QUEENS UNIVERSITY BELFAST

MASTERS THESIS

Computational Methods For Ultrafast Quantum Physics

Author:
Luke ROANTREE

Supervisors:
Dr. Andrew BROWN
Dr. Jack WRAGG

A thesis submitted in fulfillment of the requirements for the degree of Masters in Mathematics

in the

RMT Group
CTAMOP

May 8, 2019

Declaration of Authorship

I, Luke ROANTREE, declare that this thesis titled, "Computational Methods For Ultrafast Quantum Physics" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"It doesn't stop being magic just because you found out how it was done."

Terry Pratchett

QUEENS UNIVERSITY BELFAST

Abstract

School of Mathematics and Physics
CTAMOP

Masters in Mathematics

Computational Methods For Ultrafast Quantum Physics

by Luke ROANTREE

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Acknowledgements

This Masters project was undertaken in the Center for Theoretical Atomic, Molecular, and Optical Physics research department of Queen's University Belfast, under supervision of Drs. Andrew Brown and Jack Wragg.

I would like to thank each of Drs. Brown and Wragg for contributing enormously to my understanding of, insights to, and enthusiasm for the areas of mathematics, physics, and computation associated with this project; consistently both providing plentiful support in the development of the project and including me in the development of their own research for my own benefit.

Additionally, I would like to thank the other Mathematics Masters students in Queens for their support and solidarity throughout a stressful and fantastic year.

Contents

List of Figures

List of Tables

Goals & Motivations

Motivations

The study of the evolution of quantum systems covers a wide field of current research, and has produced both fantastic insights to the world around us as well as enabling the development of exciting new technologies.

In particular, the investigation of ultrafast Laser-atom interactions has led to outstanding results even outside quantum or optical physics. An example of this includes that RMT, a code developed to model ultrafast interactions with multielectron systems, is due to be used as a benchmark for performance testing on ARCHER - a large supercomputer in Edinburgh. Another example is a recent development in Heriot-Watt, where ultrafast Lasers were used to succesfully weld metal directly to glass (*cite https://epsrc.ukri.org/newsevents/news/hwultraf *)

While quantum theory as a concept has existed for some time, only certain types of quantum system can have their evolution solved analytically; others require numerical approximations to be made. Until relatively recently, the available computational resources needed for the majority of these models to any degree of accuracy were not available, but now with modern technologies and by exploiting massively parallelised computer systems, we can calculate extremely accurate approximations for the evolution of these systems; allowing new quantum phenomena to be explored.

Goals

In this project I hope to investigate and improve methods of modelling quantum systems and their evolution, in order to broaden the range of quantum phenomena that can be simulated by decreasing the computational resources needed to create and evolve the necessary models to simulate these phenomena.

Additionally I hope to investigate applications of linear algebra quirks to improve computational models, develop an understanding for parallelised computation, and to meaningfully contribute to a current research project.

Chapter 1

Background Theory

1.1 Overview of Relevent Quantum Theory

In this thesis, the interactions of laser pulses with quantum systems are investigated. The systems investigated are limited to Hydrogenic atoms; that is spherically-symmetric systems with a central attractive potential field; the Coulomb potential. Such systems allow several simplifications to calculations based on symmetry, while still remaining useful models of many real molecules, atoms, or sub-atomic particles. Before describing the mathematical models used in the investigation, some relevant quantum theory, numerical analysis, and computational methods are revisited.

1.1.1 Schrodinger Equation In Multiple Forms

The 'Schrodinger Equation' describes the wavefunction of a quantum system, and how that wavefunction changes dynamically with time (* cite - C. Cohen-Tannoudji et. al. "Quantum Mechanics", Wiley-VCH, 2005 *). In it's most general, time-dependent, form the Schrodinger equation is written;

$$i\hbar \frac{d}{dt} |\Psi(t)\rangle = \hat{H} |\Psi(t)\rangle ,$$

where $\hbar$ is Planck's constant, Ψ is the wavefunction of the system, and $\hat{H}$ is a Hermitian operator describing the energy of the system - known as the Hamiltonian of the system, and in general is time-dependent.
Should I go into more detail on the Hamiltonian?
To get a 'snapshot' of the wavefunction at a given instant, the time-independent Schrodinger equation (TISE) can be considered;

$$E |\Psi\rangle = \hat{H} |\Psi\rangle$$

(where Ψ is the wavefunction of the system at the particular 't' of the snapshot, and E is the energy of the system)
This equation can be solved as an eigenvalue problem, where the wavefunction of the system can be described in terms of eigenfunctions of the Hamiltonian - each with associated energy corresponding to the eigenfunctions' eigenvalues.

For a system with a single non-relativistic particle, under the influence of a potential V (as a combination of a centrifugal potential and an external potential), the TISE can be written in differential form;

$$\left[\frac{-\hbar^2}{2m}\nabla^2 + V(\mathbf{r})\right]\Psi(\mathbf{r}) = E\Psi(\mathbf{r})$$

The radial time-independent Schrodinger equation (R-TISE) is a simplification of the TISE for the case where the system is spherically symmetric, which is the case for a Hydrogenic atom.Making this simplification allows the, generally 3D, TISE to be re-written in spherical coordinates and the two angular terms to be discounted (as the wavefunction has no dependence on the angluar directions). The result is a more easily solvable, 1D, partial differential equation;

$$\left[\frac{-\hbar^2}{2m}\frac{d^2}{dr^2} + V(r)\right]\Psi(r) = E\Psi(r)$$

1.2 Overview Of Finite Difference Methods

1.2.1 Relation to Taylor Series

A function, $f(x)$, can be described at a point $x_0 + h$ by its Taylor expansion;

$$f(x_0+h) = f(x_0) + \frac{f'(x_0)}{1!}h + \frac{f^{(2)}(x_0)}{2!}h^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}h^n + R_n(x),$$

where h is a small distance from x_0, and $R_n(x)$ is the 'remainder' term - equal to the difference between the exact solution and the Taylor approximation to n terms.

Any continuous, infinitely differentiable, function can be exactly expressed by its infinite-term Taylor expansion; and can be approximated to any required level of accuracy by choosing a small enough h, enough terms, or a suitable combination of the two.

Considering the case of truncating the Taylor approximation to f to just two terms, and solving for $f'(x_0)$, we obtain the '1 point forward finite difference' approximation to $f'(x_0)$;

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} - \frac{R_1(x)}{h}$$

Through similar derivations, the '2 point forward finite difference' method can be obtained, as well as the n-point method. Note that so far we have only looked at using points previous to our point in question to calculate the approximation - this does not need to be the case, and will be elaborated upon in Chapter 2 in the context of our project.

1.2.2 Relation to Definition of a Derivative

The definition of the derivative with respect to x of a function, $f(x)$, is given by;

$$f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}.$$

To find an approximation for the derivative of this function at a point, a, we can simply choose a suitably small h and plug in $x = a$;

$$f'(a) \simeq \frac{f(a + h) - f(a)}{h}.$$

This result is identical to the '1 point forward finite difference' approximation to the first derivative of a function, and highlights the relationship between derivatives and their FD approximations.

1.2.3 Comparison To Basis Expansion Methods

The Taylor series expansion of a function is a decomposition of that function over a basis set of functions; the polynomials. This approach does not need to be restricted to this particular basis set, any spanning set of functions could be used in its place (with different coefficients needed). A common alternative is a Fourier series representation; this is more suitable in some cases, particularly ones where the function being modelled is periodic - due to the periodic nature of its basis set of sine and cosine waves.

Fourier Expansions: The Fourier expansion of a periodic function, $g(x)$, is given by;

$$g(x) \simeq \sum_{n=-N}^{N} c_n \cdot e^{i\frac{2\pi n x}{P}},$$

where P is the period of the function, N is the order of the approximation, and c_n are appropriate coefficents.

The Fourier expansion can alternatively be explained as an extension of the Taylor expansion in the case of a complex function.

The main differences between FD methods for obtaining derivatives to functions and Fourier-based methods include the requirement for periodicity of the function in the case of the Fourier-based methods.

Other Polynomial Expansions: As stated above, there is nothing unique about the decomposition of a funciton over the polynomials in the Taylor expansion; any spanning set of functions is suitable. Even when using polynomials, there are multiple different spanning sets to choose from - one being the standard polynomials, as we used in the Taylor expansion, but also others are commonly used - including the Legendre polynomials, Chebyshev polynomials, and Jacobi polynomials.

1.2.4 Definition of Stability of a FD Scheme

Defn. (Stability): A F.D. scheme is 'stable' if the errors at a given timestep do not cause subsequent errors to be magnified.

In general, this may occur if the size of the timestep is too large with respect to the distance between spatial points. Exact formulae to determine whether or not a scheme will be stable can be calculated, with different results depending on the FD sceheme implemented. In the case of a 'Crank-Nicholson' / 'forward-time, central-space' scheme applied to the 1-D heat equation, the ctrieria for stability is given by;

$$\frac{\alpha \Delta t}{(\Delta x)^2} \leq \frac{1}{2},$$

where Δt is the timestep size, Δx is the distance between spatial points, and α is a constant. This shows that, in order to maintain stability, if you want to obtain an approximation with twice the precision (in terms of spatial grid points) you will need to use a timestep four times smaller - showing that accounting for instability in FD schemes can have large associated computational costs.

1.3 Parallel Computation Overview

Parallel computing differs from traditional, serial, computing in that instead of having one processing unit performing calculations until the computation is complete, several processing units perform calculations simulataneously until the computation is complete.

An analogy to this in a computer-free context would be comparing one student solving eight homework questions to eight students each solving one of the questions.

However, computational problems cannot usually be perfectly split up and divided among different processing units - sometimes some work needs to be performed in a serial manner in order to initialise the problem, or recombine solutions at the end of the computation, and sometimes different processing units need to share information between them which adds additional cost.

In terms of the previous analogy; these are similar to the additional work required due to one student needing to initially tell the others which questions they should work on, one student needing to re-write all the solutions neatly at the end, and the possibility of one question relying on the result from a previous question - and so stalling that student until the other is finished, and requiring them to question the other student.

1.3.1 Types Of Parallelism

There are three main types of parallelism [1];

- **Shared Memory Systems**: These are systems consisting of several processors that are all able to access a shared memory
- **Distributed Systems**: These are systems with several seperate 'units', each with a processor and individual memory, which connect to each other over a network
- **Graphic Processing Units**: These are used as co-processors, to perform highly parallelised numerical tasks given to them by a main processor

Additionally, there are three main approaches to implementing parallelism, each roughly corresponding to one of the above 'types' of parallelism [1];

- **Multi-Threading**: This parallelism model consists of one 'master' thread, or processing unit, where the process is initialised and then the work

isplit into subtasks and distributed to several 'worker' threads to be completed. While the worker each act independently, all threads have access to shared global memory. Changing the state of this global memory allows for information to be shared between threads, but at the risk of errors caused by several thread attempting to update the same address at the same time.

- **Message Passing**: In this model, subtasks are divided among several processing units - each with its own additional local memory. The need for a shared, global, memory is removed in this model by allowing the processing units to directly send messages between them rather than updating the global memory. This comes at the cost of requiring each processing unit to be continuously 'listening' for incoming messages, rather than simply checking the global memory when needed.

- **Stream Based**: This model differs strongly from the other approaches, and has a much less general application scope. Here, instead of splitting up computational tasks, only data is split up - the same instructions are passed to each processing unit, but with different data on which to perform the instructions.

1.3.2 OpenMP & MPI

OpenMP is a leading software utility used for implementing multi-threaded programs via an exposed API. OpenMP is designed to work on shared-memory systems, and controls the distribution of work to each thread and handles the risk of 'race conditions' (where multiple threads are waiting on each other, and so none can continue) by ensuring task-parallelism in the subtasks assigned to each thread or by enforcing the use of work-sharing constructs.

MPI (Message Passing Interface) is another software utility, intended for use in communication across a distributed system. It differs in that access to a shared memory is not assumed, and all communication is controlled from a single 'master' known as the communicator.

Chapter 2

Time Independent Case - A Starting Point

2.1 Our Problem

In this project, we seek to model Hydrogenic systems and the evolution of their corresponding wavefunctions under different external fields. To do this, we must first solve the time-independent Schrodinger equation to obtain a basis of eigenstates to use to describe our wavefunctions. In this chapter, we investigate numerical approaches to finding these eigenstates and attempt to optimise their accuracy and runtime efficiencies.

2.2 Central Finite Difference Methods

Assume overview of FD methods covered in Ch.1

2.2.1 Backwards Terms

In Chapter 1, we introduced the '1 point forward finite difference' method; an alternative to this approach is to use a 'backwards finite difference' method, where instead of approximating

$$\delta f\left(x\right) \simeq \frac{f\left(x+h\right)-f\left(x\right)}{h}$$

we approximate it as

$$\delta f\left(x\right) \simeq \frac{f\left(x\right)-f\left(x-h\right)}{h}$$

i.e., approximating each point from the succeeding point rather than preceding.

Note that in both of these cases, taking the limit as $h \to 0$ gives the definition of a derivative; and so as $h \to 0$, the exact solution is obtained.

These are both accurate to a tolerance of $O(h)$ due to the truncation of the Taylor series representations of the function, and provide reasonable approximations in cases where there are no sharp peaks or discontinuities in the function over intervals narrow with respect to h, as these features can either be missed or cause $\delta f(x)$ to be poorly approximated in these areas. One way to obtain a more accurate estimate of $\delta f(x)$, and limit the impact of potential peaks / discontinuities is to use a 'central finite difference' method;

$$\begin{aligned}
\delta f\,(x) &= \frac{f\,(x+h) - f\,(x)}{h} + O(h), \\
\delta f\,(x) &= \frac{f\,(x) - f\,(x-h)}{h} + O(h) \\
&\implies \\
2\delta f\,(x) &= \frac{f\,(x) - f\,(x-h)}{h} + \frac{f\,(x+h) - f\,(x)}{h} + O\left(h^2\right)
\end{aligned}$$

As the first terms in the errors introduced from unused Taylor expansion terms will exactly cancel each other

$$\implies \delta f\,(x) = \frac{f\,(x+h) - f\,(x-h)}{2h} + O\left(h^2\right).$$

It can be seen here that the same number of function calls to f are required here, and so the computational expense is the same as previously, but we obtain an estimate accurate to $O(h^2)$. Additionally, by using both forward and backward points, we limit the impact of sharp features in the function over the interval.

2.2.2 End Points

We have seen that a CFD method will give us a more accurate approximation to $\delta f(x)$, provided that we know the values of the function at points to either side of the point in question; this leads to the problem that at the boundaries of our interval, we won't know the value of the function at one of these grid points. There is no one solution to this problem, as this CFD approach can be applied to a wide variety of problems. In our case, from physical intuition we know that at distances far away from the particle the wavefunction should

be near-zero - using this we can choose to model $f(\alpha)$ for any α outside of our interval as 0, allowing us to use the CFD even at the boundaries.

2.2.3 Getting Coefficients

We have seen examples using two points to calculate the derivative but, in general, any number of points can be used for a finite difference method; with the expected accuracy of the approximation increasing with each additional point. In these cases, not all points contribute equally to the approximation. The set of ratios at which they contribute are called the 'finite difference coefficients' of that finite difference scheme. These coefficients can be calculated by solving a system of linear equations (* CITE *);

$$\begin{pmatrix} 1 & 1 & ... & 1 & 1 \\ -q & -q+1 & ... & p-1 & p \\ (-q)^2 & (-q+1)^2 & ... & (p-1)^2 & p^2 \\ ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... \\ (-p)^{p+q} & (-p+1)^{p+q} & ... & (p-1)^{p+q} & p^{p+q} \end{pmatrix} \begin{pmatrix} a_{-q} \\ a_{-q+1} \\ a_{-q+2} \\ ... \\ ... \\ ... \\ a_p \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ ... \\ m! \\ ... \\ 0 \end{pmatrix}$$

Where p refers to the number of backward points being used in the approximation of the point in question and q the number of forward points. For a CFD setup, $p = q$. a_i is the i^{th} coefficient, and m the order of the derivative being approximated. Cramer's rule, or another method, can then be used to solve the system of linear equations to obtain $\{a_i\}$.

Finite Difference approximations, in cases like ours where function values at points outside the interval of interest are taken to be 0, can be described in matrix form as an eigenvalue problem;

$$\left[\frac{-\hbar^2}{2m}\frac{d^2}{dr^2} + V(r)\right]\Psi(r) = E\Psi(r)$$

can be approximated with a CFD method as

$$\begin{pmatrix} a_0 + V(r_0) & a_1 & ... & a_p & 0 & ... \\ a_{-1} & a_0 + V(r_1) & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... \\ 0 & ... & ... & a_{-p} & ... & a_0 + V(r_n) \end{pmatrix} \mathbf{\Psi_j}(\mathbf{r}) \;=\; E_j \mathbf{\Psi_j}(\mathbf{r}).$$

where $\{E_j\}$ and $\{\mathbf{\Psi_j}\}$ are eigenvalues and eigenvectors of the matrix; forming a basis set for all possible solutions. If we know that we can exactly, or at least fairly accurately, represent our desired initial wavefunciton with only the first few of these eigenvectors, we don't need to calculate the rest and can stop early - in fact, if our initial wavefunction is itself an eigenstate, then we only need to calculate that single eigenstate. Further, physical significance can be inferred from these values; each eigenvalue corresponds to the energy of a distinct state, and states corresponding to negative energies are 'bound' states.

2.2.4 Useful Properties of CFD method in matrix form

- Sparse:
  From the above equation it can be seen that other than along the central diagonal, and a few off-central diagonals, all elements in the matrix are 0; meaning that for a large model (high accuracy), the matrix is very sparse. Krylov Subspace based Arnoldi methods are known to be extremely efficient at finding eigenvectors of sparse matrices, due largely to the fact that they rely on repeated QR decomposition (* CITE SAAD *) which involves conversion to a Hessenberg matrix and then to a triangular matrix - which is much faster to do when the matrix is already almost in Hessenberg form (or already in Hessenberg form in the case of a three point or fewer method).

- Hermitian:
  As all of the finite difference coefficients are real, and assuming the arbitrary potential function V is real-valued at all points, the entire matrix is real. Additionally, for any CFD method, the matrix is symmetric. Combining these two statements, we have that the matrix is Hermitian; knowing this we can choose the eigensolver carefully to make use of this, for example using one based on Lanczos iteration - a special case of Arnoldi iteration with stages removed due to knowledge that the matrix is Hermitian.

2.3 Implementation

2.3.1 General Approach

> Set up a set of equally spaced points spanning an interval containing the particle

> Calculate the potential at each of the points

> Build a 'baseline' Hamiltonian of FD coefficients along the appropriate diagonals and 0 elsewhere

> Add the potential values to the central diagonal

> Use an eigensolver algorithm / tool to find eigenpairs of the Hamiltonian

2.3.2 Notes On Finding Eigenpairs

Most eigensolvers, including all of the ones I use in the course of this investigation, are based on variations on Krylov Subspace Iteration. Krylov Subspace Iteration in turn is a particular case of Subspace Iteration, which is a generalised application of Power Iteration. To provide insight to how the eigensolvers work, and later optimisation techniques, I explain here in detail the Power Iteration method followed by outlines of adaptions to it for Subspace Iteration and how the outcome differs for that, followed by similar outlines for the other methods.

Power Iteration is a method for obtaining the dominant (largest corresponding eigenvalue) eigenvector, $\mathbf{x}$, of a square matrix, A. Approximations to x of increasing accuracy are given recursively by

$$\mathbf{x_0} = \text{any non-zero random vector of the right length}$$

$$\mathbf{x_n} = \frac{A\mathbf{x_{n-1}}}{||A\mathbf{x_{n-1}}||}$$

We know from (* REF CH1 *) that an arbitrary vector, v in the space described by A can be spectrally decomposed into a weighted sum of eigenvectors, $\{\mathbf{e_i}\}$ of A; $\mathbf{v} = \sum_i c_i \mathbf{e_i} for \{c_i\} \in \mathbb{C}$.

Therefore we can write $A\mathbf{v}$ as $A\sum_i c_i \mathbf{e_i}$, and by orthogonality of eigenvectors we can rewrite this as $\sum_i c_i (A\mathbf{e_i})$.

By the definition of an eigenvector, each $A\mathbf{e_i}$ is equal to $\lambda_i \mathbf{e_i}$ where λ_i is the eigenvalue of that eigenvector. This allows us to further re-write $A\mathbf{v}$ as $\sum_i c_i (\lambda_i \mathbf{e_i})$.

From (* REF CH1 *), we additionally have that if $(\lambda, \mathbf{e})$ is an eigenpair of B, then $(\lambda^k, \mathbf{e})$ is an eigenpair of B^k. Applying this here, we have that $A^k\mathbf{v}$ can be written as $\sum_i c_i \left(\lambda_i^k \mathbf{e_i}\right)$.

From this it is clear that, for a high enough k, the term in the summation with the largest eigenvalue will be far larger than any other term - making it a good approximation for the most dominant eigenvalue after renormalisation.

Additionally it can be seen that the error in this approximation can be estimated using the next-largest eigenvalue; $\epsilon \simeq |\frac{\lambda_1}{\lambda_2}|^k$ where λ_1, λ_2 are the two most dominant eigenvalues. This shows that the rate of convergence is given by $|\frac{\lambda_1}{\lambda_2}|$.

Subspace Iteration is an extension to Power Iteration, where a set of m random vectors are initialised instead of only one, and these are repeatedly multiplied by A, then orthonormalised with respect to each other (as opposed to simply normalised like the Power Iteration method). The orthonormalisation process, usually done via repeated QR decomposition, ensures that the initial vectors will not all converge to the same, largest, eigenvector - instead one will converge to the next most dominant one, another to the third most, all the way to the m^{th} most dominant.

Krylov Subspace Iteration is a special case of Subspace Iteration where the initial subspace is not fully random vectors, and is instead built from a single random vector, $\mathbf{b}$, such that the subspace consists of vectors $\{\mathbf{b}, A\mathbf{b}, ..., A^m\mathbf{b}\}$. One reason to use a Krylov subspace over a randomised one is that vectors in the Krylov subspace will have large spectral components for dominant eigenvectors, as shown in the above proof of Power Iteration convergence. This results in in a reduced number of subspace iterations needed for convergence.

Other improvements on these methods exist, including the 'implicitly restarted Arnoldi' method and 'Hermitian Lanczos' method - which I do not describe further here, other than to describe the Lanczos method as being able to ignore redundant steps in the case that A is Hermitian.

In this investigation, while I made use of these results through eigensolver choice and parameter tuning, I did not recreate any of these algorithms directly; instead opting to make use of existing implementations in LAPACK and ARPACK, ported to Python through the Numpy and Scipy libraries.

2.4 Results

2.4.1 Expected Results for Particle In A Box, Soft-Core

In the case where there is no external potential field, i.e. $V = 0$ in interval, and the wavefunction's amplitude is forced to be 0 outside of the interval, i.e. $V = \infty$ outside interval, there is an easily obtained analytic solution for the wavefunction known as the "Particle in a box" solution(* cite Hyperphysics `http://hyperphysics.phy-astr.gsu.edu/hbase/quantum/pbox.html` *);

$$\Psi_i(r) = \sqrt{\frac{2}{L}} \sin\left(\frac{i\pi\left(r + \frac{L}{2}\right)}{L}\right), \; i \in \mathbb{Z}^+$$

where L is the width of the interval.

These Ψs are all eigenstates for the system, and so an arbitrary wavefunction in the system can be written as a combination of these. From the mathematical description of the eigenstates we see that the ground state should have exactly one turning point, the first excited state should have 2, and so on - as well as maintaining constant wavelength throughout the wave.

A visual representation (* CITE Hyperphysics *) of the first few of these states is given below for qualitative comparison with the simulated results (using 20 thousand grid points);

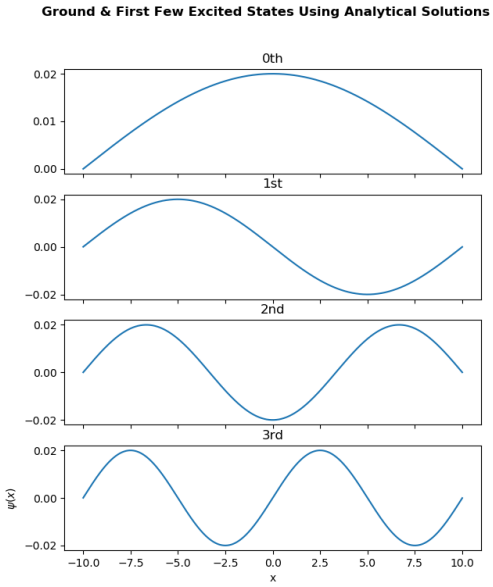

FIGURE 2.1: Expected First Few Excited States for a Particle-In-A-Box System

2.4.2 Three-Point FFD, BFD, CFD Results with no Potential (Particle In A Box)

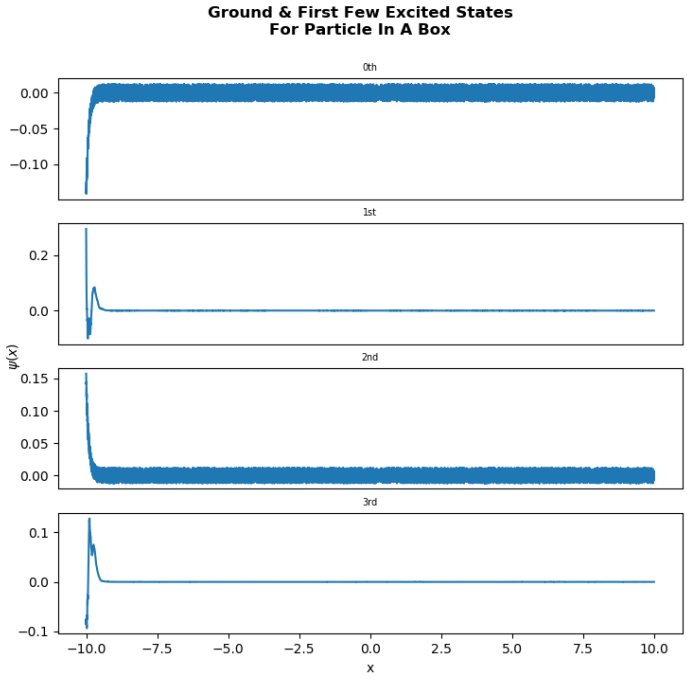

FIGURE 2.2: First Few Excited States of Particle-In-A-Box System Calculated by FFD

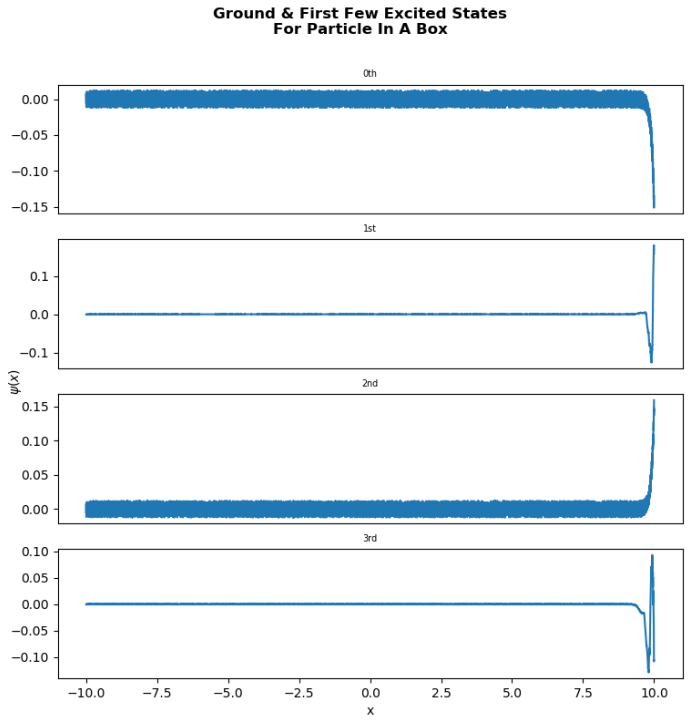

FIGURE 2.3: First Few Excited States of Particle-In-A-Box System Calculated by BFD

For both the Forward and Backwards Finite Difference calculations, the results do not at all resemble the analytic solutions, and suggest large stability issues have arisen. This instability comes primarily from consistently over- or under-estimating the gradient over the course of the interval, due to the shape of the wave and the fact that no correcting terms from the other direction are ever used in either case. A central finite difference method will have these terms, and so should be able to avoid this instability;

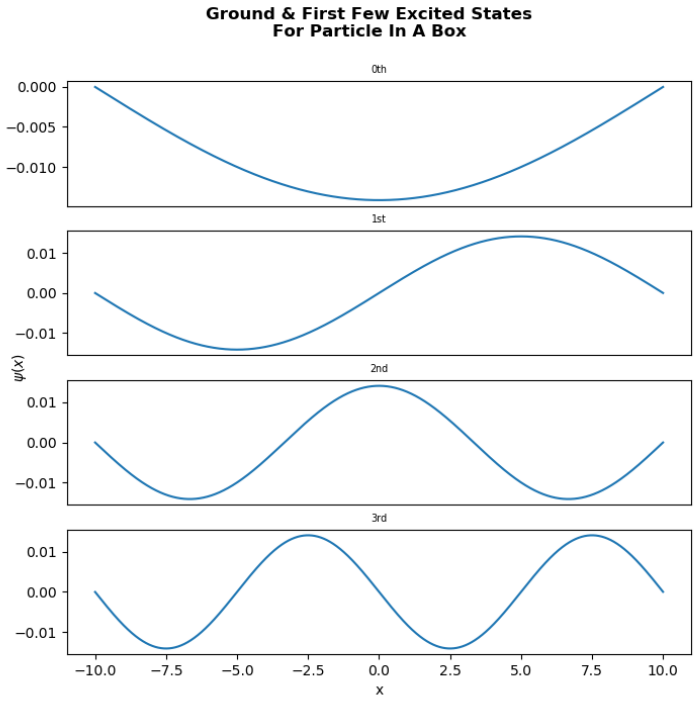

FIGURE 2.4: First Few Excited States of Particle-In-A-Box System Calculated by CFD

These results qualitatively mimic the expected ones very closely, with the same numbers of nodes, symmetries, and constant wavelengths. This provides a good indication that this is a good method to use for other models that cannot be solved analytically.

As the same order of finite difference method, and so the same computational expense, was used in all three of these calculations, it is clear that the CFD method is the only one worth developing further. As stated earlier, the CFD method should be accurate to $O(h^2)$, while the FFD and BFD methods are only accurate to $O(h)$ - this combined with decreased stability resilience resulted in the above calculations, which are not representative of the analytic solutions at all.

2.5 Optimisations

2.5.1 Sparse Storage

The first improvement to the chosen implementation (matrix method) involved moving from storing the Hamiltonian in a standard storage format, in this case a numpy ndarray, to a more suitable data structure; a sparse matrix.

The motivation for this is that to obtain a solution to a very high accuracy a large number of grid points are needed and, with ndarray storage, the memory needed to store the Hamiltonian scales with the square of the number of grid points - meaning that the maximum possible accuracy is limited by the size of the Hamiltonian that can be held in memory.

From <ref earlier>, we see that the Hamiltonian is tri-diagonal for our 3 point CFD approach. As a result we know everything about the Hamiltonian if we know its tri-diagonal elements, and so switching to a sparse diagonal storage data structure (scipy's sparse.diags) we can throw away the unneeded zeros and obtain a more efficient Hamiltonian representation whose size scales linearly with the number of grid points.

Switching to this data structure expanded the maximum size of Hamiltonian that could be held in memory on my laptop (8GB RAM) from one using 20 thousand grid points to one using over 10 million grid points.

Limitations of this data structure include that more specialised functions are needed to operate on sparse matrices; for example numpy's linalg.eig eigensolver is unable to operate on sparse data. Luckily, scipy's sparse library has a range of equivalent tools designed to replicate the effects of many of numpy's operations for sparse data structure, including several eigensolvers which we now look at as an additional optimisation method.

2.5.2 Eigensolver Choice

As seen in the above section, a sparse storage system allows for more accurate calculations to be performed. There are two main sparse eigensolvers available to use in place of numpy's linalg.eig here; scipy's sparse.linalg.eigs and sparse.linalg.eigsh (which we will refer to from here as 'eigs' and 'eigsh' respectively). These both work similarly, using Arnoldi-based methods for subspace iteration to obtain the eigenpairs, with the difference being that the former uses standard Arnoldi iteration, while the latter uses Lanczos iteration * CITE *.

Lanczos iteration is a simplification of Arnoldi iteration for the case where the matrix in question is Hermitian; from * cite previous *, we know that a CFD-based Hamiltonian will be real and symmetric - and therefore Hermitian.

Additional optimisations involving this eigensolver are possible, including the optional parameter 'k' which allows the number of desired eigenstates to be specified. By default, the 'k' most dominant eigenstates will be returned (dominant implying largest eigenvalue magnitude).

As we only care about bound states, with eigenvalues below zero, we don't necessarily want the 'k' most dominant eigenstates; rather we want the 'k' most dominant eigenstates *corresponding to negative eigenvalues*.

Another optional parameter, 'sigma', can be used to achieve this. 'sigma' allows a 'target' eigenvalue value to be given and, through performing shift-invert preconditioning (* cite my L3? *), the 'k' eigenstates closest to that value be returned.

By passing in a suitably large negative number, we ensure that the returned eigenstates will correspond to the ground state and 'k-1'-first excited states. This allows a massive reduction in computational effort needed to obtain particular bound states as, without these, all eigenstates would need to be calculated and then sorted.

For a model TISE solution using a model with 10k grid points to obtain only the Ground State, and using an estimate for the ground state for the 'sigma'/Shift-Invert parameter within 10% of the real value, the eigsh routine on average took 93.421% of the time that the eigs routine needed to converge - averaged over 7 trials.

For the same model and goal; when an estimate that was very far off the real value (around 200 times the real value) was used instead in the Shift-Invert parameter, the eigsh and eigs routines performed very similarly to each other - with eigs being 0.5% faster averaged over 7 trials.

Using 1 million grid points very similar results were obtained, with eigsh taking 94.753% as long as eig with the good estimate and 99.832% as long with a the bad estimate - based on this, it seems the benefit of using eigsh does not scale with Hamiltonian size.

Switching to calculating the Ground state and first 9 excited states, with a 10k grid point model, the benefit of eigsh over eigs for the accurate esetimate appeared to decrease slightly - taking 97.933% as long, and again for the bad estimate they took very similar times - with eigsh taking 100.263% as long as eigs.

Additional optimisations involving the 'sigma' parameter are described below, which lead to further large reductions in computational expense.

2.5.3 FastBOI Improvement

An additional efficiency improvement that I developed makes use of results from (* cite previous *), where it is seen that the rate of convergence of an eigenstate during subspace iteration is proportional to the magnitude of the

corresponding eigenvalue and that eigenstates of of a matrix are also eigenstates of that matrix's inverse (if it exists), with corresponding eigenvalues equal to the reciprocal of the original eigenvalues.

Combining these it can be seen that if an estimate of an eigenstate's eigenvalue is known, say λ, then λ times the identity matrix, I, can be subtracted from the original matrix to create a new matrix with the same eigenstates, but with eigenvalues equal to the original ones minus λ.

As a result, the eigenstate desired will have a 'new' eigenvalue of 0 (or very near, depending on accuracy of the estimate) - thus, if we invert the matrix, the result will have the same eigenvectors but reciprocal eigenvalues, which in the case of the desired eigenstate will be very large and tend towards ∞ with increasing accuracy.

Using the knowledge that the rate of convergence is proportional to eigenvalue magnitude, this shifted and inverted matrix will converge extremely rapidly to the desired eigenstate during subspace iteration, with increased speed if the accuracy to which the original estimate is known is increased.

Based on this result, I developed a 'Fast Boosted Optimiser Iteration' method to improve convergence rates for the eigensolver used to find bound states from my Hamiltonian. Another result used in the development of this approach is that a model with fewer grid points than another should still produce relatively correct eigenstates and eigenvalues, simply to a lower accuracy than a model with more grid points.

Based on these results, the general idea of this approach is to find the ground state eigenvalue for a model with a greatly reduced number of grid points, and then use this as an estimate in the 'sigma' parameter of the eigensolver for the model with full number of grid points - greatly increasing the convergence rate for the full model at the cost of having to solve an extra, much smaller, problem.

Additionally, although only useful for models with a very large number of grid points, this approach can be stacked; a small model can generate an estimate ground state eigenvalue for a medium sized system, which can then generate a better estimate for the full very large system. A flow-chart describing the basic approach is described below, followed by adaptions to allow several 'boosting' stages.

Basic FastBOI algorithm:

Improved (multi-stage) FastBOI algorithm:

Using a model with 50 thousand grid points, only seeking the Ground State, and using a very large negative number (around 10^3x the real energy)

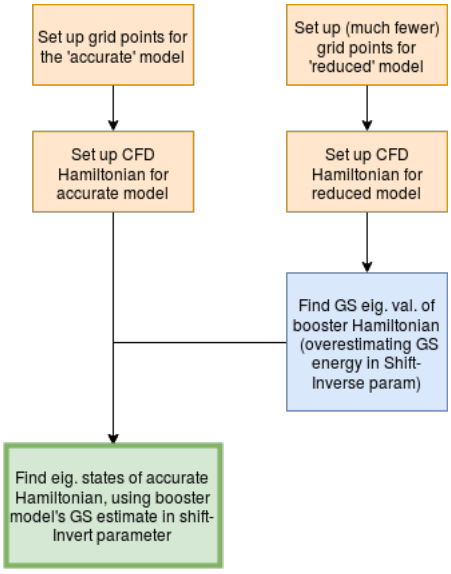

FIGURE 2.5: Basic FastBOI algorithm

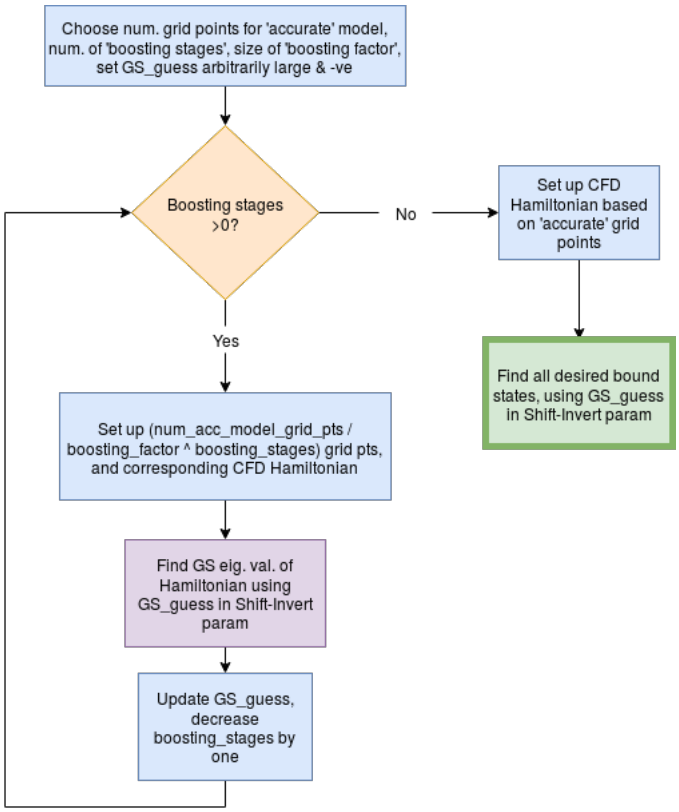

FIGURE 2.6: Improved (recursive) FastBOI algorithm

as the baseline estimate - using the FastBOI algorithm with a boosting factor of 100 and a single boosting stage was 90.948 times faster than not using it, jumping to 107.928 times faster when the boosting factor was increased to 1000.

Using the same comparison, but with a 500 thousand grid point model,

the FastBOI algorithm with 1 boosting stage and a boosting factor of 100 was 58.914 times faster than without, with 1 boosting stage and a boosting factor of 1000 it was 77.749 times faster, and with 2 boosting stages and a boosting factor of 100 it was 87.1 times faster.

It was noticed that the time to solve the model without using the FastBOI algorithm scaled linearly with model size - and so for the following, larger, models estimates were used for the comparisons as each calculation woud otherwise take several hours. With this assumption; for a model with 5 million grid points and only seeking the ground state, the FastBOI algorithm with 1 boosting stage and a boosting factor of 1000 was about 89 times faster, with 2 boosting stages and a boosting factor of 250 it was about 96 times faster, with 3 boosting stages and a boosting factor of 40 it was about 93 times faster, and with 4 boosting stages and a boosting factor of 10 it was about 85 times faster.

Larger models were uable to fit into RAM on my laptop, and so could not be tested.

2.5.4 Pre-Trained Predictive Model Improvement

An alternative to the Boosted Optimiser Iteration method, suitable for other use cases, is to find estimates to the Ground State energy for many differently shaped potentials and train a predictive model to generate an estimate to use for the Shift-Invert parameter of the eigensolver based on the shape of the potential being used. To investigate this approach I used reduced models with 1000 grid points, and generated 1000 different potentials.

For each model I then found the ground state eigenvalue, using a massively negative estimate in the Shift-Invert parameter, and then recorded the ground state value along with statistics to describe the potential used in that model; such as mean, median, skew, standard deviation, kurtosis.

Several machine learning models(* cite sklearn `https://scikit-learn.org/stable/modules/g` `https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegress` including a 3-hidden-layer neural network and a boosted random forest, were trained on a randomly selected 85% of the data. After feature selection and hyperparameter tuning, the models ranged from 97.3% (simple linear regression) to 99.996% (neural network) accurate on the unseen 15% of the data (measured using Mean Average Error). Once trained, these predictive models can be saved for future usage - allowing for a near-immediate Ground

State estimate in the future as opposed to having to solve one or more reduced models every time.

Using a prediction from the neural network model in the Shift-Invert parameter when solving a 10k grid point model gave around a 400x speedup compared to using a large negative guess - including the time taken to generate statistics about the potential and feed them into the neural network.

Limitations of this approach include that, unless similar calculations are going to be performed often, the time taken to generate the synthetic data and train the predictive model is much greater than the cost of not using a good ground state estimate, and also that this approach can't 'stack' like the Boosted Optimiser Iteration approach - limiting the speedup benefit for very high-precision models.

Advantages of it are that, once trained, there is very little extra work or computational effort needed to allow the speedup, and also over time the higher-accuracy ground state energies that are calculated can be saved and the predictive model retrained using them.

Chapter 3

TDSE Solution & Optimisation

3.1 Time Evolution of a Quantum State

3.1.1 Exact Solutions and Limitations

The evolution of a quantum state can be described fully by the Schrödinger equation, in its time-dependent form, which we will refer to from now as the TDSE. The TDSE states that an arbitrary quantum state, Ψ, will evolve according to;

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

where H is the Hamiltonian of the system, and is in general also time-dependent. We now consider the case where the Hamiltonian does not evolve with time - this is a much easier case to solve, in fact only a few particular systems with non-constant Hamiltonians are analytically solvable. We will investigate these numerically later in this chapter.

Solving this equation, we obtain

$$\Psi\left(t\right) = e^{\frac{i}{\hbar}Ht}\Psi\left(0\right),$$

where a constant of integration has been neglected as it amounts to a global phase-shift.

By spectrally decomposing Ψ into its component eigenstates, $\sum_i c_i \mathbf{e_i}$, we can rewrite this as;

$$\Psi\left(t\right) = \sum_j e^{\frac{i}{\hbar}\lambda_j t} c_j \mathbf{e_j}\left(0\right),$$

allowing us to see that each individual eigenstate oscillates in amplitude, with a frequency determined by the energy of the eigenstate.

Solutions for this case can therefore be exactly calculated if the eigenstates are known, and so we can use this case to compare to our numerical model solutions to measure their accuracy.

3.1.2 The Case for Numerical Methods

The TDSE is not trivially solvable analytically other than for particular time-dependent Hamiltonians; and so to be able to investigate the time evolution of more complicated systems, or systems with more complicated external potentials, a different approach must be used.

One alternative approach is pertubation theory, where known solutions to Hamiltonians of simpler systems are used to approximate the more complicated systems along with small additional 'perturbative' terms - for example investigating the effect of a small external field on the ground state of the system. This approach's accuracy is mostly limited by the size of the pertubation, and so is useful for modelling the evolution for systems similar to an analytically solvable one with a small additional term - but not in the general case for modelling arbitrary systems. For example, trying to model the effect of a large external field on the ground state would be innacurate with pertubation theory if the corresponding pertubation was larger than the difference between the ground and first excited states.

Another approach is to use numerical methods; discretizing the Hamiltonian and wavefunction and using numerical solutions to the differential equation by approximating derivatives with series or basis expansions - as we did for the numerical solutions to the TISE.

Advantages of this approach include that computers are easily able to compute solutions to discretized models, and that these approaches are not limited to systems similar to analytically solvable ones - as they do not rely on the solutions to the differential equation being similar, instead solving the differential equation in a completely different way.

In this chapter we investigate and compare two numerical methods of solving the TDSE; a Finite Difference propagator, and a Krylov Subspace propagator. These both work similarly, using one or more previous states of the wavefunction to predict the state at the next timestep, and updating the Hamiltonian at every timestep.

3.2 Finite Difference Propagator

A finite difference approach to this problem is very similar to the one used in solving the TISE - with the main difference being that we are now propagating to approximate the first derivative in time rather than the second in space. Additionally, as we don't know anything about the wavefunction at points

in time ahead of the current timestep, we can't use any backwards terms in our finite difference scheme.

FD methods require knowledge of the wavefunction at several initial timesteps to be able to start to propagate; in the TISE solution we took the boundary points to be all 0, but that is an unsuitable assumption for this case as a zero-everywhere wavefunciton is unphysical and implies that, given a particle in the system, the particle has 0 probability of existing in any region of the system - contradicting that we have a particle existing in our system.

Instead, we spectrally decompose the wavefunction and use the previously described analytic solutions (noting that the evolution operator is only constant in the abscence of a time-varying field) to evolve each component eigenstate to build up the first few timesteps. This is equivalent to assuming that no external potential is applied to the system for the first few timesteps, which is a much more realistic assumption.

3.3 Krylov Subspace Propagator

The Krylov Subspace approach works differently, calculating the state of the wavefunction at the next timestep in terms of only the current one.

To do this, a krylov subspace is built from a matrix representation of the Hamiltonian at the next timestep and using the current wavefunction as the starting vector for the subspace.

The vectors in the subspace are then added, weighted by appropriate Taylor series coefficients, to give the approximation for the wavefunction at the next timestep. This approximation increases in accuracy with increased subspace size, although almost all information is contained in the first few terms and there is a rapidly decreasing benefit to additional subspace elements - the rate of the descrease here is considerably greater than the decrease in benefit of using additional finite difference terms in the FD propagator case. However, the computational costs of using more terms are not the same in both propagators and so it is non-trivial to determine which method is more efficient in any given case - this will be discussed later in this chapter.

To summarise the process; First the Krylov subspace, $\mathbb{K}$, is built:

$$\mathbb{K} = \left[\Psi\left(t\right),\ H\left(t+\delta t\right)\Psi\left(t\right),\ H^2\left(t+\delta t\right)\Psi\left(t\right),\ ...\right]$$

Then a weighted sum of the subspace elements, $\{k_i\}$, is used to approximate $\Psi\left(t+\delta t\right)$:

$$\Psi\left(t+\delta t\right)=\sum_i \frac{-i\delta t^i}{i!}k_i$$

It should be noted that as the subspace size tends to infinity, the Krylov subspace propagation is identical to the full Taylor series expansion of the first derivative in time for the wavefunction; further, each term is identical to a term in the Taylor expansion and so limiting the basis to n elements is equivalent to truncating the Taylor expansion at the nth term - and so leaving an error of $\mathrm{O}\left(h^{(n+1)}\right)$.

Advantages of this approach include accurate convergence with relatively few terms needed, and that there is no need for the first few terms to be analytic solutions to the field-free case. Disadvantages of it include that a large matrix-vector multiplication must be performed for every term in the approximation, as opposed to a single one regardless of the number of terms in the FD approach.

3.4 Comparisons Between Time Propagators

3.4.1 Efficiency Comparisons For 6000 Timesteps With A Small Grid

Using only the constant Soft-Core potential, and so with no time-dependent terms in the Hamiltonian, the two propagators were used to propagate the Ground State wavefunction 6000 timesteps - using a 1000 spatial grid points to represent the wavefunction, a timestep of 10^{-3} atomic units of time, and both propagators using 6 terms. The results shown are averaged over 5 repeated calculations.

For context if attempting to reprodce the results, the calculations were performed in a Python3.7 Jupyter Notebook, running on a 3.2GHz i7u processor with 8GB of RAM.

Krylov Subspace Propagator:

Mean runtime: 32.28598s
Standard Deviation: 0.10144s / 0.31419%

Finite Difference Propagator:

Mean runtime: 26.02711s
Standard Deviation: 0.10275s / 0.39478%

Analysis:

From these results, it is clear that for this setup the Finite Difference propagator performs more efficiently - taking only around 80% of the time as the Krylov Subspace propagator. Additionally, both propagators had a very similar spread of times taken; showing that each is roughly as reliable as the other.

3.4.2 Efficiency Comparisons For 6000 Timesteps With A Large Grid

The previous experiment was repeated, changing only the number of spatial grid points used to 100,000. Due to the length of time required for these calculations, they were not averaged to obtain a more accurate result and determine the spread of results; instead the spread after 500 timesteps was investigated, using 5 calculations for each propagator, afterwards to provide some insight to the expected spread of results for the full 6000 timesteps.

Krylov Subspace Propagator:

Mean runtime: 2392.27s

Finite Difference Propagator:

Mean runtime: 2093.00s

Standard Deviations after 500 timesteps:

Krylov Subspace propagator:
Mean runtime: 198.092076s
Standard Deviation: 0.62765s / 0.31685%

Finite Difference propagator:
Mean runtime: 172.43040s
Standard Deviation: 0.58337s / 0.33832%

Analysis:

From the standard deviations at 500 timesteps, we can expect that at the full 6000 timesteps the standard deviations for each propagator will be very small compared to the mean - and similar in magnitude, with the Finite Difference propagator possibly having a slightly larger relative standard deviation. From this, we can conclude that for a model with many grid points the Finite Difference model outperforms the Krylov Subspace model, although to less of an extent than it does with a smaller model; taking around 87% of the time rather than 80%.

3.4.3 Efficiency Comparisons For 60,000 Timesteps With A Small Grid

The initial experiment was repeated, using 60,000 timesteps instead of 6,000, to investigate how the model performances scale with propagation distance.

Krylov Subspace Propagator:

Mean runtime: 327.40101s
Standard Deviation: 0.53118s / 0.16224%

Finite Difference Propagator:

Mean runtime: 263.82221s
Standard Deviation: 0.61104s / 0.23161%

Analysis:

Here we see that the Finite Difference propagator still only takes about 80% of the time to run as the Krylov Subspace propagator, matching the results for the smaller number of timesteps with the same grid size. It can also be seen that the relative standard deviations of the times taken have reduced to around half of their values for the model with fewer timesteps; this would imply that the main inconsistencies were in the setup of the models, and once that is complete there is a lower level of difference.

3.4.4 Accuracy Comparisons For 6000 Timesteps Over A Small Grid

To compare the accuracies achievable with the two methods, we are limited to cases that can be exactly solved; for this purpose we compare results for modelling field-free propagation.

The setup of the computational experiment is mostly the same as described for the first 'efficiency' test, with one difference being that the timestep size of 10^{-3} was found to be too large for the Krylov Subspace propagator to handle, causing massive instabilities regardless of the order of the propagator. Instead, a timestep size of 10^{-4} was used.

It is worth noting that the Finite Difference propagator was able to produce reliable results even for these larger timesteps and so could be more useful in some contexts than the Krylov Subspace propagator due to this. Additionally, the fact that the end results for the Krylov Subspace propagator are completely unphysical does not detract from the findings in the above efficiencies comparison; as the number of operations performed at each timestep does not depend on the magnitude of the timestep in any way, and the number of timesteps used was kept constant.

Initially sixth-order propagators were used for each method and the accuracy is measured by comparing the population of the Ground State, i.e. the Ground State coefficient of the spectral decomposition of the propagated wavefunction, at the end of the propagation.

Given that we use the ground state as our wavefunction, and there is no external field to excite the system, the population of the ground state should remain 1 throughout the propagation - the magnitudes of any increases or reductions to this would indicate numerical errors.

CASE 1 - dt = 10^{-4}, Grid Size = 1000, End Time = 0.6, propagation order = 6

Krylov Subspace Error: 1.1102230246251565e-16
Finite Difference Error: 2.220446049250313e-16

CASE 2 - dt = 10^{-4}, Grid Size = 1000, End Time = 6, propagation order = 6

Krylov Subspace Error: 0.0
Finite Difference Error: -2.220446049250313e-16

Analysis:

From these, it is clear that both are very accurate models. For clarification on the repeated result for the Finite Difference propagator, and for the fact that the first Krylov Subspace propagator result is exactly half that of the Finite Difference ones, the wavefunction is stored internally as an array of double precision floating point numbers - which have a maximum precision of -2.220446049250313e-16. Therefore the accuracy of the results are seemingly limited more by their storage medium than the fact that they are only approximations of the solution.

Because of this, in order to determine which method is more accurate compared to the other, we must either reduce the number of terms used by each propagator or greatly increase the 'end time' of the propagation.

CASE 3 - dt = 10^{-4}, Grid Size = 1000, End Time = 0.6, propagation order = 3

Krylov Subspace Error: 3.775724177756956e-10
Finite Difference Error: 0.0

CASE 4 - dt = 10^{-4}, Grid Size = 1000, End Time = 6, propagation order = 3

Krylov Subspace Error: 3.775721957310907e-10
Finite Difference Error: 1.1102230246251565e-16

Analysis:

These results show that even when using half the number of terms in the propagators the FD method can retain accuracy to within the maximum possible tolerance of the storage medium - this was unexpected as it was believed that while the first few terms in a Krylov subspace should contain almost all of the information about the wavefunction at the next timestep, the same was not expected to be true of the FD method. One reason the FD method could be more accurate at a lower propagation order in this case is that we are investigating the ground state which is relatively smooth over the entire interval.

Additionally, while less accurate than the FD propagator, the Krylov Subspace approach was still very accurate - and retained the accuracy when moving from the 6,000 timesteps case to the 60,000 case. This shows that the model is stable, i.e. does not contain terms that cause it to reduce in accuracy over time.

The FD propagator was further tested to failure, in order to determine how large of a timestep it could handle and still remain accurate. It was found that, with 3rd order propagation, after 6,000 timesteps it was still accurate to around machine precision for a timestep of $5e^{-3}$, for $6e^{-3}$ it had an error of 6.54306e-07, and for $7e^{-3}$ it had an error of 1. This shows that, while it is extremely accurate and stable for small timesteps, it rapidly loses stability if the timestep is even slightly too large.

Additionally this shows that in order to approximate the wavefunction at a given time to a given tolerance, the FD approach can use both a smaller number of terms and a larger timestep - this results in orders of magnitude fewer calculations needing to be performed, allowing it to be calculated much more quickly.

For both of these propagators the critical magnitude of the timestep before instability depends on the size of the gap between grid points. We investigated this by repeating the experiments using a larger number of grid points over the same interval (100,000 vs. 1,000). As it has already been shown that either instabilities will cause the error to quickly grow, or the error will remain fairly constant during propagation if the model is stable, we only need to compare the errors after a relatively small number of timesteps in order to determine the accuracy of the model. For this purpose, we will propagate for only 300 timesteps.

CASE 5 - dt = 10^{-4}, Grid Size = 100000, End Time = 0.03, propagation order = 6

Krylov Subspace Error: -103435.8494626187
Finite Difference Error: 0.9999999999723448

CASE 6 - dt = 10^{-5}, Grid Size = 100000, End Time = 0.003, propagation order = 6

Krylov Subspace Error: 0.8971347740628693
Finite Difference Error: 4.440892098500626e-16

CASE 7 - dt = 10^{-7}, Grid Size = 100000, End Time = 0.00003, propagation order = 6

Krylov Subspace Error: 0.9999999999999252

CASE 8 - dt = 10^{-8}, Grid Size = 100000, End Time = 0.000003, propagation order = 6

Krylov Subspace Error: 4.440892098500626e-16

Analysis:

Here we can see that when we use more grid points we need to decrease the size of the timestep to maintain stability. We see that for a grid 100 times the size, we need to decrease the the size of the timestep by around 2 orders of magnitude for the FD propagator to achieve stability, while the Krylov Subspace propagator needs the timestep to be decreased by around 4 orders of magnitude. From this we can conclude that the accuracy and stability of the solutions scale more effectively with the FD propagator than the Krylov Subspace propagator as the number of grid points increases.

3.4.5 Summary of Findings

3.5 Parallelisation of the Time Propagation

We have seen in this chapter that, assuming stability conditions are met and enough propagation terms are used, either propagator can give results at a very high level of accuracy at even distant times - with the largest limit on accuracy coming from the number of spatial grid points used to model the wavefunction. Therefore, to increase the overall accuracy we need to use a more accurate spatial representation with more grid points.

However, as seen in the above 'efficiency' comparisons, our time propagation speeds do not scale well with increased spatial grid size.

To try to reduce this problem as much as possible, the propagators were parallelised - allowing different chuncks of the wavefunction to be propagated separately on different computers/processors/cores.

To do this, both the wavefunction and the Hamiltonian were split into 'chunks' - this is equivalent to considering the time evolution of subrergions of our interval seperately, and so they can be recombined afterwards to re-form the full evolved wavefunction. A diagram describing this process is shown below (Note - this is an overview of the process, and leaves out contributions from previous timesteps in the FD case, and repetitions of this matrix multiplication for the Krylov Subspace case);

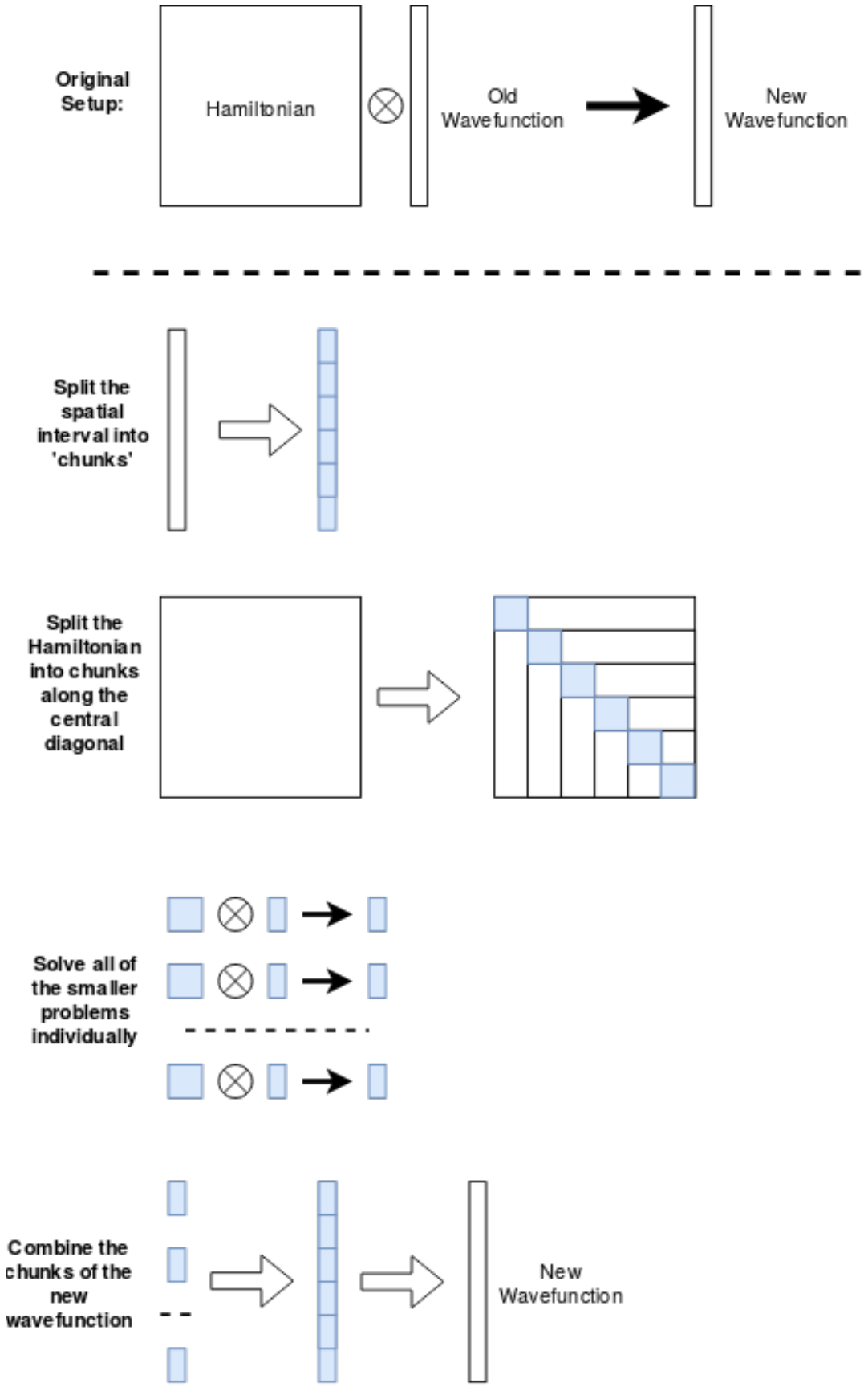

FIGURE 3.1: Description of Parallelisation Process For Time Propagation

Is Any Information Lost When The Propagation Is Split Up?

In short, yes, although the amount can be mitigated through a few different methods. The reason we lose information, or accuracy, is that by splitting up the spatial region we introduce new boundaries in the system at the edges of the chunks. These boundaries effectively force the wavefunction to drop to 0 at any distance past the edge of the chunk, although this itself is not important as at distances past the chunk's edge the next chunk is simply used instead. What is important is that during propagation, the wavefunction inside each chunk thinks the rest of the wavefunction is 0-valued - this causes 'reflections' at the boundaries, which cause inaccuracies at the edges of the chunks and can start to cause inaccuracies further and further within the chunk over time.

The figure below shows an example of this; where the wavefunction described by 5000 spatial points was split into two equally sized chunks, propagated seperately for 1000 timesteps, then recombined. It can be seen that there are large differences from the expected result at a couple of points at the centre, but elsewhere a very accurate result is obtained.

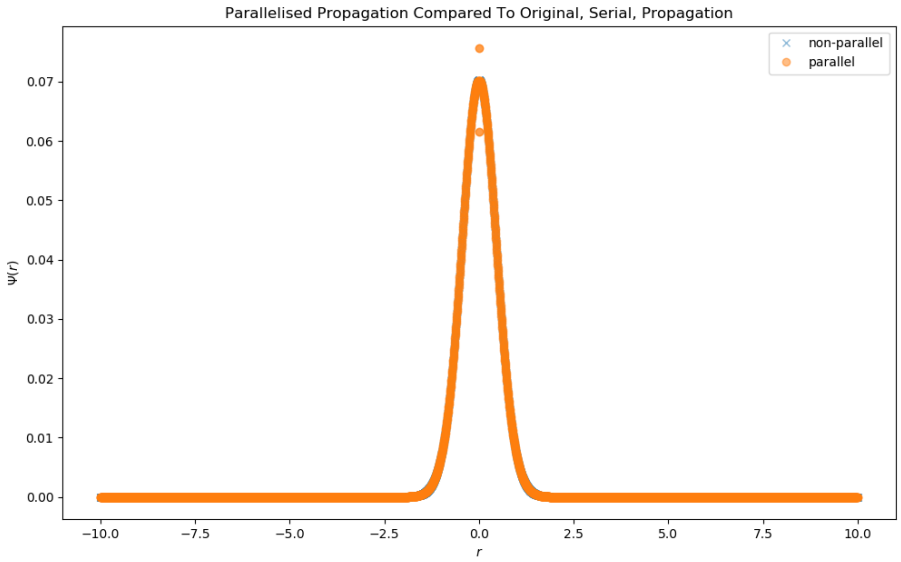

FIGURE 3.2: Comparison Between Evolved Wavefunction From Serial Propagation And Parallel Propagation

Note that these 'reflection' errors are limited to only a couple of grid points at the chunk edges, even after a relatively long (1000 timesteps) propagation - we use this knowledge in further improvements to increase accuracy of the parallelised propagators.

These 'reflections' should actually occur even for the non-parallelised case, except that in this case the assumption that the wavefunction is 0-valued beyond the edges is actually a good assumption and closely approximates the physical system. From the description of the cause of these reflections, it can be seen that the impact of these reflections depend on how far from 0 the wavefunction should be at the boundaries - for example, in the above figure we have that the chunk boundaries are at the peak of the wavefunction, and so this is a worst-case scenario. We now consider methods of mitigating the errors introduced by these reflections.

3.5.1 Avoiding Problems At Chunk Edges

As mentioned previously, there are several approaches to limiting or avoiding the errors introduced by reflections. One such method is to share knowledge of the values of a few spatial points of the chunked wavefunctions with their neighbouring chunks after every few timesteps; allowing the neighbours to re-adjust their wavefunction based on this knowledge - this is the approach used in RMT, which we will see in more detail in Chapter 5. A diagram describing this process is given below;

Limitations of this method include that requiring communication between chunks is costly, and limits the advantage of parallelisation as the subprocesses can no longer operate fully independently. For example, if the propagation was split across three computers, one of which was far slower than the other two and information about edge points was shared every 10 timesteps; although each computer would only need to perform one third of the total number of calculations, the two fast computers are only able to calculate up to 10 timesteps further before being stalled to wait for the third computer. If information did not need to be shared, the two fast computers could complete their calculations without needing to wait for the third.

This cost can be limited, by either ensuring all computers perform similarly or by varying the size of the chunks to match the performance of the comuter propagating them such that they can all propagate their chunk at around the same speed. The communication of edge information between computers itself also has an associated computational cost, which cannot be as easily limited.

Another method of avoiding problems at the edges of chunks is to 'overlap' chunks. If, for example, 10 extra spatial points are given to each chunk on either side - and knowing that after propagation the errors caused by

reflections mostly affect the outermost spatial points, then the next most outermost, and so on - then provided we haven't propagated far enough for the reflections to permeate too deeply into the wavefunction chunk the errors are almost entirely confined to the outer 10 spatial points. As these were 'extra' points we stuck on to each chunk, we can simply discard their values, then recombine the remainder of our chunks to form the full wavefunction with the errors caused by reflections almost entirely ignored.

The process described is explained in the diagram below;

In the event that we want to propagate further in time, the errors caused by reflections will come further inside the chunked wavefunction. We can avoid this by increasing the number of spatial points used in the overlap, although this will increase the computational cost of propagation.

Limitations of this method include that extra calculations (in the overlapped regions) need to be performed, increasing the overall computational cost, and these costs only increase with further time propagation meaning it does not scale well for large end times. However, relatively few overlap points are needed to mitigate most of the errors, and the there is an exponential decay on the impact of reflection errors as you move further from the edges - and so the rate of increase of the number of extra spatial points needed to propagate to the end time reduces as the end time grows.

An advantage this approach has over the last is that no communication is needed between the chunks; due to this there is no additional communication cost, and the wavefunction and Hamiltonian can be split into more chunks than there are computers - reducing the amount of work needed to propagate each chunk and allowing any faster computers to immediately start working on the next chunk upon completion of a chunk propagation, rather than waiting for the propagation of the other chunks to finish first.

3.5.2 Efficiency Testing

ARCHER (* CITE *) define several quantities; 'Speed-Up' (S), 'Parallel Efficiency' (E_{par}), and 'Serial Efficiency' (E_{ser}) which we will use to evaluate the effectiveness of our parallelisation. These quantities are defined in terms of execution time, T.

- **Speed-Up:**

$$S(N,P) = \frac{T(N,1)}{T(N,P)}$$

- **Parallel Efficiency:**

$$E_{\text{par}}(N,P) = \frac{S(N,P)}{P} = \frac{T(N,1)}{PT(N,P)}$$

- **Serial Efficiency:**

$$E_{\text{ser}}(N) = \frac{T_{\text{best}}(N)}{T(N,1)}$$

Where N, P are the size of the problem and number of processors respectively.

Amdahl (* cite archer *) stated in 1967 that "The performance improvement to be gained by parallelisation is limited by the proportion of the code which is serial". Effectively, this means that a given process can be approximated by a part or parts that is non-parallelisable and a part or parts that are; and so even with the best parallelisation scheme the process will take at least as long as the non-parallelisable parts take to run.

He exteds this concept into 'Amdahl's law'; given a process that has a fraction, α, that is completely serial and that the rest is a perfectly efficient parallel scheme, we can describe the parallel runtime and speedup as follows:

$$T(N,P) = \alpha T(N,1) + \frac{(1-\alpha)T(N,1)}{P}$$

$$S(N,P) = \frac{T(N,1)}{T(N,P)} = \frac{P}{\alpha P + (1-\alpha)}$$

From these, we can see that a parallel scheme will only be worthwhile if the serial part, α, is small.

To properly investigate the benefit of our parallel scheme, we need to investigate both the Parallel and Serial Efficiencies. This was performed on a server cluster, AIGIS (node aigis11), which has 32 available cores - each capable of running 2 threads and with a clock speed of 2.3GHz.

Parallel Efficiency

The parallel efficiency of our setup was investigated by timing the propagation of a wavefunction with 50,000 spatial grid points 10,000 timesteps, using between 1 and 32 cores. The following results were obtained;

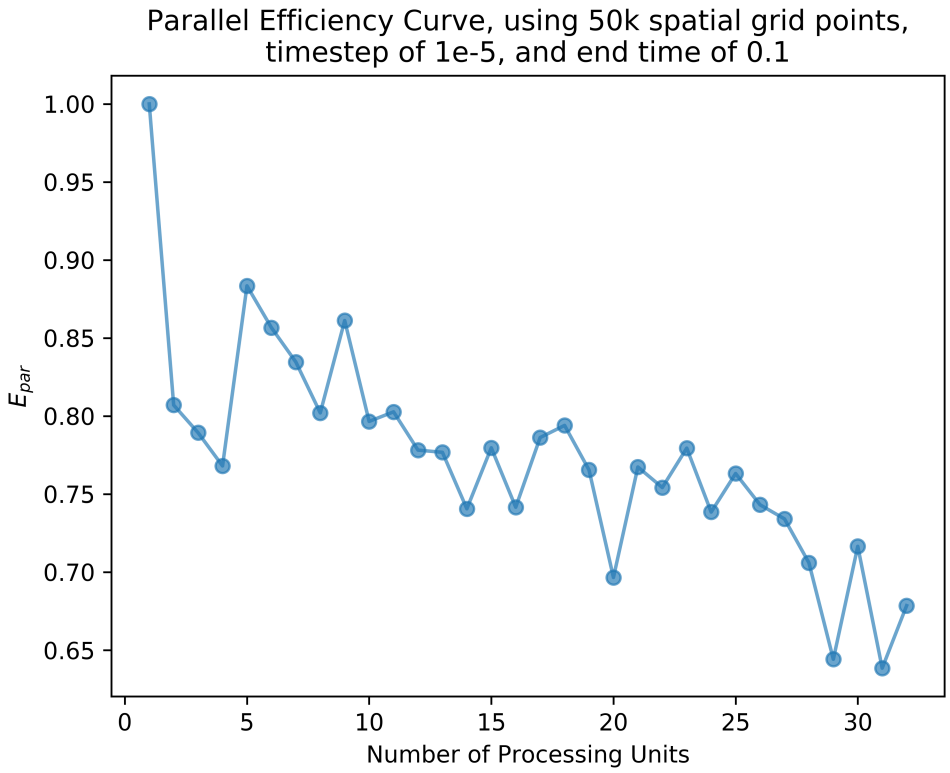

FIGURE 3.3: Parallel Efficiency Curve

Figure 3.2 shows that the rate of speedup for our scheme falls as the number of additional processors increases, although only a little; from the first additional processor to the thirty-first additional one, we only see an efficiency drop of around 0.2. So although there is a reducing amount of benefit from each additional processor, scheme still performs well on our server cluster - as it is limited to 32 cores, and our scheme has a reasonable benefit from each additional core even until all available cores are used here.

Serial Efficiency

The serial efficiency of our parallelised model was investigated by timing the propagation of a wavefunction with between 250 and 41,000 spatial grid points for 10,000 timesteps using both our 'best' parallelisation scheme and the non-parallelised scheme. I took the 'best' scheme to be the case where we use 31 cores, as it performed very similarly to when using the full 32 cores and as it was less likely to be interupted if anyone else started a job on the same node at the same time. The graph below describes the resuts;

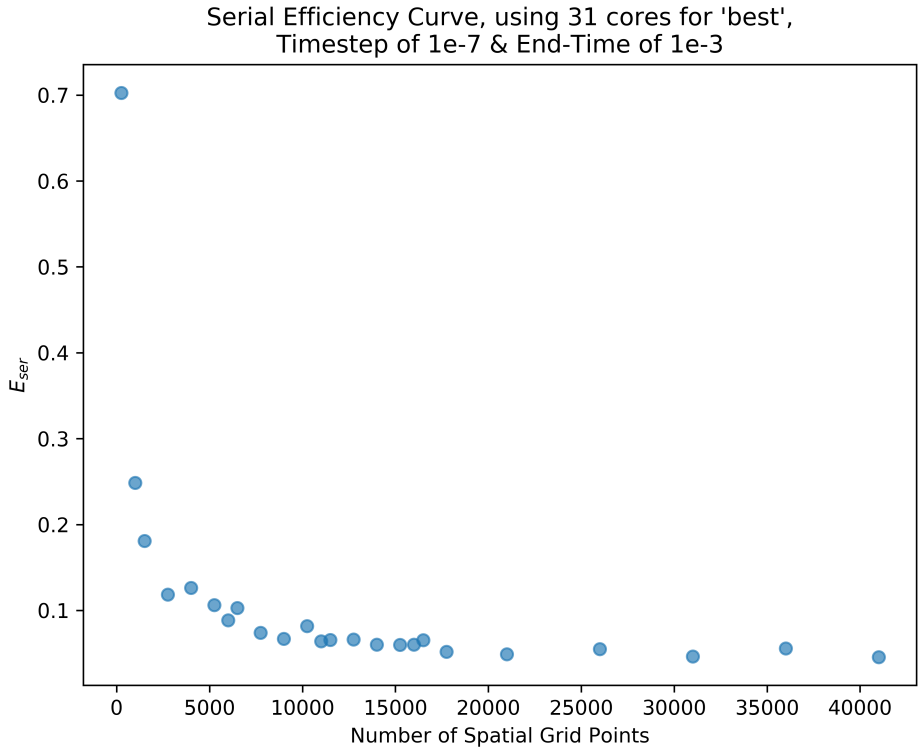

FIGURE 3.4: Serial Efficiency Curve

Figure 3.3 shows an increasing benefit ratio to using the parallelised scheme as the number of grid points grows, but this ratio converges at 17500 spatial grid points - after which the benefit ratio is fairly constant at around 0.04, showing that this scheme caps out at being around 25x faster than the non-parallelised version no matter how many additional processing units are given.

Comments

Chapter 4

TDSE Solutions for Time-Dependent Potentials

4.1 Qualitative Expectation for Results

Previously we saw that solutions to the TISE can be expressed as weighted sums of bound eigenstates, where each bound eigenstate corresponds to a distinct energy. It was also shown that if no external potential acts on the system these weights will not change as the wavefunction evolves with time; we used this fact to measure the accuracy of our propagators in the previous chapter.

The fact that these weights remain constant with no external potential being applied is equivalent to saying that the energy of the system remains constant under time evolution; this makes physical sense as a statement of the conservation of energy. When an external field is applied, the system will either gain or lose energy over time - and so these weights will change during time evolution.

As the weight / coefficient of each eigenstate corresponds to the probability of the system being in that state, by constructing appropriate time-dependent external potentials we can model how the probabilities of the system being in particular eigenstates changes in the event of some physical interactions; for example absorbing a laser pulse.

The rest of this chapter consists of investigating how the system, initially in the Ground State, evolves under the influence of different simulated interactions - by investigating the change to the Ground State population under the interactions over time.

4.2 Linearly Increasing Potential Field

Here we investigate the evolution of the system under a linearly growing (w.r.t. time) potential field.

4.2.1 Potential Description

We simulate the external potential term as $\alpha t\mathbf{r}$; where $\mathbf{r}$ is the spatial interval, t is the time, and α is a constant to determine the rate of increase of intensity of the potential.

This extra potential term was then added to the softcore potential, and the resulting total potential used along the central diagonal of the Hamiltonian.

4.2.2 Effect On State Evolution

Figure 4.1, below, shows the effect of the linearly increasing potential on the wavefunction (initially Ground State) over time.

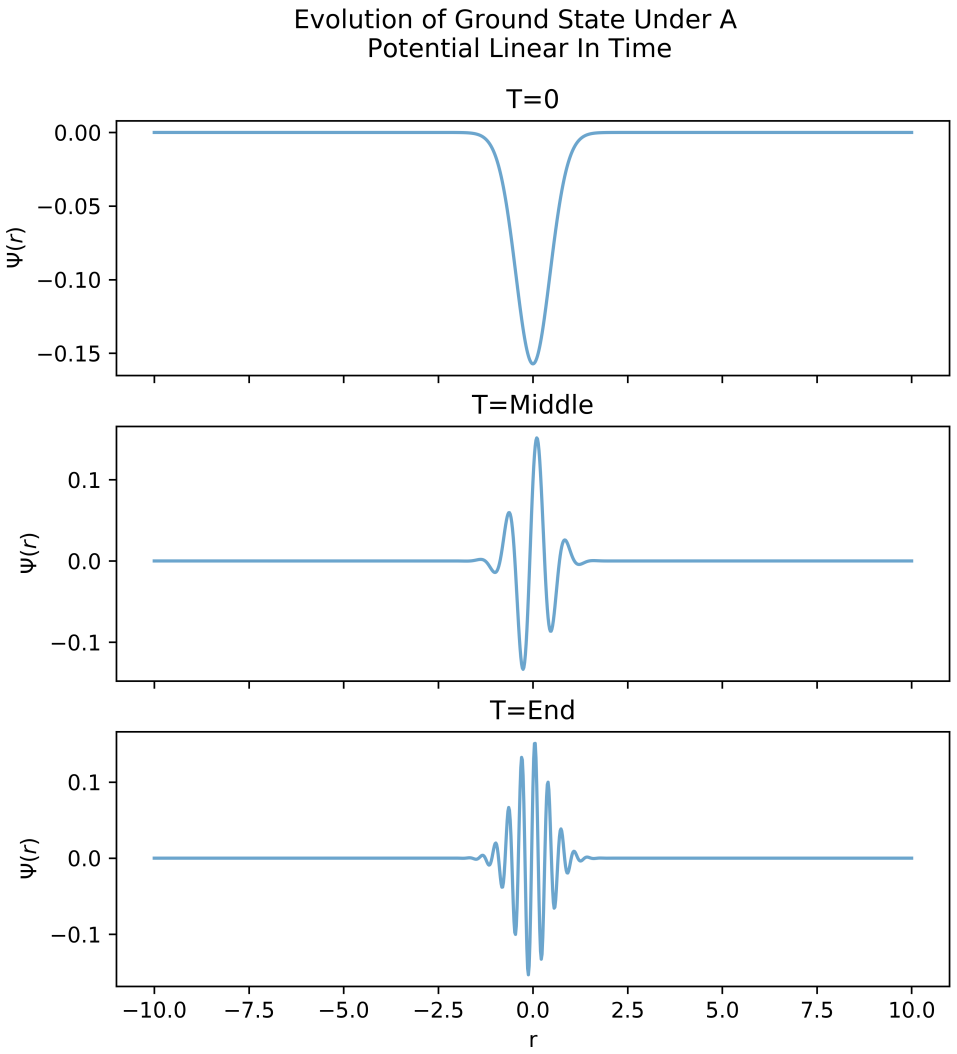

FIGURE 4.1: Effect of a linearly growing potential on wavefunction

From this figure, we see qualitatively that as time goes on the system is excited to higher and higher energies; shown through the increased numbers of turning points, and their relative magnitudes, indicating increasing contributions from higher energy eigenstates. Based on this, we can expect that the Ground State population will reduce over time as more and more eigenstates contribute and their contributions gain more and more weight. Figure 4.2, below, shows the decrease in the Ground State population over the same time interval:

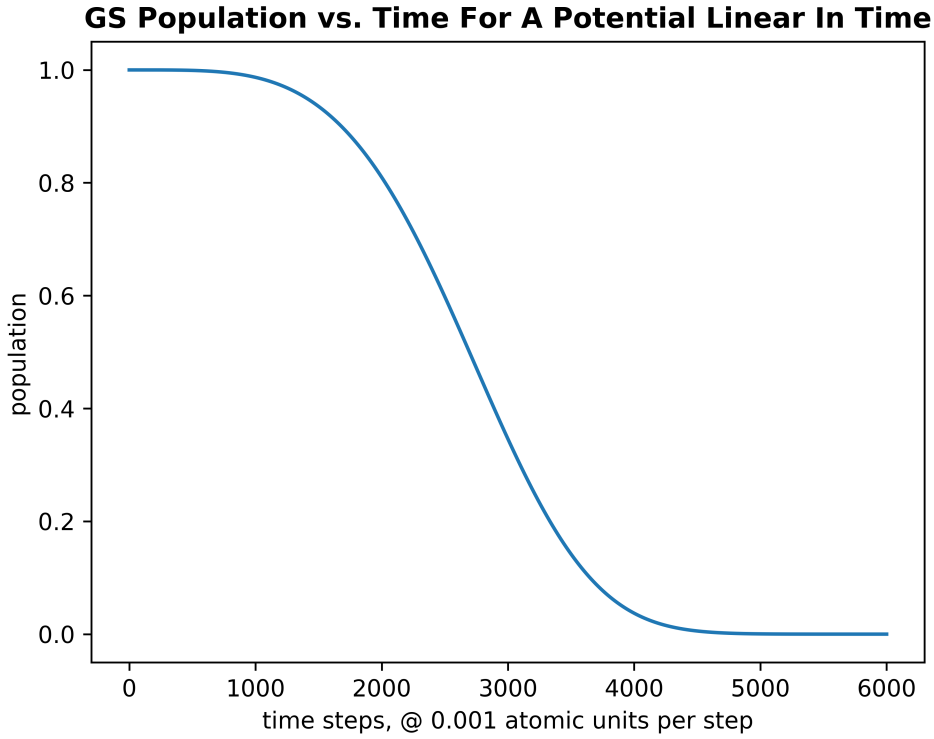

FIGURE 4.2: Effect of a linearly growing potential on Ground State Population

4.3 Gaussian Packet Potential

In this section we investigate how our system would behave under the influence of a gaussian (with time) potential.

4.3.1 Potential Description

In this case, we build the external potential term as a gaussian packet w.r.t. time such that the centre of the packet (most intense part) occurs at 1 atomic unit of time, and the standard deviation of the packet is 0.25 atomic units of time. The intensity of the potential is controlled by a multiplicative factor, α, giving the following expression for the external potential:

$$V_{\text{ext}} = \alpha \frac{4}{\sqrt{2\pi}} e^{-8(t-1)^2} \mathbf{r}$$

As with the previous example, this extra potential term was added to the softcore potential, and the result used along the central diagonal of the Hamiltonian.

4.3.2 Effect On State Evolution

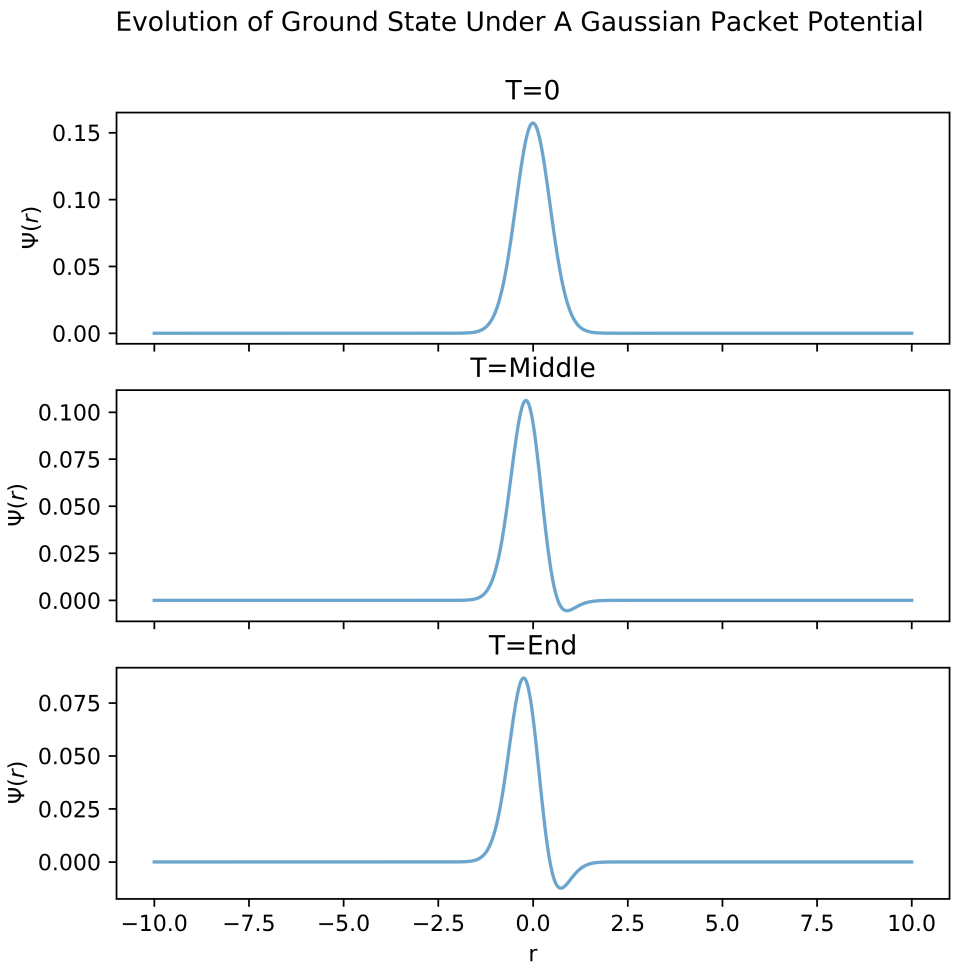

FIGURE 4.3: Effect of Single Gaussian Packet on wavefunction

Figure 4.3, above, we see that almost all of a relatively small amount of energy is absorbed some time between the start and mid-point of the simulation (an interval of 3 atomic units of time), seen through the small additional turning point indicating a small contribution from the first excited state. There is very little change in shape between the mid-point and end of the simulation, indicating that there was very little incoming / outgoing energy in this time period.

The change to Ground State population was investigated over the same time interval, and the results shown in Figure 4.4, below:

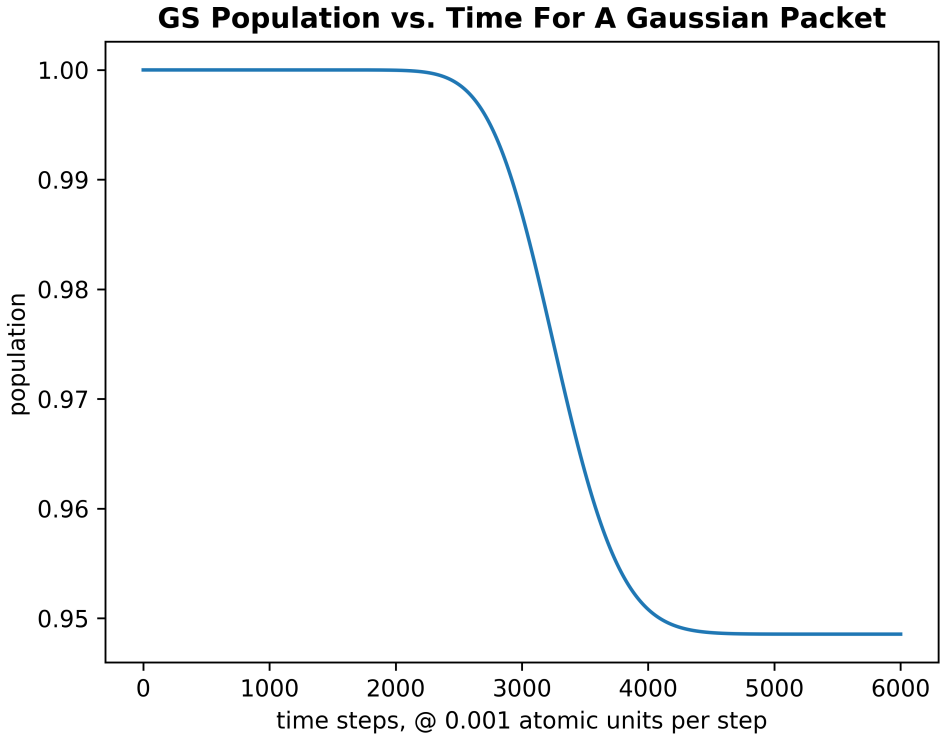

FIGURE 4.4: Effect of Gaussian Packet on GS population

This graph shows that between around 0.5 and 1.5 atomic units of time into the simulation, the wavefunction is rapidly excited; losing 5% of the population by 1.5 atomic units of time. Adter this initial loss, the population stays steadily at around 95% for the rest of the simulation, indicating no further change to its energy.

4.4 Multiple Gaussian Packets

We investigate the effects on our system of subjecting it to a series of gaussian potentials over the course of the simulation.

4.4.1 Potential Description

The external potential used in this simulation is similar to the one used in the previous simulation, but with three added to the softcore potential to form the total potential term rather than one. Additionally, two of the packets have their centres shifted to occur at 3 and 5 atomic units of time.

4.4.2 Effect On State Evolution

The graphs in the figure below describe the evolution of the system as it absorbs energy from repeated gaussian potentials. This differs from the previous investigation of a single packet in that the end result is a system with noticeably higher contributions from excited states than previously, indicating that the system has absorbed more energy. Additionally, the wavefunction of the system at the end shows higher exctited state contributions than in the middle of the simulation - indicating that energy is absorbed both before and after the mid-point of the simulation, matching expectations as the packets were staggered to be centred around $\frac{1}{6}$, $\frac{3}{6}$, and $\frac{5}{6}$ of the simulation.

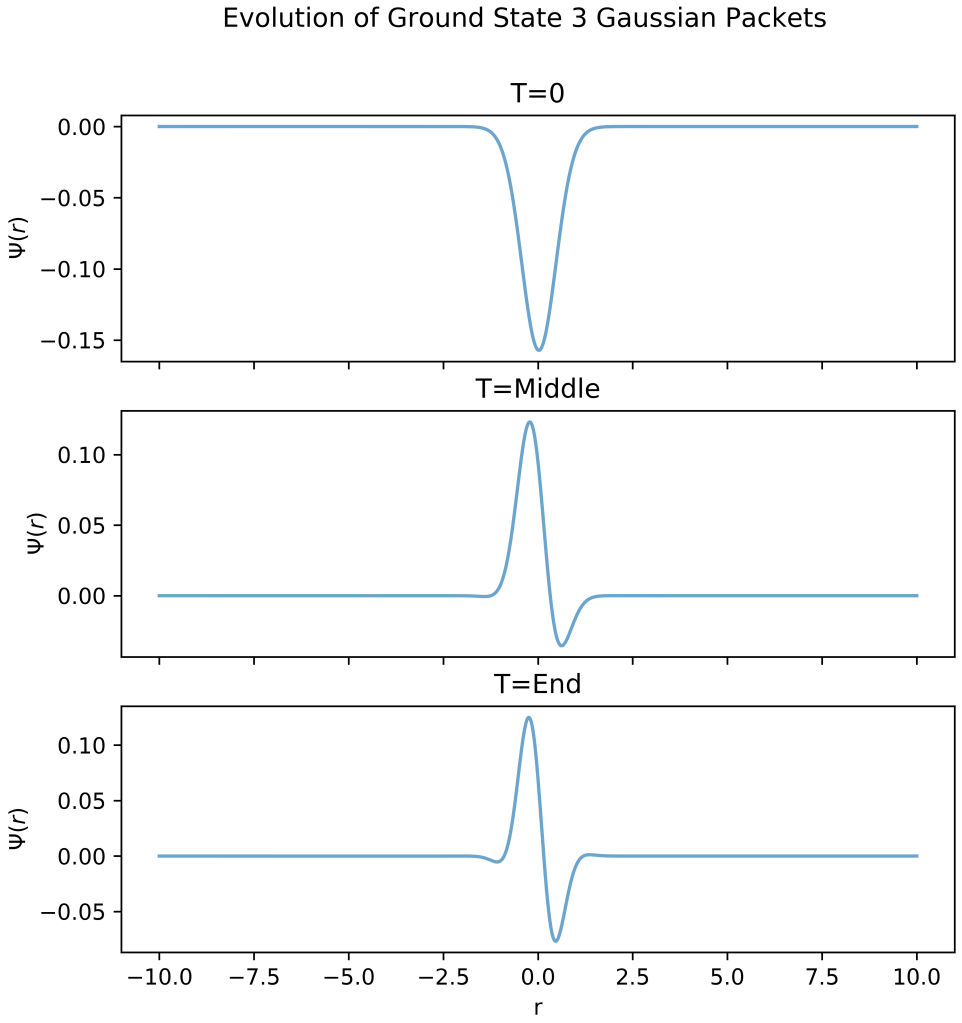

FIGURE 4.5: Effect of 3 Gaussian Packets on wavefunction

The Ground State population was also investigated, with the results shown in Figure 4.6 below, and we can see that there are three distinct drops in

population - centred around 1, 3, and 5 atomic units of time; matching when we placed the maxima of the laser pulses. Additionally, we can see that the repeated laser pulses led to a higher overall drop in Ground State population by the end of the simulation, matching the analysis of the evolution of the wavefunction where we noticed higher contributions from excited states than with the single laser pulse.

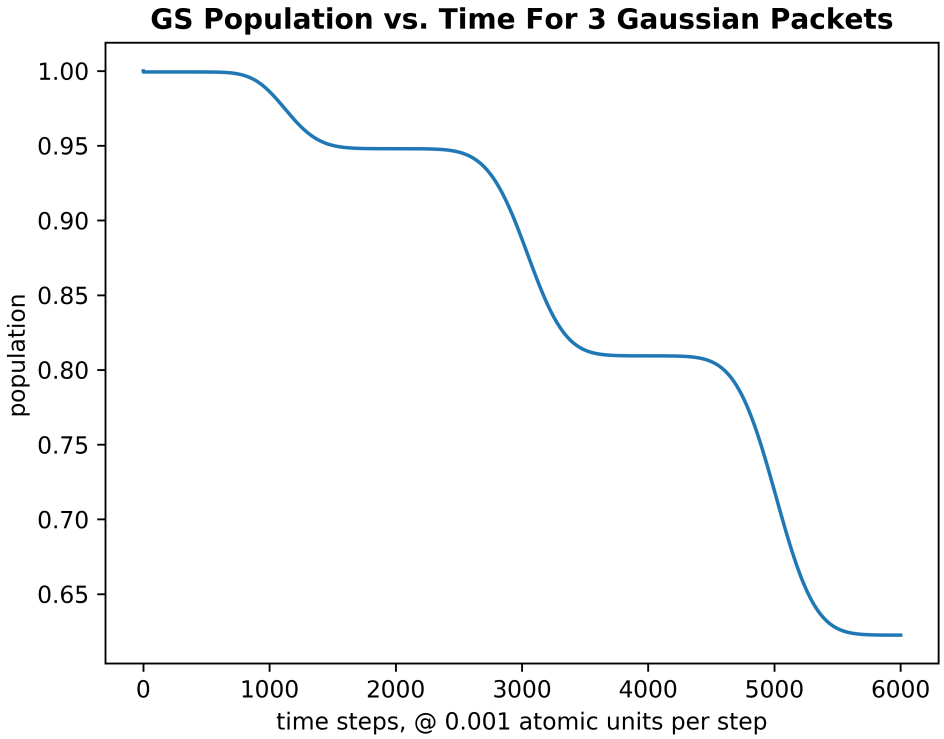

FIGURE 4.6: Effect of 3 Gaussian Packets on GS population

4.5 Simulated Laser Pulse

Here we model a Laser pulse interacting with our system using a potential built by encapsulating a sinusoidal wave within a gaussian packet to model our Laser pulse. Mathematically, this potential is described by;

$$V_{\text{ext}} = \alpha \frac{2}{\sqrt{2\pi}} e^{-2(t-3)^2} \mathbf{r} \sin(\omega t),$$

where ω is the frequency of the sinusoidal term and α is a multiplicative factor controlling the intensity.

The figure below describes the field strength of this potential over the course of the simulation, and qualitatively demonstrates its similarity to a laser pulse potential:

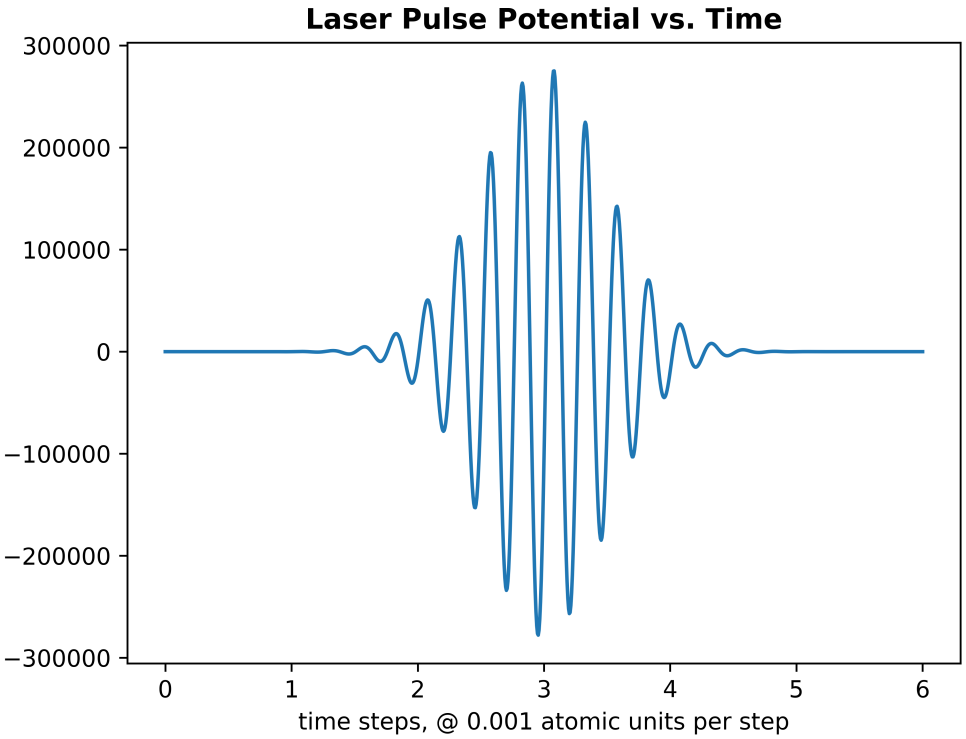

FIGURE 4.7: Potential Field Strength Over Time For Laser Pulse

Upon interacting with our system, rapid gains and losses of energy are made as the direction of the potential cycles. The following figure demonstrates this effect;

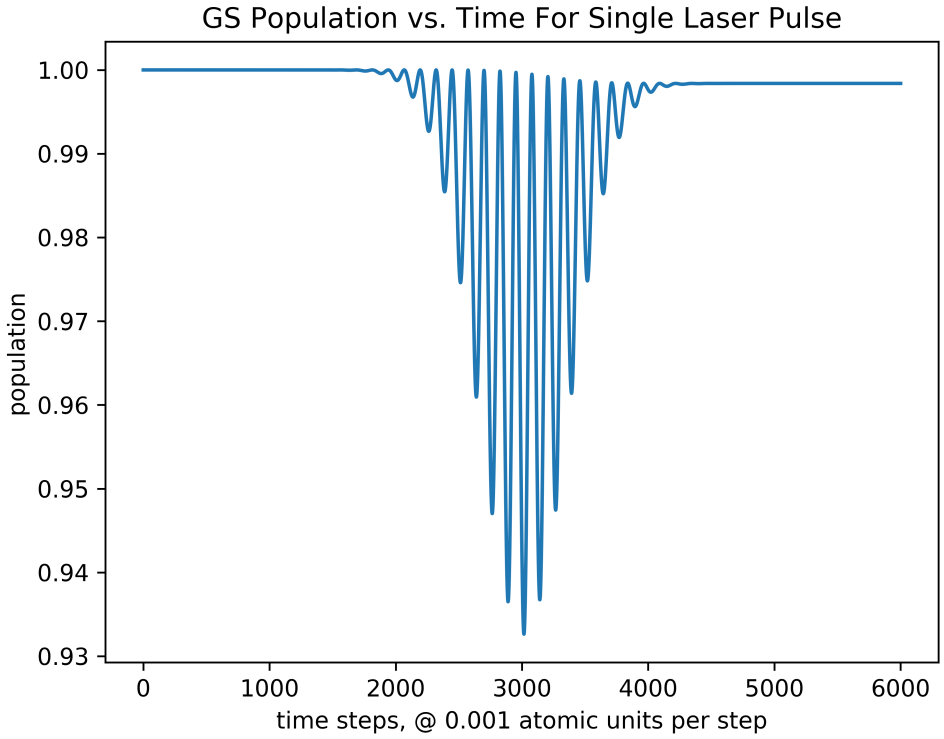

FIGURE 4.8: Effect of Simulated Laser Pulse on GS population

Here we see that the wavefunction, initially entirely in the ground state, continuously gains and loses population in higher excited states as the pulse interacts - with the magnitudes of these gains and losses scaling with the magnitude of the gaussian packet enveloping the pulse. Additionally we see that almost, but not all of, the population is retained at the end of the pulse; leaving a small amount ($\simeq 1\%$) of the population in higher excited states.

Chapter 5

RMT

5.1 R-Matrix Theory

5.1.1 Origins & Usage

(* cite "https://arxiv.org/pdf/1001.0678.pdf" P. Descouvemont, D. Baye *)
R-Matrix theory is a method, or branch of methods, to obtain scattering properties from the Schrodinger equation of a range of physical systems (* cite "https://arxiv.org/pdf/1001.0 P. Descouvemont, D. Baye *).

It was first proposed in 1946 by Wigner(* cite Wigner E P 1946, Phys. Rev.7015 *) as a tool for modelling in quantum physics, initially intended to be used to describe resonances in nuclear reactions. In modern usage, R-Matrix theory is mostly used to describe scattering states resulting from interactions of particles or systems of particles.

Adaptions to, and improvements on, the original R-Matrix method have been developed to extend the use cases for it; one example of this is 'Time-dependent R-matrix' theory, where an R-Matrix basis set describing the wavefunction of the system is evolved through time.

5.1.2 Description of R-Matrix Method

The central idea of the R-Matrix method is to consider the system being modelled as consisting of two regions of space; a large 'outer' region, where only long-range interactions are considered and spherical symmetery can be assumed, surrounding a smaller 'inner' region, where additional short-range interactions are considered and any antisymmetrizational effects must be accounted for.

The boundary between these two regions is described by the 'channel radius', chosen such that the stated assumptions for the two regions hold true, and allows calculation of the 'R-Matrix' - the inverse of the logarithmic

derivative of the inner-region wavefunction at the boundary. Comparing this with the external-region solution allows calculation of the scattering matrix, and allows bound states to be obtained.

5.2 Overview Of RMT

5.2.1 What Is RMT?

RMT is a toolset of codes based on [R]-[M]atrix theory with [T]ime-dependence and is used for investigating atomic, molecular, and optical physics.

It was first proposed(* cite L. R. Moore. The RMT method for many-electron atomic systems in intense short-pulse laser light *) in 2011 as an ab-initio method for solving the TDSE for multi-electron systems exposed to intense short-pulse laser light, and was developed in the CTAMOP department of Queen's University Belfast; where further development continues to take place, allowing for the investigation of other phyical phenomena such as high-harmonic generation, polarisation effects, and even some relativistic effects.

In addition to development on the RMT toolset to allow invesigation of new physics, development aimed towards improving the computational efficiency is often undertaken as the cost of running simulations can be prohibitive for some investigations and reduce the scope in which RMT can be a useful toolset.

5.3 Wavefunction Propagation In RMT

5.3.1 Overview of Arnoldi Propagator

At its current version, RMT uses Arnoldi-based propagators in both the inner and outer regions - calling the subroutines 'arnoldi_propagate_inner' & 'arnoldi_propagate_outer respectively to perform each step of this propagation. These propagators work as we have described in Chapter 3, and the Krylov basis size to be used can be specified in the parameter setup file. While RMT parallelises the evolution of the wavefunction across many processing units, the parallelisation steps are abstracted outside these subroutines such that within each subroutine call the process is completely serial - a fact I later exploit in order to simplify the work needed to implement finite difference propagation in RMT without needing to reimplement a parallelisation scheme.

5.3.2 Information Sharing Between Regions

During propagation, the FD points of the wavefunciton near the boundary in the inner region are projected onto the basis set for the outer region wavefunction, and then communicated to the outer region, in order to account for boundary reflections. A similar communication interface exists to communicate information about the outer region to the inner region during propagation. This communication represents a sizeable chunk of the computational effort needed to perform the propagation, which is mitigated in RMT by not sharing this information between the regions at every timestep; instead only every few timesteps.

5.3.3 Information Sharing At Spatial Chunk Boundaries

In addition to information sharing between the regions, the parallelisation scheme used to propagate the wavefunctions splits the propagation of chunks of the spatial intervals across several processing units uses a different method to account for boundary reflections than used in the parallelisation scheme outlined in Chapter 3, and requires information sharing between processing nodes.

In the scheme I implemented in my 'toy code' implementations additional, redundant, points are used in each spatial chunk to contain the errors introduced by reflections at boundaries to outside the actual spatial split; allowing the redundant points to simply be discarded after propagation, and with them almost all of the errors. A different implementation is used in RMT; similarly to the information sharing between the inner region and outer region at the boundary, information about the edges of each spatial chunk is shared between neighbouring chunks during propagation; this allows for a more accurate end result than the method I used, and reduces redundant computational cost in the propagation of each chunk at the cost of a higher computational cost in communication. As the number of redundant points needed to contain the reflection errors increases as propagation continues, this scheme makes sense for the majority of the use cases of RMT - as longer propagation, and smaller timesteps, are usually required.

5.3.4 Insights

As noted, the parallelisation implementation of the propagation in RMT is abstracted outside the scope of the two arnoldi-propagation subroutines; meaning that a different propagation method can be implemented while keeping the

original parallelisation scheme; reducing the amount of development needed to implement an alternative method.

5.4 Finite Difference RMT Implementation

5.4.1 Overview Of Potential Inefficiency

As mentioned in Chapter 4 in the comparison between the two propators I built for my 'toy code', Arnoldi-based propagation requires multiple large matrix-vector multiplications while FD propagation only requires a single matrix-vector multiplication - and these multiplications represent the bulk of the work needed for each propagation. Having shown in my simplified implementations that FD propagation allows cheap increases to propagation order, and that in general FD propagator efficiency outperforms Arnoldi propagator efficiency, it seemed worth investigating a FD-propagation implementation of RMT - where the main considerations would be in ensuring stability of the wavefunction, and efficiency compared to the existing Arnoldi-propagation.

5.4.2 Method Of Attempted Improvement

During the course of this project, I worked towards a FD RMT implementation; I did not manage to complete this goal to a fully working state within the timeframe, but I did manage to implement a FD based alternative to the 'arnoldi_propagate_inner' subroutine and test it produced correct results for short timescales (such that the wavefunction was contained within the inner region).

Additionally I made progress toward a FD alternative to 'arnoldi_propagate_outer' but, as of publication of this thesis, this work is incomplete.

To implement the FD propagation in the inner region, the first challenge faced was to build up the initial set of 'previous wavefunctions' to be used in the propagation - this was not needed in the Arnoldi-propagation subroutine. To do this, instead of using the analytic solutions to the field-free case to propagate the first few timesteps and assume no field is 'activated' until a few timesteps into the simulation, I instead used the analytic solutions to propagate the initial wavefunction backwards in time - i.e. to find it at timesteps corresponding to before the start of the simulation. This removed the need to ensure there are no features in the field near the start of the

simulation, or to change any variables or parameters controlling what 'time' each part of the program believes it to be at any given instant in the simulation.

The next challenge was to represent the Hamiltonian as a matrix with all effects and phenomena considered, then efficiently evaluate its matrix product with the current wavefunction. Luckily, this step is also needed in Arnoldi propagation; and so I was able to simply use an existing parallelised subroutine to efficiently do this step - 'parallel_matrix_vector_multiply_zm'

With these two challenges taken care of, the remainder of the propagation method closely resembled the method used in my toy code - simply subtracting previous wavefunctions multiplied by finite difference coefficients. Given this, I was able to copy my previous work (albeit needing to convert it to a different programming language), and then ensure that the correct global variables corresponding to the chunks of wavefunction are correctly updated and shared between timesteps.

5.4.3 Analysis Of Effectiveness & Usefulness Of Completed Work

dsfg kjd klds lk sd klndf lk s sd dfg sdfgdsfgsd

Chapter 6

Conclusions

6.1 Review of Project Goals

6.2 Overview of Findings

6.2.1 TISE

adslkjn

6.2.2 TDSE

asdf

6.2.3 RMT

kdadfkj

6.3 Potential Further Research Directions

6.3.1 Complete FD RMT Outer Region Propagator

kjna kma

6.3.2 Add Functionality To FD RMT Implementation

fs hsdf

6.3.3 Investigate 'Non-Finite' Difference Model

asdbb

Appendix A

Frequently Asked Questions

A.1 How do I change the colors of links?

The color of links can be changed to your liking using:

`\hypersetup{urlcolor=red}`, or

`\hypersetup{citecolor=green}`, or

`\hypersetup{allcolor=blue}`.

If you want to completely hide the links, you can use:

`\hypersetup{allcolors=.}`, or even better:

`\hypersetup{hidelinks}`.

If you want to have obvious links in the PDF but not the printed text, use:

`\hypersetup{colorlinks=false}`.

Bibliography

[1] Roman Trobec Boštjan Slivnik Patricio Bulić Borut Robič. *Introduction To Parallel Computing*. Springer Nature Switzerland, 2018. URL: https://doi.org/10.1007/978-3-319-98833-7.