGF_H[i][j][k]*dr
                D_y[1][0] += r_y[i]*CGF_H[i][j][k]*CGF_He[i][j][k]*dr
                D_y[1][1] += r_y[i]*CGF_H[i][j][k]**2*dr

    D_z = [[0.,0.],[0.,0.]]
    for i in range(0,len(r_z)) :
        for j in range(0,len(r_z)) :
            for k in range(0,len(r_z)) :

                #Integrate over grid points
                D_z[0][0] += r_z[i]*CGF_He[i][j][k]**2*dr
                D_z[0][1] += r_z[i]*CGF_He[i][j][k]*CGF_H[i][j][k]*dr
                D_z[1][0] += r_z[i]*CGF_H[i][j][k]*CGF_He[i][j][k]*dr
                D_z[1][1] += r_z[i]*CGF_H[i][j][k]**2*dr

    D_tot=D_x+D_y+D_z

    return D_x,D_y,D_z,D_tot

def GetP(KS,S):
    S=Matrix(S)
    U,s = S.diagonalize()
    s = s**(-0.5)
    X = np.matmul(np.array(U,dtype='float64'),
 ↪  np.array(s,dtype='float64'))
    X_dag = np.matrix.transpose(np.array(X,dtype='float64'))
    KS_temp = Matrix(np.matmul(X_dag,np.matmul(KS,X)))
    C_temp, e = KS_temp.diagonalize()
    C = np.matmul(X,C_temp)
    P_new=np.array([[0., 0.],[0., 0.]])
    for u in range(0,2):
        for v in range(0,2):
            for p in range(0,1) :
                P_new[u][v] += C[u][p]*C[v][p]
            P_new[u][v] *=2

    return P_new

def LagrangeExtrapolate(t,H,tnew):
    f=lagrange(t,H)
    return np.real(f(tnew))

def pop_analysis(P,S) :
    PS = np.matmul(P,S)
    pop_total = np.trace(PS)
    pop_He = PS[0,0]
    pop_H = PS[1,1]
    return pop_total, pop_He, pop_H

def ShannonEntropy(P):
    P_eigvals=np.linalg.eigvals(P)
    P_eigvals_corrected=[x for x in P_eigvals if x>0.00001]
    P_eigvecbasis=np.diag(P_eigvals_corrected)
    SE=np.trace(np.dot(P_eigvecbasis,np.log(P_eigvecbasis)))

    return SE

def vonNeumannEntropy(P):
    vNE=np.trace(np.dot(P,np.log(P)))
    return vNE

def EntropyPInitBasis(P,Pini):
    PPinitBasis=np.matmul(np.linalg.inv(Pini),np.matmul(P,Pini))
    EPinitBasis=np.trace(np.dot(np.abs(PPinitBasis),
     ↪  np.log(np.abs(PPinitBasis))))
    return EPinitBasis
```

```
def rttddft(nsteps,dt,propagator,SCFiterations,L,N_i,alpha,Coef,R_I,
↪   Z_I,Cpp,P_init,kickstrength,kickdirection,projectname):

    print(' *******           *******   ********** *******\n'+
    '/**////**  **   **/**////** /////**/// /**////**\n'+
    '/**   /** //** ** /**   /**     /**    /**   /**\n'+
    '/*******   //***  /*******      /**    /******* \n'+
    '/**////     /**   /**///**      /**    /**////  \n'+
    '/**         **    /**  //**     /**    /**      \n'+
    '/**        **     /**   //**    /**    /**      \n'+
    '//        //      //     //     //     //       \n'+
    '-------------------------------------------------------\n')
    print('Contributing authors:\nMatthew Thompson -
    ↪   MatThompson@lincoln.ac.uk\nMatt Watkins -
    ↪   MWatkins@lincoln.ac.uk\n'+
    'Date of last edit: 14/05/2023\n'+
    'Description: A program to perform RT-TDDFT exclusively in
    ↪   Python.\n'+
    '\t     A variety of propagators (CN, EM, ETRS, etc.) can be \n'+
    '\t     selected for benchmarking and testing. Comments on \n'+
    '\t     the approaches used are in progress.\n'+
    '\n'+'System requirements:\n'+
    '- Python >= 3.10\n'+
    '- Numpy >= 1.13\n'+
    '- npy-append-array >= 0.9.5\n'+
    '- Sympy >=1.7.1\n'+
    '- Scipy >= 0.19\n'+
    '- PySCF >=2.0a, requires either Unix-based system or WSL\n'+
    '-------------------------------------------------------------
    ↪   -------\n')

    print('Ground state calculations:\n')
    # Performing calculation of all constant variables to remove the
    ↪   need to repeat it multiple times.
    r_x,r_y,r_z,N,dr,GTOs_He,CGF_He,GTOs_H,CGF_H,G_u,G_v,G_w,PW_He_G,
    ↪   PW_H_G,S,delta_T,E_self,E_II=dftSetup(R_I,alpha,Coef,L,N_i,Z_I)
```

```
# Compute the ground state first
P,H,C,KS=computeDFT(R_I,alpha,Coef,L,N_i,P_init,Z_I,Cpp,
↪  SCFiterations,r_x,r_y,r_z,N,dr,GTOs_He,CGF_He,GTOs_H,CGF_H,G_u,
↪  G_v,G_w,PW_He_G,PW_H_G,S,delta_T,E_self,E_II)
# initialising all variable arrays
Pgs=P
energies=[]
mu=[]
propagationtimes=[]
SE=[]
vNE=[]
EPinit=[]
# Writing a short .txt file giving the simulation parameters to
↪  the project folder
os.mkdir(os.path.join(os.getcwd(),projectname))
lines = ['PyRTP run
↪  parameters','------------------------','Propagator:
↪  '+propagator,'Timesteps: '+str(nsteps),'Timestep:
↪  '+str(dt),'Kick strength: '+str(kickstrength),'Kick direction:
↪  '+str(kickdirection)]
with open(projectname+'/'+projectname+'_parameters.txt', 'w') as
↪  f:
    for line in lines:
        f.write(line)
        f.write('\n')

# time array can be set now
t=np.arange(0,nsteps*dt,dt)

for i in range(0,nsteps):

    ↪  print('---------------------------------------------------------
    ↪  -----------\nPropagation timestep: '+str(i+1))
    #Applying perturbation
    KS=GaussianKick(KS,kickstrength,kickdirection,t[i],r_x,r_y,r_z,
    ↪  CGF_He,CGF_H,dr)
    #Getting perturbed density matrix
    P=GetP(KS,S)
```

```
# Propagating depending on method.
# AETRS, CAETRS and CFM4 require 2 prior timesteps, which use
↪   ETRS
if i<2 and propagator==('AETRS'):
    P,proptime=propagate(R_I,Z_I,P,H,C,dt,'ETRS',N_i,Cpp,r_x,
    ↪   r_y,r_z,N,dr,CGF_He,CGF_H,G_u,G_v,G_w,delta_T,E_self,
    ↪   E_II,L,[],t,energies,t[i],i)
elif i<2 and propagator==('CAETRS'):
    P,proptime=propagate(R_I,Z_I,P,H,C,dt,'ETRS',N_i,Cpp,r_x,
    ↪   r_y,r_z,N,dr,CGF_He,CGF_H,G_u,G_v,G_w,delta_T,E_self,
    ↪   E_II,L,[],t,energies,t[i],i)
elif i<2 and propagator==('CFM4'):
    P,proptime=propagate(R_I,Z_I,P,H,C,dt,'ETRS',N_i,Cpp,r_x,
    ↪   r_y,r_z,N,dr,CGF_He,CGF_H,G_u,G_v,G_w,delta_T,E_self,
    ↪   E_II,L,[],t,energies,t[i],i)
elif i>1 and propagator==('AETRS'):
    P,proptime=propagate(R_I,Z_I,P,H,C,dt,propagator,N_i,Cpp,
    ↪   r_x,r_y,r_z,N,dr,CGF_He,CGF_H,G_u,G_v,G_w,delta_T,
    ↪   E_self,E_II,L,energies[i-1],t,energies,t[i],i)
elif i>1 and propagator==('CAETRS'):
    P,proptime=propagate(R_I,Z_I,P,H,C,dt,propagator,N_i,Cpp,
    ↪   r_x,r_y,r_z,N,dr,CGF_He,CGF_H,G_u,G_v,G_w,delta_T,
    ↪   E_self,E_II,L,energies[i-1],t,energies,t[i],i)
elif i>1 and propagator==('CFM4'):
    P,proptime=propagate(R_I,Z_I,P,H,C,dt,propagator,N_i,Cpp,
    ↪   r_x,r_y,r_z,N,dr,CGF_He,CGF_H,G_u,G_v,G_w,delta_T,
    ↪   E_self,E_II,L,energies[i-1],t,energies,t[i],i)
else:
    P,proptime=propagate(R_I,Z_I,P,H,C,dt,propagator,N_i,Cpp,
    ↪   r_x,r_y,r_z,N,dr,CGF_He,CGF_H,G_u,G_v,G_w,delta_T,
    ↪   E_self,E_II,L,[],t,energies,t[i],i)
print('Propagation time: '+str(proptime))
# Converging on accurate KS and P
P,H,C,KS=computeDFT(R_I,alpha,Coef,L,N_i,P,Z_I,Cpp,
↪   SCFiterations,r_x,r_y,r_z,N,dr,GTOs_He,CGF_He,GTOs_H,
↪   CGF_H,G_u,G_v,G_w,PW_He_G,PW_H_G,S,delta_T,E_self,E_II)
# Information Collection
```

```
D_x,D_y,D_z,D_tot=transition_dipole_tensor_calculation(r_x,r_y,
↪  r_z,CGF_He,CGF_H,dr)
mu_t=np.trace(np.dot(D_tot,P))
energies.append(H)
SE.append(ShannonEntropy(P))
vNE.append(vonNeumannEntropy(P))
EPinit.append(EntropyPInitBasis(P,Pgs))
mu.append(mu_t)
propagationtimes.append(proptime)
# Saving data to external files every 10 steps
if  i>0 and (i+1)%10==0:
    with
    ↪  NpyAppendArray(projectname+'/'+projectname+'_energies.npy')
    ↪  as npaa:
        npaa.append(np.array(energies[i-9:i+1]))
    with NpyAppendArray(projectname+'/'+projectname+'_mu.npy')
    ↪  as npaa:
        npaa.append(np.array(mu[i-9:i+1]))
    with
    ↪  NpyAppendArray(projectname+'/'+projectname+'_timings.npy')
    ↪  as npaa:
        npaa.append(np.array(propagationtimes[i-9:i+1]))
    with NpyAppendArray(projectname+'/'+projectname+'_SE.npy')
    ↪  as npaa:
        npaa.append(np.array(SE[i-9:i+1]))
    with NpyAppendArray(projectname+'/'+projectname+'_t.npy')
    ↪  as npaa:
        npaa.append(t[i-9:i+1])
    with
    ↪  NpyAppendArray(projectname+'/'+projectname+'_vNE.npy')
    ↪  as npaa:
        npaa.append(np.array(vNE[i-9:i+1]))
    with
    ↪  NpyAppendArray(projectname+'/'+projectname+'_EPinit.npy')
    ↪  as npaa:
        npaa.append(np.array(EPinit[i-9:i+1]))

# Outputting calculated data
```

```
        print('Total dipole moment: '+str(mu_t))
        print('Shannon entropy: '+str(SE[i]))
        print('von Neumann Entropy: '+str(vNE[i]))
        print('P_init Basis Entropy: '+str(EPinit[i]))

    return t,energies,mu,propagationtimes,SE,vNE,EPinit

#%%
# Simulation parameters
nsteps=20
timestep=0.1
SCFiterations=100
kickstrength=0.1
kickdirection=[1,0,0]
proptype='ETRS'
projectname='ETRSrun'

#Grid parameters
L=10.
N_i=60

# Molecule parameters
alpha = [[0.3136497915, 1.158922999, 6.362421394],[0.1688554040,
↪  0.6239137298, 3.425250914]] #alpha[0] for He, alpha[1] for H
Coef = [0.4446345422, 0.5353281423, 0.1543289673] #Coefficients are
↪  the same for both He and H
R_I = [np.array([0.,0.,0.]), np.array([0.,0.,1.4632])] #R_I[0] for He,
↪  R_I[1] for H.
Z_I = [2.,1.]
Cpp = [[-9.14737128,1.71197792],[-4.19596147,0.73049821]] #He, H
P_init=np.array([[1.333218,0.],[0.,0.666609]])

t,energies,mu,timings,SE,vNE,EPinit=rttddft(nsteps,timestep,proptype,
↪  SCFiterations,L,N_i,alpha,Coef,R_I,Z_I,Cpp,P_init,
↪  kickstrength,kickdirection,projectname)

#%%
# Post-Processing section
```

```
#%%
# Energy plot
plt.plot(t,np.array(energies))
plt.xlabel('Time, $s$')
plt.ylabel('Energy, $Ha$')

#%%
# Dipole moment plot
plt.plot(t,np.array(mu))
plt.xlabel('Time, $s$')
plt.ylabel('Dipole moment, $\mu$')

#%%
# Absorption Spectrum plot
filterpercentage=0
mu=np.array(mu)
c=299792458
h=6.62607015e-34
sp=scipy.fft.rfft(mu)
indexes=np.where(sp<np.percentile(sp,filterpercentage))[0]
sp[indexes]=0
freq = scipy.fft.rfftfreq(mu.size,(0.1*2.4188843265857e-17))
freqshift=scipy.fft.fftshift(freq)
ld=c/freq
en=h*freq
plt.plot(ld,np.abs(sp))
plt.xlabel('Wavelength, $\lambda$')
plt.ylabel('Intensity')
#plt.xlim([0, 5e-7])
```

A.6 A discussion on LR-TDDFT vs RT-TDDFT

TDDFT has two main methods within it: linear response TDDFT and real-time TDDFT, both with benefits and downsides. In order to understand which method is most suitable for a given system, it is worthwhile to discuss how linear-response TDDFT works.

The term ‘linear-response’ in LR-TDDFT comes from the fact that the applied perturbation is sufficiently small enough that the change in electron density is proportional to the perturbation strength. From this, the perturbed electron density can be written as a first order Taylor series, and thus, we may use the following equation:

$$\rho_1(\boldsymbol{r},t) = \iint \chi_{KS}(\boldsymbol{r},t,\boldsymbol{r}',t')\, v_{KS,1}(\boldsymbol{r}',t)\, \mathrm{d}^3\boldsymbol{r}'\, \mathrm{d}t' \tag{109}$$

Such that the effective potential $v_{KS,1}$ is given by the following equation:

$$v_{KS,1}(\boldsymbol{r}',t) = v_1(\boldsymbol{r},t) + \int \frac{\rho_1(\boldsymbol{r}',t)}{|\boldsymbol{r}-\boldsymbol{r}'|}\mathrm{d}^3\boldsymbol{r}' + \iint f_x[\rho_0](\boldsymbol{r},t,\boldsymbol{r}',t')\rho_1(\boldsymbol{r},t)\mathrm{d}^3\boldsymbol{r}'\mathrm{d}t' \tag{110}$$

This method is generally significantly more computationally efficient, due to not needing to converge upon the true electron density as in RT-TDDFT. LR-TDDFT, however, suffers from the limitation of the perturbation needing to be sufficiently small so that the system remains in a linear regime. For this reason, RT-TDDFT was chosen as the basis of this project, given its adaptability to many systems, alongside its ability to compute time-dependent phenomena, such as dipole moments, at the cost of additional computational time (in most cases). It is worth noting that while this project uses RT-TDDFT as the preferred method, LR-TDDFT has very useful applications, however if computational cost is not a limiting factor, it is *usually* more beneficial to use RT-TDDFT, given the more robust platform to process information.

A.7 A derivation of the kinetic energy derivative

The kinetic energy of a system, expressed in terms of the Kohn-Sham orbitals is given by the following equation [35]:

$$\hat{T} = \langle \phi^*(\boldsymbol{r}) \mid -\frac{1}{2}\nabla^2 \mid \phi(\boldsymbol{r})\rangle \tag{111}$$

Which, as written in terms of the approximated Slater-type orbitals, is equal to the following:

$$\hat{T} = \sum_{\mu\nu} P^{\mu\nu} \langle \phi_\mu^{STO-3G}(\boldsymbol{r}) \mid -\frac{1}{2}\nabla^2 \mid \phi_\nu^{STO-3G}(\boldsymbol{r})\rangle \tag{112}$$

Thus, we may take the derivative of $\hat{T}$ with respect to $\boldsymbol{P}$ to be:

$$\frac{\delta \hat{T}}{\delta \boldsymbol{P}} = \langle \phi_\mu^{STO-3G}(\boldsymbol{r}) \mid -\frac{1}{2}\nabla^2 \mid \phi_\nu^{STO-3G}(\boldsymbol{r})\rangle \tag{113}$$

This allows for an analytical solution of $\frac{\delta \hat{T}}{\delta \boldsymbol{P}}$, however it is significantly more simple to use a plane wave, as the kinetic energy of any plane wave is equivalent to $\frac{\boldsymbol{G}^2}{2}$ [36]. To get to this, we may write the kinetic energy derivative using a substitution of the plane wave expansion:

$$\frac{\delta T_{\mu\nu}}{\delta P_{\mu\nu}} = \frac{1}{2N^2}\int d\boldsymbol{r} \sum_{\boldsymbol{G}} \phi_\mu^{*\,PW}(\boldsymbol{G}) \exp[i\boldsymbol{G}\cdot\boldsymbol{r}] \sum_{\boldsymbol{G}'} \boldsymbol{G}'^2 \phi_\nu^{PW}(\boldsymbol{G}') \exp[-i\boldsymbol{G}'\cdot\boldsymbol{r}] \tag{114}$$

Plane waves are orthogonal, given the earlier description of them, and thus, the above equation is equal to zero at all values of $\boldsymbol{G}$ besides $\boldsymbol{G} = \boldsymbol{G}'$. Thus, we find the final kinetic energy derivative in terms of the reciprocal space vector $\boldsymbol{G}$ to be equal to:

$$\frac{\delta T_{\mu\nu}}{\delta P_{\mu\nu}} = \frac{1}{2N^2} \int \mathrm{d}\boldsymbol{r} \sum_{\boldsymbol{G}} \boldsymbol{G}^2 \, \phi_{\mu}^{*\,PW}(\boldsymbol{G}) \, \phi_{\nu}^{PW}(\boldsymbol{G}) \tag{115}$$

For the purposes of integrating on the grid, as is done in PyRTP and CP2K, the integral $\int \mathrm{d}\boldsymbol{r}$ is equal to the volume of the grid, which is L^3. Thus, the kinetic energy derivative used in PyRTP is given as:

$$\frac{\delta T_{\mu\nu}}{\delta P_{\mu\nu}} = \frac{L^3}{2N^2} \sum_{\boldsymbol{G}} \boldsymbol{G}^2 \, \phi_{\mu}^{*\,PW}(\boldsymbol{G}) \, \phi_{\nu}^{PW}(\boldsymbol{G}) \tag{116}$$

A.8 A derivation of the Hartree potential

The Hartree potential is given as the electrostatic potential from the electron density, and thus may be described by the following equation [35]:

$$\hat{V}_H(\boldsymbol{r}) = \int \mathrm{d}\boldsymbol{r}' \frac{\rho_{gs}(\boldsymbol{r})}{|\boldsymbol{r} - \boldsymbol{r}'|} \tag{117}$$

Where ρ_{gs} is the ground-state electron density. This potential is subject to a continuity equation in the form of the Poisson equation, given as the following:

$$\nabla^2 \hat{V}_H(\boldsymbol{r}) = -4\pi \rho_{gs}(\boldsymbol{r}) \tag{118}$$

This, in a similar fashion to the kinetic energy, can be solved analytically, however to minimise computational cost, the plane wave substitution can again be used:

$$\nabla^2 \sum_{\boldsymbol{G}} V_H(\boldsymbol{G}) \exp[i\boldsymbol{G} \cdot \boldsymbol{r}] = -4\pi \sum_{\boldsymbol{G}'} \rho(\boldsymbol{G}') \exp[i\boldsymbol{G}' \cdot \boldsymbol{r}] \tag{119}$$

We may then take the derivative on the left hand side and use the same condition to cancel the exponentials (the equation is only non-zero when $\boldsymbol{G} = \boldsymbol{G}'$), giving the following equation when rearranged for $\hat{V}_H(\boldsymbol{G})$:

$$V_H(\boldsymbol{G}) = 4\pi \frac{\rho(\boldsymbol{G})}{\boldsymbol{G}^2} \tag{120}$$

A.9 Lagrange Interpolation method

The Lagrange interpolation method is a process to determine an nth-order polynomial which connects a set of $n + 1$ nodes [37]. The approximated polynomial $P(x)$ is given by the following equation:

$$P(x) = \sum_{j=0}^{n} \left\{ y_j \prod_{k=0, k \neq j} \frac{x - x_k}{x_j - x_k} \right\} \tag{121}$$

Such that $y_i = f(x_i)$. Given a collection of points $\{x_i, y_i\}$, we may use this to approximate any point within the set of points accurately, but also outside the domain of the points with acceptable accuracy, so long as the point lies close to the domain of the points. However, it is worth noting that this approximation works best with minimal points, as it becomes rapidly unstable beyond around 10-15 points. Many Python packages which perform this operation (such as SciPy) include guidance to not use more than 20 points, and thus, for the purposes of using the '*scipy.interpolate.lagrange*' function in PyRTP, the input arrays were restricted to contain no more than 5 points.

A.10 Absorption spectra calculation source code

```python
#%%
# Title: Absorption spectrum calculator
# Author: Matt Thompson (MatThompson@lincoln.ac.uk)
# Date: 1/3/23
# Matt Thompson (2023), all rights reserved

import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq, fftshift, rfft, rfftfreq
import re

# This section was contributed and adapted from Christopher Dicken's
↪  code of the same purpose
pattern = "X="

data = []

file = open("muData/HeH+ETRS-dipole-moments.dat", "r")
for line in file:
    print(line)
    if re.search(pattern, line):
        data += line.replace(' ', '').replace('X', '').replace('Y',
         ↪  '').replace('Z', '').replace('Total', '').replace('i',
         ↪  '').replace('E', '')

data = ''.join(data)
data = data.split('\n')
data = ''.join(data)
data = data.split('=')

del data[0]

#x = []
#y = []
#z = []
tot = []

for i in range(int(len(data)/4)):
    #x.append(float(data[i * 4]))
    #y.append(float(data[i * 4 + 1]))
    #z.append(float(data[i * 4 + 2]))
    tot.append(float(data[i * 4 + 3]))

moments=np.array(tot)
t=np.linspace(0.,moments.size*0.1,moments.size)*2.4188843265857e-17
plt.plot(t,moments)
plt.xlabel('Time, $s$')
plt.ylabel('Dipole moment, $\mu$')
#%%
c=299792458
h=6.62607015e-34
sp=rfft(moments)
freq = rfftfreq(moments.size,(0.1*2.4188843265857e-17))
freqshift=fftshift(freq)
ld=c/freq
en=h*freq
plt.plot(ld,np.abs(sp))
plt.xlabel('Wavelength, $\lambda$')
plt.ylabel('Intensity')
plt.xlim([0, 5e-7])

# %%
```

A.11 cutoff_inputs.sh

As a brief overview of what each script does:

- cutoff_inputs.sh - creates a folder for each cutoff value, containing the necessary files to run the DFT calculation in CP2K.
- cutoff_run.sh - uses MPI to run the programs simultaneously, up to the number of threads available.
- cutoff_analyze.sh - reads the output files of each run to collect the energies and grid distribution information and then formats the data into a usable table.

```bash
#!/bin/bash

cutoffs="50 100 150 200 250 300 350 400 450 500"

basis_file=BASIS_SET
potential_file=GTH_POTENTIALS
template_file=template.inp
input_file=HeH+-cutofftest.inp


for ii in $cutoffs ; do
    work_dir=cutoff_${ii}Ry
    if [ ! -d $work_dir ] ; then
        mkdir $work_dir
    else
        rm -r $work_dir/*
    fi
    sed -e "s/LT_cutoff/${ii}/g" \
        $template_file > $work_dir/$input_file
    cp $basis_file $work_dir
    cp $potential_file $work_dir
done
```

A.12 cutoff_run.sh

```bash
#!/bin/bash

cutoffs="50 100 150 200 250 300 350 400 450 500"

cp2k_bin=cp2k.psmp
input_file=HeH+-cutofftest.inp
output_file=HeH+-cutofftest.out
no_proc_per_calc=1
no_proc_to_use=8

counter=1
max_parallel_calcs=$(expr $no_proc_to_use / $no_proc_per_calc)
for ii in $cutoffs ; do
    work_dir=cutoff_${ii}Ry
    cd $work_dir
    if [ -f $output_file ] ; then
        rm $output_file
    fi
    mpirun -np $no_proc_per_calc $cp2k_bin -o $output_file $input_file
 ↪  &
    cd ..
    mod_test=$(echo "$counter % $max_parallel_calcs" | bc)
    if [ $mod_test -eq 0 ] ; then
        wait
    fi
    counter=$(expr $counter + 1)
done
wait
```

A.13 cutoff_analyse.sh

```bash
#!/bin/bash

cutoffs="50 100 150 200 250 300 350 400 450 500"

input_file=HeH+-cutofftest.inp
output_file=HeH+-cutofftest.out
plot_file=cutoff_data.csv


echo "# Grid cutoff vs total energy" > $plot_file
echo "# Date: $(date)" >> $plot_file
echo "# PWD: $PWD" >> $plot_file
echo -n "# Cutoff (Ry) | Total Energy (Ha)" >> $plot_file
grid_header=true
for ii in $cutoffs ; do
    work_dir=cutoff_${ii}Ry
    total_energy=$(grep -e '^[ \t]*Total energy'
↪   $work_dir/$output_file | awk '{print $3}')
    ngrids=$(grep -e '^[ \t]*QS| Number of grid levels:'
↪   $work_dir/$output_file | \
             awk '{print $6}')
    if $grid_header ; then
        for ((igrid=1; igrid <= ngrids; igrid++)) ; do
            printf " | NG on grid %d" $igrid >> $plot_file
        done
        printf "\n" >> $plot_file
        grid_header=false
    fi
    printf "%10.2f  %15.10f" $ii $total_energy >> $plot_file
    for ((igrid=1; igrid <= ngrids; igrid++)) ; do
        grid=$(grep -e '^[ \t]*count for grid' $work_dir/$output_file
↪   | \
               awk -v igrid=$igrid '(NR == igrid){print $5}')
        printf "  %6d" $grid >> $plot_file
    done
    printf "\n" >> $plot_file
done
```

A.14 Runtime plots

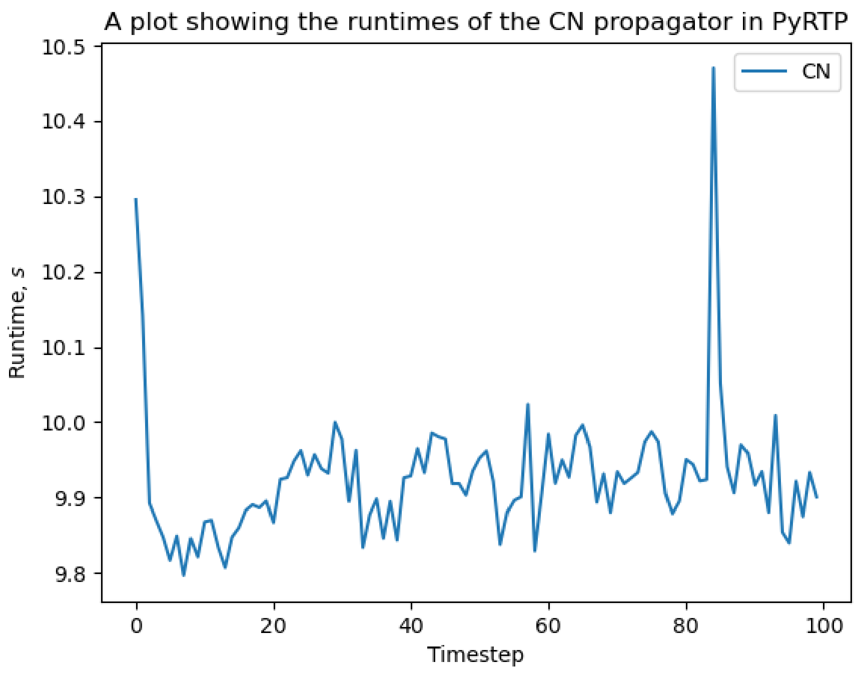


Figure 23: A plot showing the runtime of the Crank-Nicholson propagator against the timestep in the PyRTP program.

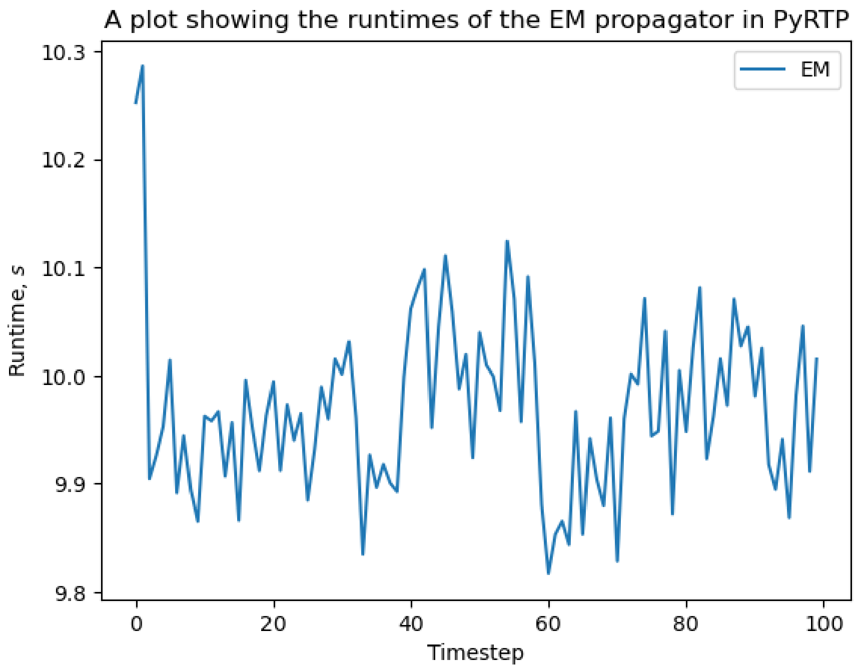


Figure 24: A plot showing the runtime of the Exponential Midpoint propagator against the timestep in the PyRTP program.

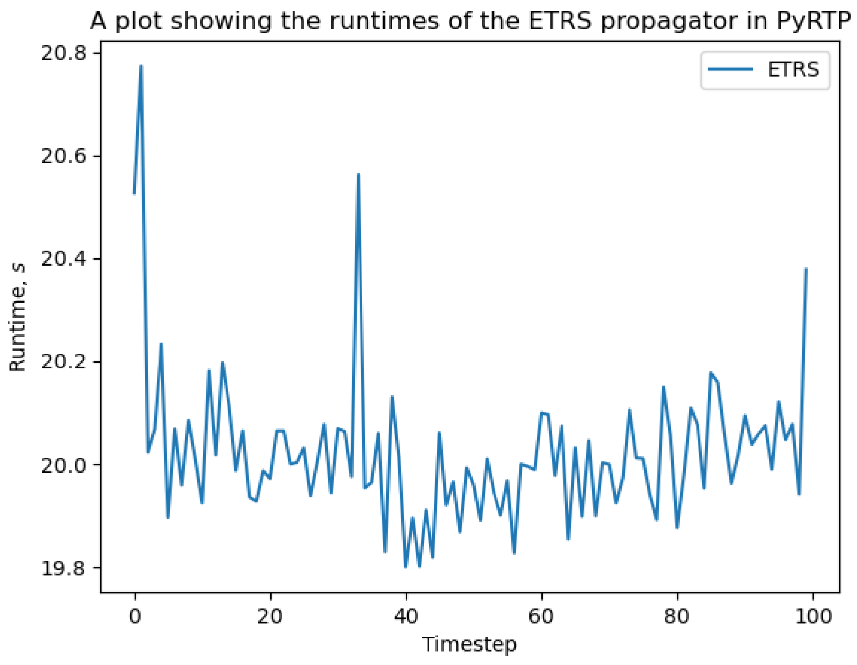


Figure 25: A plot showing the runtime of the ETRS propagator against the timestep in the PyRTP program.

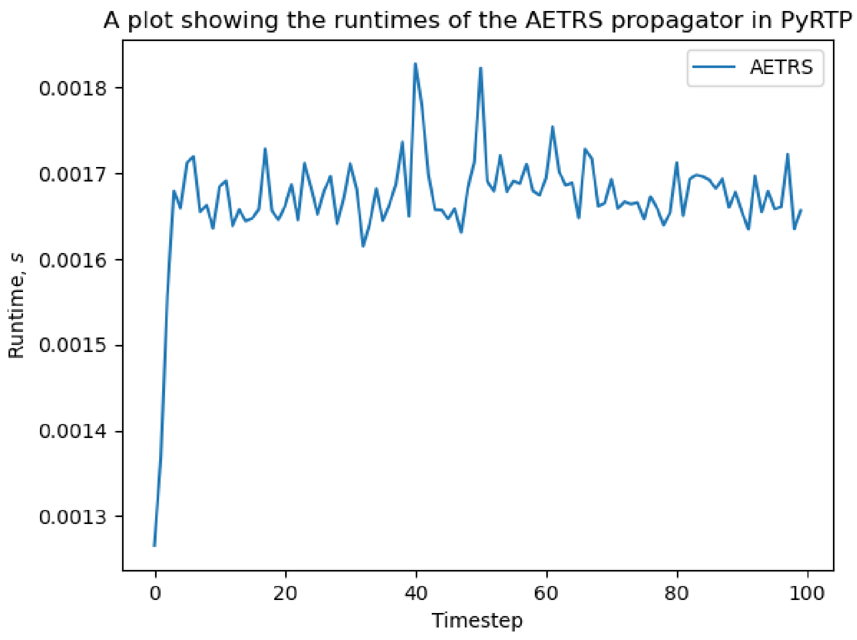


Figure 26: A plot showing the runtime of the AETRS propagator against the timestep in the PyRTP program.

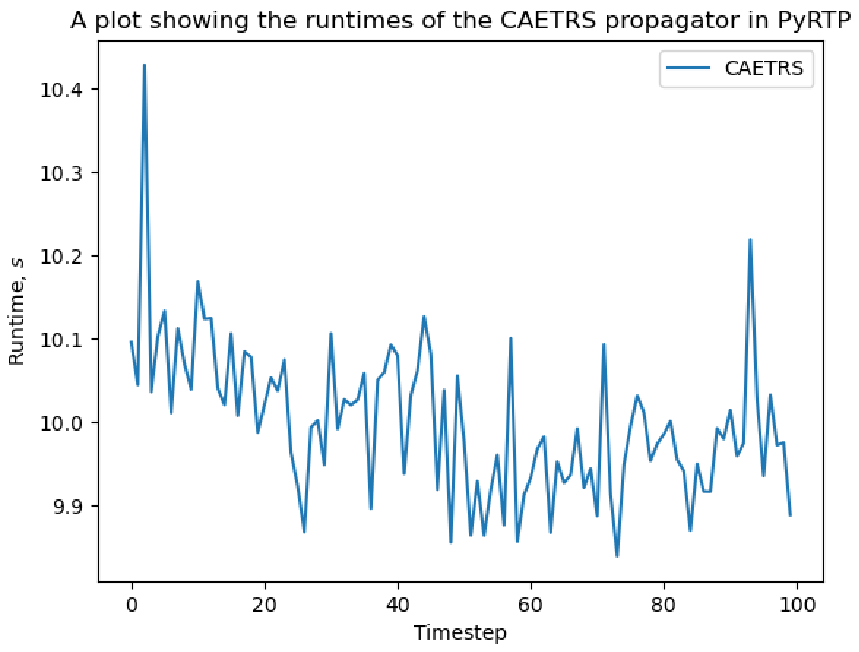


Figure 27: A plot showing the runtime of the CAETRS propagator against the timestep in the PyRTP program.

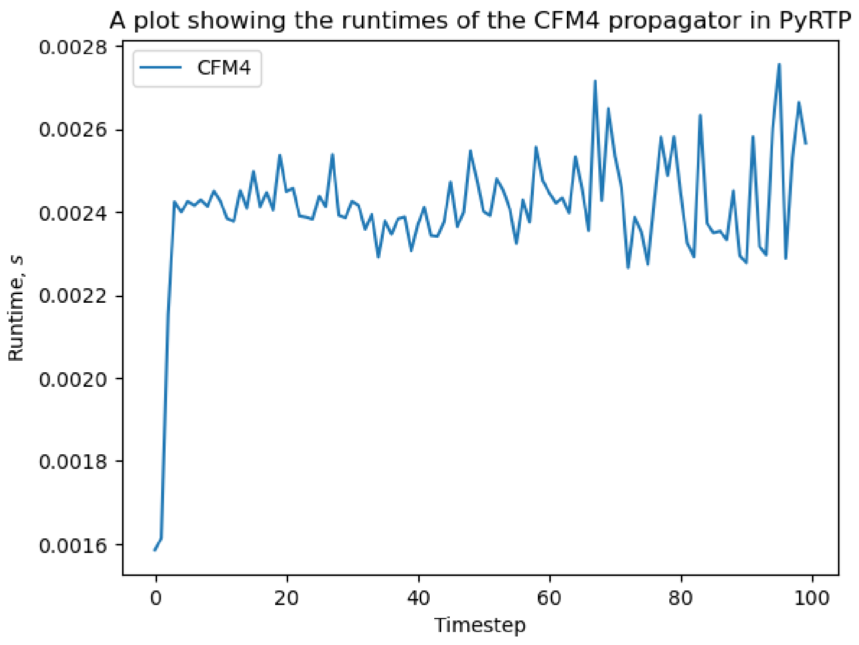


Figure 28: A plot showing the runtime of the Commutator-Free 4th order Magnus propagator against the timestep in the PyRTP program.

A.15 CP2K LR-TDDFT code

```
&GLOBAL
  PROJECT HeH+LR
  RUN_TYPE  energy
  PRINT_LEVEL LOW
&END GLOBAL
&FORCE_EVAL
  METHOD Quickstep
  &PROPERTIES
    &TDDFPT
      NSTATES 3
      MAX_ITER  10
      CONVERGENCE [eV] 1.0e-3
    &END TDDFPT
  &END PROPERTIES
  &DFT
    CHARGE +1
    LSD
    BASIS_SET_FILE_NAME BASIS_MOLOPT_UZH
    POTENTIAL_FILE_NAME POTENTIAL
    !RESTART_FILE_NAME HeH+_rtp-RESTART.rtpwfn
    &MGRID
      CUTOFF 350
    &END MGRID
    &QS
    &END QS
    &SCF
      MAX_SCF 20
      SCF_GUESS ATOMIC
    &END SCF
    &XC
      &XC_FUNCTIONAL PADE
      &END XC_FUNCTIONAL
    &END XC
  &END DFT
  &SUBSYS
    &CELL
      ABC 7 7 7
    &END CELL
    &COORD
    He   0.000000    0.000000   0.000000 HeH+
    H   0.000000   0.000000    1.463200 HeH+
    &END COORD
    &KIND H
      BASIS_SET TZV2P-MOLOPT-GGA-GTH-q1
      POTENTIAL GTH-PADE-q1
    &END KIND
    &KIND He
      BASIS_SET TZV2P-MOLOPT-GGA-GTH-q2
      POTENTIAL GTH-PADE-q2
    &END KIND
  &END SUBSYS
&END FORCE_EVAL
```

A.16 Research Proposal

The following pages contain the research proposal submitted for this project:

UNIVERSITY OF LINCOLN

SCHOOL OF MATHEMATICS AND PHYSICS

Optimisation of Time-Dependant Density Functional Theory methods for quantum materials modelling in CP2K

Research Proposal

Matthew Thompson
THO19699057

Supervised by Professor Matt Watkins

22nd February 2023

Introduction

Atomistic simulation has been used for many years now to both predict theoretical phenomena, and arguably more importantly, verify experimental data. As such, there is a great need by the experimental physics and chemistry communities for accurate programs and packages for use in research performed by these communities. CP2K is one of many quantum chemistry/solid state physics program used around the world for simulations of various atomic-scale systems, using a variety of methods. It is favoured by many for its open-source, community-developed code and its applicability to a wide variety of systems, given the large array of methods featured in the program. One such class of method featured in CP2K is 'Time-Dependant Density Functional Theory', first implemented into the program by Florian Schiffmann as part of his PhD thesis at the University of Zurich in 2010 [1].

This method differs from conventional Density Functional Theory (DFT) calculations, as while conventional DFT calculations provide information on the ground state of an atomistic system, very little (if anything) can be learned about excited states [2]. By comparison, by using time-dependent density functional theory (TDDFT), we may gather information both about excited states of an atomistic system, but also on time-dependant phenomena affecting the system, such as the influence of an external electric or magnetic field, among a variety of other things [3]. The implementation of TDDFT in CP2K currently allows for a range of information to be acquired, including emission and absorption spectra. However, further information may be obtained during the TDDFT calculation method, which presently, are not obtainable in CP2K. The collection of further information during the process of quantum materials simulation would be useful in a wide variety of research, and as CP2K is one of the most popular molecular modelling programs in use today, any improvements to the program would benefit research around the world.

The primary objective of this project is to develop a computational method to produce the greatest amount of useful information from all Time-Dependant Density Functional Theory calculations in CP2K, with the secondary objective being to analyse and potentially optimise the 'real-time propagation' code in CP2K with respect to computational efficiency.

Motivations

The primary motivation for this project aligns heavily with the motivations for developing the CP2K program, which is to develop a comprehensive and accurate method to perform atomistic simulations of a variety of systems, in order to compare them with experimental data. More specifically, in the scope of this project, the motivation is to provide a method to acquire further useful information about quantum materials and how they evolve during the process of simulation in CP2K. The most obvious application of this would be to spintronics, however other phases of matter may also benefit from this additional information. A secondary motivation of this project is to heavily inspect the code, in order to determine if any improvements, either efficiency or accuracy, can be made, for the benefit of all users of CP2K. Lastly, I wish to further my understanding of time-dependant density functional theory methods for my own interest, with the view of continuing my research into this area in the future.

Previous Work

The content of and skills required in this project overlap heavily with several modules from my degree, most notably Molecular Modelling, Nano-Physics and Quantum Mechanics. I have a very solid knowledge of the mechanisms which underly density functional theory, as well as various computational methods, and thus I feel well equipped to undertake this project. I am confident with the *bash* terminal in both GNU/Linux and MacOS, and thus am confident in using CP2K. Finally, I am fast to learn new programming languages, and thus while I am currently unfamiliar with the syntax of the *Fortran* programming language, I am confident in my ability to learn this in due time.

Literature Survey

As previously stated, time-dependant density functional theory methods were implemented into CP2K by Florian Schiffmann in his PhD thesis entitled ‘An atomistic picture of the active interface in dye sensitized solar cells’ [1]. As such, this paper gives the most extensive summary of the theory and the current computational methods used by the ‘real-time propagation’ section of the CP2K source code to perform TDDFT calculations. The documentation for CP2K also cites two papers: one by Kunert and Schmidt [4], and the other by Andermatt, Cha, Schiffmann and VandeVondele [5]. The latter of these papers is referenced in the source code, representing a change to the method used. Together, these three papers give the best summary of the current implementation of TDDFT in CP2K, however these papers are not specific to TDDFT methods in CP2K. The Marques and Gross paper [2], alongside the Runge and Gross paper [3], while they do not discuss CP2K, give very clear overviews of the TDDFT method, with the Marques paper covering much of the Runge paper in more detail. It is clear that, presently, there exists no such paper exclusively on TDDFT methods in CP2K. There are several papers that have been written in recent years on using the CP2K program for theoretical physics and chemistry research. Some notable examples of this are “Spin-Orbit Couplings for Nonadiabatic Molecular Dynamics at the ΔSCF Level”, published in 2022 by Mališ, Vandaele and Luber [6], and “Structure and Mechanisms of Formation of Point defects in HfO2, MgO and hexagonal boron nitride”, written by Jack Strand as part of his PhD study [7]. These papers are useful for looking at potential applications of information gained during TDDFT calculations.

Equipment and Facilities

This project is largely theoretical and computational, and thus access to CP2K, preferably installed onto a GNU/Linux distribution, is required. The costs associated with this are strictly electricity costs, as all equipment is provided either personally or by the University. This cost is negligible, given the small computational cost of calculations for simple systems in CP2K. For more complex systems which are more computationally expensive, access and use of the Young High Performance Computing Hub is provided by the University of Lincoln through existing funding, and thus incurs no cost for the purposes of the project.

Ethics Application

An ethics application has been submitted and approved for this project, with reference “2023_13849”.

Meeting Schedule

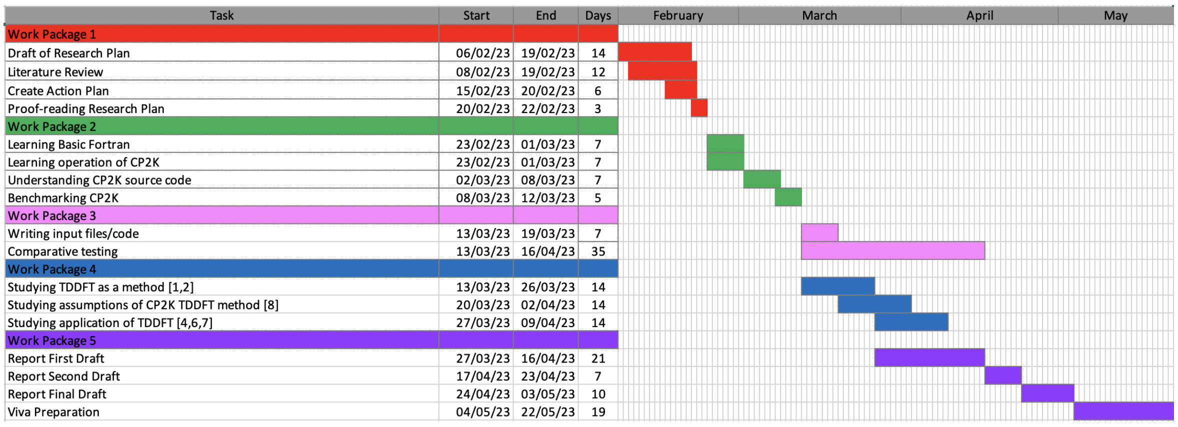
Regular meetings have been organised with my supervisor, Professor Matt Watkins, every Tuesday at 10:30 for 20 minutes. The purpose of these meetings is to discuss the progress of the project and to receive advice on how I may proceed efficiently. I have been advised by Professor Watkins that it is not necessary to book meeting spaces for this session, as we may meet in his office.

Action Plan

In order to complete this research, I have divided the work into 5 work packages:

- Work Package 1: Planning the project
- Work Package 2: Learning the use of CP2K and required programming
- Work Package 3: Programming and comparative testing
- Work Package 4: Theoretical study
- Work Package 5: Report writing

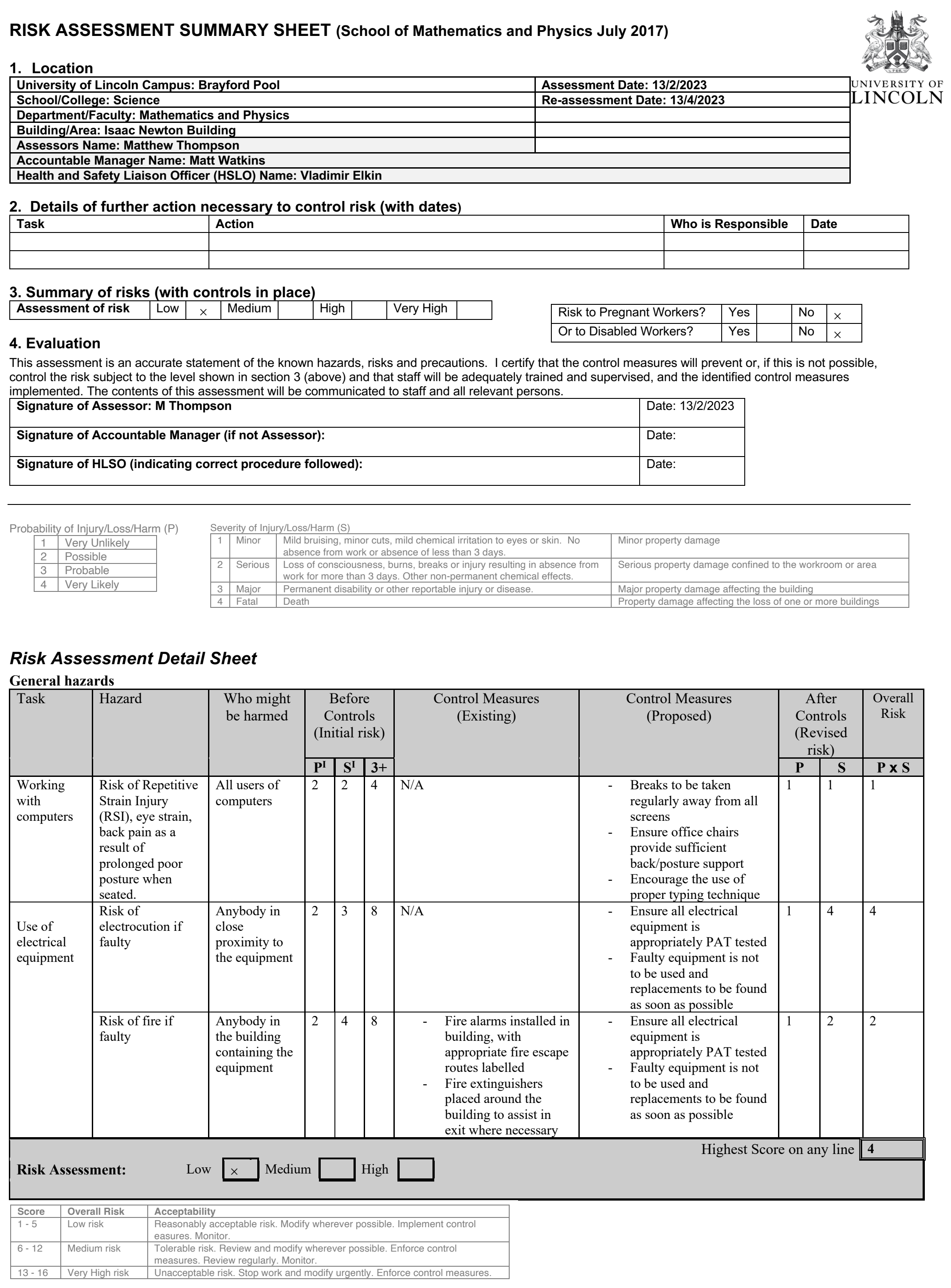
In order to complete the theoretical portion of this project, I will use the Schiffmann thesis [1] and the Marques and Gross paper [2] to study the TDDFT method in CP2K, the Cheng et al. paper entitled "Molecular properties in the Tamm-Dancoff approximation: indirect nuclear spin-spin coupling constants" to study the assumptions of the TDDFT method in CP2K [8] and the Kunert and Schmidt paper [4], the Mališ et al. paper [6] and the Strand PhD thesis [7] to study the application of TDDFT. Much of the use of CP2K will be learned from the CP2K documentation [9] and the "Modern Fortran Explained" book by Michael Metcalf [10]. I intend to begin writing the report early, in order to give sufficient time to proof-read and review the work completed.



Task	Start	End	Days
Work Package 1			
Draft of Research Plan	06/02/23	19/02/23	14
Literature Review	08/02/23	19/02/23	12
Create Action Plan	15/02/23	20/02/23	6
Proof-reading Research Plan	20/02/23	22/02/23	3
Work Package 2			
Learning Basic Fortran	23/02/23	01/03/23	7
Learning operation of CP2K	23/02/23	01/03/23	7
Understanding CP2K source code	02/03/23	08/03/23	7
Benchmarking CP2K	08/03/23	12/03/23	5
Work Package 3			
Writing input files/code	13/03/23	19/03/23	7
Comparative testing	13/03/23	16/04/23	35
Work Package 4			
Studying TDDFT as a method [1,2]	13/03/23	26/03/23	14
Studying assumptions of CP2K TDDFT method [8]	20/03/23	02/04/23	14
Studying application of TDDFT [4,6,7]	27/03/23	09/04/23	14
Work Package 5			
Report First Draft	27/03/23	16/04/23	21
Report Second Draft	17/04/23	23/04/23	7
Report Final Draft	24/04/23	03/05/23	10
Viva Preparation	04/05/23	22/05/23	19

Figure 1: A Gantt chart showing the time required for each task in the project, alongside the start and end dates for each task. Different work packages are illustrated in different colours for visual clarity.

Risk Assessment



RISK ASSESSMENT SUMMARY SHEET **(School of Mathematics and Physics July 2017)**

UNIVERSITY OF LINCOLN

1. Location

University of Lincoln Campus: Brayford Pool	**Assessment Date: 13/2/2023**
School/College: Science	**Re-assessment Date: 13/4/2023**
Department/Faculty: Mathematics and Physics	
Building/Area: Isaac Newton Building	
Assessors Name: Matthew Thompson	
Accountable Manager Name: Matt Watkins	
Health and Safety Liaison Officer (HSLO) Name: Vladimir Elkin	

2. Details of further action necessary to control risk (with dates)

Task	**Action**	**Who is Responsible**	**Date**

3. Summary of risks (with controls in place)

Assessment of risk	Low	×	Medium		High		Very High	

Risk to Pregnant Workers?	Yes		No	×
Or to Disabled Workers?	Yes		No	×

4. Evaluation

This assessment is an accurate statement of the known hazards, risks and precautions. I certify that the control measures will prevent or, if this is not possible, control the risk subject to the level shown in section 3 (above) and that staff will be adequately trained and supervised, and the identified control measures implemented. The contents of this assessment will be communicated to staff and all relevant persons.

Signature of Assessor: M Thompson	Date: 13/2/2023
Signature of Accountable Manager (if not Assessor):	Date:
Signature of HLSO (indicating correct procedure followed):	Date:

Probability of Injury/Loss/Harm (P)

1	Very Unlikely
2	Possible
3	Probable
4	Very Likely

Severity of Injury/Loss/Harm (S)

1	Minor	Mild bruising, minor cuts, mild chemical irritation to eyes or skin. No absence from work or absence of less than 3 days.	Minor property damage
2	Serious	Loss of consciousness, burns, breaks or injury resulting in absence from work for more than 3 days. Other non-permanent chemical effects.	Serious property damage confined to the workroom or area
3	Major	Permanent disability or other reportable injury or disease.	Major property damage affecting the building
4	Fatal	Death	Property damage affecting the loss of one or more buildings

Risk Assessment Detail Sheet

General hazards

Task	Hazard	Who might be harmed	Before Controls (Initial risk)			Control Measures (Existing)	Control Measures (Proposed)	After Controls (Revised risk)		Overall Risk
			P¹	**S¹**	**3+**			**P**	**S**	**P x S**
Working with computers	Risk of Repetitive Strain Injury (RSI), eye strain, back pain as a result of prolonged poor posture when seated.	All users of computers	2	2	4	N/A	- Breaks to be taken regularly away from all screens - Ensure office chairs provide sufficient back/posture support - Encourage the use of proper typing technique	1	1	1
Use of electrical equipment	Risk of electrocution if faulty	Anybody in close proximity to the equipment	2	3	8	N/A	- Ensure all electrical equipment is appropriately PAT tested - Faulty equipment is not to be used and replacements to be found as soon as possible	1	4	4
	Risk of fire if faulty	Anybody in the building containing the equipment	2	4	8	- Fire alarms installed in building, with appropriate fire escape routes labelled - Fire extinguishers placed around the building to assist in exit where necessary	- Ensure all electrical equipment is appropriately PAT tested - Faulty equipment is not to be used and replacements to be found as soon as possible	1	2	2

Highest Score on any line: 4

Risk Assessment: Low [✓ ×] Medium [ ] High [ ]

Score	Overall Risk	Acceptability
1 - 5	Low risk	Reasonably acceptable risk. Modify wherever possible. Implement control easures. Monitor.
6 - 12	Medium risk	Tolerable risk. Review and modify wherever possible. Enforce control measures. Review regularly. Monitor.
13 - 16	Very High risk	Unacceptable risk. Stop work and modify urgently. Enforce control measures.

References

[1] F. Schiffmann, “An atomistic picture of the active interface in dye sensitized solar cells,” Ph.D. dissertation, University of Zurich, Faculty of Science, University of Zurich, Rämistrasse 71, CH-8006 Zürich, Switzerland, 2010.

[2] M. A. L. Marques and E. K. U. Gross, “Time-dependent density functional theory,” *Annu Rev Phys Chem*, vol. 55, pp. 427–455, June 2004.

[3] E. Runge and E. K. U. Gross, “Density-functional theory for time-dependent systems,” *Physical Review Letters*, vol. 52, no. 12, pp. 997–1000, March 1984.

[4] T. Kunert and R. Schmidt, “Non-adiabatic quantum molecular dynamics: General formalism and case study H2+ in strong laser fields,” *The European Physical Journal D - Atomic, Molecular, Optical and Plasma Physics*, vol. 25, no. 1, pp. 15–24, 2003. [Online]. Available: https://doi.org/10.1140/epjd/e2003-00086-8

[5] S. Andermatt, J. Cha, F. Schiffmann, and J. VandeVondele, “Combining Linear-Scaling DFT with Subsystem DFT in Born–Oppenheimer and Ehrenfest Molecular Dynamics Simulations: From Molecules to a Virus in Solution,” *Journal of Chemical Theory and Computation*, vol. 12, no. 7, pp. 3214–3227, 2016, pMID: 27244103. [Online]. Available: https://doi.org/10.1021/acs.jctc.6b00398

[6] M. Mališ, E. Vandaele, and S. Luber, “Spin-Orbit Couplings for Nonadiabatic Molecular Dynamics at the ΔSCF Level,” *Journal of Chemical Theory and Computation*, vol. 18, no. 7, pp. 4082–4094, 2022, pMID: 35666703. [Online]. Available: https://doi.org/10.1021/acs.jctc.1c01046

[7] J. W. Strand, “Structure and Mechanisms of Formation of Point defects in HfO2, MgO and hexagonal boron nitride,” Ph.D. dissertation, Department of Chemistry, University College London, https://discovery.ucl.ac.uk/id/eprint/10076686/1/Main.pdf, June 2019.

[8] C. Y. Cheng, M. S. Ryley, M. J. Peach, D. J. Tozer, T. Helgaker, and A. M. Teale, “Molecular properties in the Tamm-Dancoff approximation: indirect nuclear spin-spin coupling constants,” *Molecular Physics*, vol. 113, no. 13-14, pp. 1937–1951, January 2015.

[9] T. D. Kühne, M. Iannuzzi, M. D. Ben, V. V. Rybkin, P. Seewald, F. Stein, T. Laino, R. Z. Khaliullin, O. Schütt, F. Schiffmann, D. Golze, J. Wilhelm, S. Chulkov, M. H. Bani-Hashemian, V. Weber, U. Borštnik, M. Taillefumier, A. S. Jakobovits, A. Lazzaro, H. Pabst, T. Müller, R. Schade, M. Guidon, S. Andermatt, N. Holmberg, G. K. Schenter, A. Hehn, A. Bussy, F. Belleflamme, G. Tabacchi, A. Glöß, M. Lass, I. Bethune, C. J. Mundy, C. Plessl, M. Watkins, J. VandeVondele, M. Krack, and J. Hutter, “CP2K: An electronic structure and molecular dynamics software package - Quickstep: Efficient and accurate electronic structure calculations,” *The Journal of Chemical Physics*, vol. 152, no. 19, p. 194103, May 2020. [Online]. Available: https://doi.org/10.1063/5.0007045

[10] M. Metcalf, M. Cohen, and Rutherford Appleton Laboratory, *Modern Fortran Explained*, ser. Numerical Mathematics and Scientific Computation (Paperback). Cary, NC: Oxford University Press, May 2014.

A.17 Ethics Application

The following pages contain the letter received from the University of Lincoln Ethics Application System, providing a favourable opinion on the ethics of this project:

UNIVERSITY OF
LINCOLN

Application Details

Ethics Reference: UoL2023_13849

Title of Project: Optimisation of TDDFT methods in CP2K

Lead Researcher: Matt Thompson

Academic Supervisor (if applicable): Matt Watkins

Date of Letter: 22 February 2023

FAVOURABLE OPINION

Thank you for the submission of your Project Registration Form (PRF), on behalf of the committee and I am pleased to confirm a favourable ethical opinion for the above research on the basis described in the application form and supporting documentation.

The favourable ethical opinion provided is conditional to the following requirements:

1. Commencement of the research

1.1 Governance Audit: Your application may be subject to audit, should any issues your application be identified, your application may be returned to you for modification and a full ethics application *may* be required.

1.2 Risk Assessment: In accordance with H&S policy and guidance, a risk assessment must be completed or existing risk assessment reviewed/updated before any data collection commences. A copy of the risk assessment should be retained with your research data.

1.3 It is assumed that the research will commence within 12 months of the date of the favourable ethical opinion.

1.4 If the research does not commence within 12 months of the favourable opinion being issued, the lead applicant (or academic supervisor for student research) should send a written explanation for the delay. A further written explanation should be sent after 24 months if the research has still not commenced.

1.5 If the research does not commence within 24 months, the REC may review its opinion.

2. Duration of favourable opinion

2.1 The favourable ethical opinion of the Research Ethics Committee (REC) for a specific research study applies for the duration of the study, as detailed in your application (or any subsequent amendments).

3. Amendments

3.1 If it is proposed to make an amendment to the research, the lead applicant (authorised by the academic supervisor for student research) should submit an amendment to the REC by accessing the original application form on LEAS and creating an amendment form.

4. Monitoring

4.1 A REC may review a favourable opinion in the light of progress reports and any developments relevant to the study. The lead applicant and academic supervisor (for student research), is responsible for ensuring the research remains scientifically sound, safe, ethical, legal and feasible throughout its duration. The lead applicant and academic supervisor (for student research) should submit a progress report to the REC 13 months after the date on which the favourable opinion was given. Annual progress reports should be submitted thereafter.

4.2 Progress reports should be completed and submitted using the forms in LEAS.

5. Conclusion or early termination of the research

5.1 The Lead Applicant should complete the End of Study Form in LEAS once the study has completed. It is also their responsibility to inform the REC of early termination of the project or if the work is not completed.

6. Long Term Studies

6.1 The lead applicant and academic supervisor (for student research) is responsible for ensuring that the study procedures and documentation are updated in light of legislative or policy changes and also for reasons of good practice (e.g. standards for supporting documentation). This should be documented in the progress report to the REC (see above) and, where necessary, an amendment (see above) should be submitted to the REC. The REC may review its opinion in light of legislative changes or other relevant developments.

Additional guidance may be found at here

Statement of Compliance

The Committee is constituted in accordance with the University of Lincoln Research Ethics policy and E-QMS SOP E-01 Ethics Committee Operations.

Yours Sincerely

Sam Lewis | Research Governance Manager, on behalf of University Research Ethics Committee

Approved list of documents (if applicable)**:**