State-dependent forces in cold quantum gases

Christopher Billington

Submitted in total fulfilment of the requirements
of the degree of Doctor of Philosophy

Supervisory committee:
Prof Kristian Helmerson
Dr Lincoln Turner
Dr Russell Anderson

School of Physics and Astronomy
Monash University

June, 2018

Copyright Notices

Abstract

Declaration

This thesis contains no material that has been accepted for the award of any other degree or diploma in any university or other institution. To the best of my knowledge the thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis. For parts of this thesis that are based on joint research or publications, the relative contributions of the respective authors are detailed appropriately.

Christopher Billington

Professor Kristian Helmerson

Dr Lincoln Turner

Dr Russell Anderson

Acknowledgements

Contents

Chapter I

Introduction

The subject of study of this thesis is Bose–Einstein condensation, as well as associated experimental and theoretical techniques and phenomena in cold atom physics. The following chapters describe my work in a cold atom research group over the past several years, pertaining to the design and modelling of new experimental techniques, the development of modelling methods for more effectively simulating cold-atom systems, and the design and development of laboratory software for control and analysis of experiments on cold atoms. An overarching theme is *state-dependent forces* on cold atoms. Selectively subjecting atoms to forces based on what state they are in is at the core of many phenomena in cold atom physics. As I go into in the following chapters, different types of state selectivity allow for cooling and imaging techniques that would otherwise not be possible, and semiclassical models run into a problem when state-selective forces cannot be disregarded in determining the classical force that atoms modelled semiclassically ought to be subjected to.

Bose–Einstein condensates (BECs) in dilute atomic gases are superfluids that can be created in the lab at extremely low temperatures. This strange state of matter was predicted in 1925 by Einstein [1] based on the work of Bose [2], and first produced experimentally in 1995 [3] in a cloud of rubidium atoms, and has since been made out of many other atoms, usually alkali metals [4–7]. In a BEC, a macroscopic sample of bosonic atoms all occupy the same quantum state, and many of the features of the single particle wavefunctions are exhibited by the cloud as a whole. Bose–Einstein condensation and cold atoms and ions more generally have rich applications in precision measurement [8, 9], quantum computation [10, 11] and quantum simulation [12, 13].

Various experimental techniques are used to produce and study Bose–Einstein condensates, many of which exploit or necessitate an understanding of the quantum behaviour of the atomic systems in question. I summarise some of these techniques and detail the atomic physics principles underlying them in Chapter 2.

The fields of Bose–Einstein condensation and cold atoms more generally enjoy a tight coupling between theory and experiment, not least because of the enduring usefulness and accuracy of mean-field theory. In mean-field theory, the quantum matter field operator of the atoms comprising a Bose–Einstein condensate is replaced with its expectation value at each point in space, allowing the entire multi-particle system to be modelled with little more computational complexity than that required to model a single-particle wavefunction.[1] The resulting differential equation—the Gross–Pitaevskii equation—is nonlinear and using it to propagate a condensate wavefunction in time generally requires numerical techniques rather than analytic ones.

Chapter 3 is a pedagogical presentation of how time-dependent quantum mechanics is simulated on a computer. I lay out the path one takes from differential equations

[1] Mean-field theory is accurate in the low-temperature limit, in which it has some remaining limitations—it does not for example predict the observed *s*-wave scattering halos when two BEC wavepackets collide [14], but it is sufficient for modelling a wide range of experiments nonetheless.

containing abstract quantum-mechanical kets, to concrete arrays of numbers appropriate for a computer to perform calculations on. I give a detailed quantitative appraisal of a range of different techniques, their uses in cold atom physics and Bose–Einstein condensation, and compare their relative efficacies. I present arguments that a fairly sophisticated method of discretising partial differential equations—the finite-element discrete-variable representation—may offer less computational efficiency than simpler methods for computing solutions of comparable accuracy to the Gross–Pitaevskii and Schrödinger wave equations. I also develop a variation on fourth-order Runge–Kutta integration that improves on one of its deficiencies for simulating quantum systems.

Production, control, and measurement of cold atom systems require more than the necessary optics and magnetic sources to be installed—these devices must be controllable in a time-accurate way in order to execute the necessary cooling processes, manipulate the system as desired, and observe the results. Production of a condensate takes on the order of tens of seconds, requiring precisely-timed pulses of laser light at specific frequencies, sweeps of magnetic field strengths, and frequency sweeps of radio and microwave radiation. This cannot all be done by human experimenters alone, and so requires computer automation of some kind. In Chapter 4 I a discuss a suite of software programs, the *labscript suite*, designed and developed by myself and others in the Monash Quantum Fluids group. The suite leverages modern software development techniques such as object orientation, abstraction and isolation, as well as older principles—such as aspects of the Unix philosophy—to produce a powerful, maintainable, extensible system for designing, running and analysing shot-based experiments on commodity hardware. This chapter also reproduces our publication on the labscript suite, *A scripted control system for autonomous hardware-timed experiments* [15].

As superfluids, BECs have zero viscosity and as such can support persistent flows. In classical fluid dynamics the absence of viscosity means that a fluid cannot support vorticity,[2] and must be irrotational. However, fluid circulation can still occur around points of zero fluid density, known as vortices. In BECs this circulation is also quantised, in units of h/m.

These quantised vortices are topological defects—the phase of the macroscopic wavefunction winds by a multiple of 2π around them, and is undefined at the centre of the vortex core itself. Quantised vortices were observed in superfluid helium[3] in the early 1960s [16], and in BEC in a dilute atomic gas in 1999 [17]. The formation, dynamics and decay of these vortices are believed to be important for the study of superfluid turbulence [18].

In Chapter 5 I present simulations exploring the feasibility of imaging these vortices in-situ using *tracer particles*. Atoms of one kind (^{87}Rb) may become trapped in the cores of quantised vortices in a condensate of another kind (^{41}K), and if imaged in a time-resolved way, reveal the motion of these vortices. A primary concern in any implementation of such a scheme is keeping the tracer atoms cold enough that they remain trapped in the vortex cores even as they scatter light for imaging. To that end, in Chapter 5 I present modelling of imaging tracer atoms in a BEC while they are cooled sympathetically via collisions with condensed atoms. I also present a novel—if impractical—laser cooling scheme for Sisyphus cooling of ^{87}Rb atoms in a 34 G magnetic field—a field strength at which ^{87}Rb and ^{41}K repel each other strongly (leading to tighter trapping in the vortex cores).

At high temperatures (higher than that at which atoms Bose-condense) atoms are well described as classical particles. This is true in the sense that the wavelike nature of the atoms can be disregarded—they move through space like classical billiard balls obeying Newtonian mechanics. The internal state of the atoms, however—for example the state of an outer shell electron—may not be well modelled by classical mechanics. Even at room temperature, an electron is poorly described as a classical charged particle orbiting a nucleus. When there is *coupling* then, between this internal state of an atom

[2] This is because the motion of vorticity is described by a diffusion equation—with viscosity as the diffusion constant. When the diffusion constant is zero, there is no way for vorticity to enter the fluid from a boundary in the first place!

[3] In which 10% or so of the atoms undergo Bose–Einstein condensation.

and its motional state, the quantum-ness of the internal state can in some sense 'leak' into its motional state even if the motion is otherwise modelled well classically. The classic example of this is the Stern–Gerlach experiment [19], in which a beam of atoms splits into two beams as it passes through a magnetic field gradient. A similar situation arises for atoms in a magnetic trap—a common feature of cold atom experiments and often used in the final stage of cooling to BEC. To correctly model the losses of atoms from these traps, one needs to model the internal state of the atoms quantum-mechanically, but it is computationally expensive to also model their spatial motion using full quantum wavefunctions. We would like a way to model the atoms' motion classically, but in such a way that it can reproduce Stern–Gerlach separation—with modelled atoms taking one or the other trajectory probabilistically, with the probabilities consistent with those of a fully quantum treatment. In Chapter 6 I present such a model, one that is based on a *hidden variable* carried around with each atom being modelled, which selects one of the atom's internal eigenstates. The apparent definiteness of the hidden variable allows the spatial motion part of the modelling to treat the atoms' spin projection degree of freedom as if it were in a definite state, allowing the modelling to take a single, definite trajectory. The hidden variable itself is evolved using a stochastic hidden-variable theory that ensures its probability of corresponding to any particular spin-projection state is consistent with the underlying quantum evolution of the atom's internal degrees of freedom.

Further background and introductory material is presented in the separate chapters of this thesis.

1.1 What's new in this thesis

The following two results comprise the primary scientific contributions of this thesis:

The use of hidden variables in semiclassical models of atomic dynamics was motivated by simulating evaporative cooling in a quadrupole magnetic trap. During a collaboration with Drs Turner and Anderson and Chris Watkins on this topic, we had the sobering realisation that a bedrock experiment of quantum mechanics—the Stern–Gerlach experiment—was not simulable with oft-used semiclassical methods, despite the motional degree of freedom being irrefutably classical. I developed the hidden variable semiclassical method independently; only during the preparation of this thesis did I discover that the core idea mirrors that of surface hopping in quantum chemistry [20]. This positioned me to identify these 'hopping algorithms' with hidden-variable theories [21], and elucidate unique aspects of my implementation.

The design and implementation of the *labscript suite* [15] advances the state of the art of laboratory control and on-line data analysis by importing powerful principles and techniques from the field of software engineering for use in not only the laboratory control software, but the experiments themselves. The realisation that physics experiments conceptually map well to computer programs, with modularity, re-use, input parameters and return values allowed us to apply solutions typical in software development to manage the complexity of a scientific experiment without limiting the power of the control system to execute arbitrarily complex experiments. Driven by ideas advanced by Scott Owen and David Hall [22] and discussion within the Monash Quantum Fluids group with Drs Turner and Anderson, fellow PhD students Philip Starkey, Shaun Johnstone, and others; I extended the idea of writing experiments as code to the powerful yet simple to learn Python programming language, ideal for a physics laboratory in which new students must learn to operate the experiment effectively. This 'experiment compiler', called `labscript`, is the core of the software suite we now call the labscript suite, which has been developed over the course of my PhD by myself, Philip Starkey and others, and now includes a number of separate programs for setting input parameters, executing the experiment on heterogeneous hardware, and running user-provided analysis routines on

the results, among other things. The software has been adopted for use by a number of groups at leading institutions worldwide, and as an open-source project has attracted an increasing number of third-party contributions that enhance its functionality or resolve issues. This open-source model, with code not only being available to end users but with development occurring in the open on the internet has proved to be beneficial for the long-term sustainability of the project, which is now seven years old and receives continuous bugfixes and updates to ensure it continues to benefit the experimental physics community.

Less significant but nevertheless noteworthy original contributions of this thesis include:

Simulations of tracer particles of one atomic species trapped in vortices in a Bose–Einstein condensate of another species, to asses the viability of the use of tracer particles in a non-destructive vortex imaging scheme.

Numerical results of a new Sisyphus-like laser cooling scheme able to cool to sub-Doppler temperatures in a 34 G magnetic field, of interest due to a Feshbach resonance between ^{87}Rb and ^{41}K at this field strength.

A modification to the fourth-order Runge–Kutta method for time evolving differential equations in the presence of large energy differences, in addition to original analyses and appraisals of existing numerical methods contrasting their relative merits in the context of Bose–Einstein condensation quantum mechanics more generally.

Chapter 2

Atomic physics: Experimental techniques and theory

Bose–Einstein condensates provide such a tantalising opportunity for studying quantum phenomena not only because of their interesting properties, but also because of the level of control they afford, with parameters able to be tuned and manipulated in order to investigate various regimes. Many of the same techniques which allow experimentalists such control over a BEC are also employed in the production of BEC.

The main experimental techniques used to create BECs are Doppler cooling, magneto-optical, magnetic, and dipole trapping, polarisation gradient (Sisyphus) cooling, and evaporative cooling. Section 2.1 is a whirlwind summary of these and a few other experimental techniques. Then in Section 2.2, I'll present some of the theory describing superfluid flow and vortices and in BEC, which is central to the simulations of vortex tracking presented in Chapter 5. The final section of this chapter, Section 2.3, will construct the detailed theory for describing the internal state of a ^{87}Rb atom in magnetic and optical fields, considering only the first $L = 0$ to $L = 1$ excitation is accessible to rubidium's sole outer-shell electron. The resulting Hamiltonian describes 32 sublevels, and forms the basis of any detailed calculations of laser cooling and trapping.

2.1 Cooling, trapping, and manipulating atoms

2.1.1 Doppler cooling

Doppler cooling, demonstrated in 1978 [23] is a consequence of the simple observation that atoms see the wavelength of incident light Doppler shifted depending on their velocity. For example, the electric field of a linearly polarised optical plane wave is:

$$\boldsymbol{E} \propto \cos(\boldsymbol{k}\cdot\boldsymbol{r} - \omega t), \tag{2.1}$$

which for an atom moving with constant velocity $\boldsymbol{v}$ can be written:

$$\begin{aligned} \boldsymbol{E} &\propto \cos(\boldsymbol{k}\cdot\boldsymbol{v}t - \omega t) && (2.2)\\ &= \cos(-\omega_{\text{eff}}t), && (2.3)\end{aligned}$$

where $\omega_{\text{eff}} = \omega - \boldsymbol{k}\cdot\boldsymbol{v}$ is the effective angular frequency of the laser as seen by the atom.

This can be used to selectively transfer momentum to only fast-moving atoms, by tuning an incident laser to a slightly lower frequency than that required for a resonant absorption. If six lasers in counter-propagating pairs orthogonal to each other surround

a cloud of atoms, an idealised two-level atom theoretically will be cooled close to the *Doppler limit* [24, p 58]

$$k_B T_D = \frac{\hbar \Gamma}{2} \tag{2.4}$$

where Γ is the line width of the atomic transition. For the effectively two-level cooling transition used for Doppler cooling ^{87}Rb[1], this gives 146 μK, which is approximately a factor of a thousand too high for Bose-condensation. These atoms are also not trapped.

[1] The D_2 line, $5S_{1/2} \rightarrow 5P_{3/2}$, approximately 780 nm.

2.1.2 Magneto-optical and magnetic trapping

Magneto-optical trapping, first demonstrated in 1987 [25] comes from the realisation that a magnetic field can be used to *spatially* vary the detuning from resonance that the atoms in the above mentioned arrangement of lasers see. This is possible due to the Zeeman effect [26], in which the wavelengths of atomic transitions are shifted in a magnetic field.

If a field profile can be found that causes the transition to come closer to resonance as the atoms move away from a central point, then it forms a trap—atoms that stray too far from the centre will absorb more strongly and be deflected back[2].

[2] The polarisations of the beams are such that absorption from the inward facing beam occurs, rather than from the one that would accelerate the atom further outward!

The field configuration used is an anti-Helmholtz one: two coils opposite each other carrying opposing currents. The resulting magnetic field profile has a zero in the middle and increases in magnitude in all directions.

With the Doppler beams off, this magnetic field still provides a spatially-varying potential, due to the magnetic dipole interaction:

$$V(\boldsymbol{r}) = -\boldsymbol{\mu} \cdot \boldsymbol{B}, \tag{2.5}$$

where $\boldsymbol{\mu}$ is the atomic magnetic moment, and $\boldsymbol{B}$ the magnetic field. This only traps some atomic spin states, and has losses due to spin-flips [27] near the field zero.

The optimal magnetic field gradient for forming a magneto-optical trap (MOT) is lower than that required to merely hold atoms against gravity, and thus there is little if any magnetic trapping occurring in a MOT—the trapping almost entirely results from the position-dependent rate of photon absorption caused by the Zeeman shift due to the magnetic field. To transfer atoms from a MOT to a solely magnetic trap, the magnetic field gradient must be increased substantially.

2.1.3 Optical dipole trapping

Optical dipole trapping on the other hand relies on the *dipole force*, in which off-resonant light shifts the energy of the eigenstates of the combined atom-light system, the so called *dressed states*. This energy shift, called the *light shift*, depends on the intensity of the light, and so results in a potential that spatially varies as the intensity of the light. In the limit of large detuning (compared to Rabi frequency), the resulting energy shift for an atom in a ground state (the shift for an excited state is the same but with the opposite sign) is [24, p 8]:

$$\Delta E = \frac{\hbar \Omega^2}{4\delta} \tag{2.6}$$

where δ is the detuning from resonance and the Rabi frequency is:

$$\Omega = \frac{E_0}{\hbar} \langle e|e\hat{\boldsymbol{r}}|g\rangle \,, \tag{2.7}$$

where E_0 is the amplitude of the light's electric field and $\langle e|e\hat{\boldsymbol{r}}|g\rangle$ is the transition dipole moment between the excited state $|e\rangle$ and ground state $|g\rangle$ of an effectively two-level system. This transition dipole moment can be either for two specific sublevels

coupled by a laser, the calculation of which is detailed in Section 2.3.5, or an overall effective transition dipole moment between two manifolds containing many sublevels, if the detuning much larger compared to the spacing between the sublevels.

With the potential proportional to E_0^2, and thus the light's intensity, the force the atom experiences is proportional to the light's intensity gradient. For this reason, the dipole force is also called the *gradient force*. The name *dipole force* comes from the fact that the force can be equivalently understood to arise from the polarisability of atoms in a light field, in which an optically-induced dipole moment gives rise to a force that can be used to trap polarisable materials in optical tweezers [28].

2.1.4 Polarisation gradient cooling

Polarisation gradient cooling, also called Sisyphus cooling, was proposed in 1989 [29, 30] to explain experimentally measured cold atom cloud temperatures [31] which, at NIST in 1988, were found to be well below the expected limit obtainable by the well understood method of Doppler cooling[3], one of the few examples of experiments turning out better than expected. A one dimensional theory has been developed [29] which has found remarkable agreement with three dimensional experiments [32]

[3] As well as to explain other discrepancies between experiments and the theory of Doppler cooling, such as the optimal detuning of light being much greater than predicted.

One common configuration for Sisyphus cooling comprises two counter-propagating laser beams in each spatial dimension, both linearly polarised but with their polarisation angles perpendicular to one another. The optical field resulting from the two beams' superposition has regions of linear polarisation and of both helicities of circular polarisation, and varies between them on a length scale shorter than an optical wavelength.

The effect on multi-level atoms as they move from regions of one circular polarisation to another is that they are pumped alternately from one extreme of their spin-projection states to the other, alternately climbing and descending potential hills due to the dipole forces from the regions of different polarisations[4]. And so, like the Greek legend of Sisyphus[5], who was doomed to push a rock uphill for eternity, the atoms are climbing hills repeatedly. Due to the state dependence of the strength of the dipole forces, the atoms climb steeper hills than they descend, and are thus slowed and cooled.

[4] If you consider only one polarisation of light, its intensity varies sinusoidally in space, creating a series of potential hills and wells via the dipole force.

This type of cooling does not work in a magnetic field; the splitting of transition frequencies makes it impossible for an atom to traverse its spin manifold on one laser frequency. For this reason the Sisyphus cooling stage is performed with magnetic fields off, though a sufficiently short period is required such that the atoms can be recaptured when the trapping field is restored.

[5] Polarisation gradient cooling is but one of a family of so called 'Sisyphus cooling' methods, all of which involve atoms repeatedly climbing potential hills.

2.1.5 Evaporative cooling

The final stage of cooling is forced RF evaporative cooling [3, 33], which decreases the temperature of the cloud by systematically removing the hottest atoms. This is performed in a magnetic trap, which as mentioned earlier, only traps certain spin states. Evaporation proceeds by using an *RF knife* to induce spin flips in the atoms. The RF frequency is chosen such that it is only resonant with atoms some distance away from the centre of the trap (via the Zeeman shift). The furthest out atoms are the most energetic, possessing the energy to climb the magnetic potential the furthest. By flipping their spins, these atoms are ejected due to the magnetic field becoming anti-trapping for them.

The cloud is given some time to rethermalise and the knife[6] is moved inward where it removes slightly colder atoms. This is repeated until the desired compromise of lower temperature/lower atom number is reached. Usually some method is employed to prevent atoms near the centre of the trap from undergoing spin flips [27] as they move across the field zero. The method used in our lab is to use an optical dipole trap in combination with the magnetic trap [34], such that the coldest atoms get trapped in the dipole trap which is offset from the magnetic field zero.

[6] So called because it cuts the tail off the velocity distribution of the atom cloud.

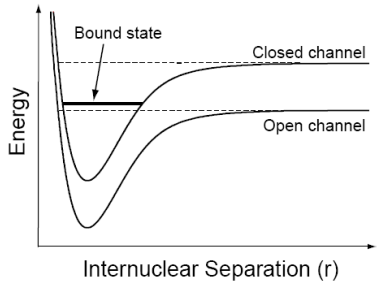

Figure 2.1: When atoms approach each other with spins aligned, they are in the *open channel*. In this channel they are unbound, but do not have enough energy to be free in the other channel—the *closed channel*. In the close range however, the atoms may have energy corresponding to a bound (molecular) state of the closed channel, a resonance that causes a divergence in the scattering length. The energy difference between the two channels can be tuned with a magnetic field and so these resonances can be induced in a wide range of situations.

2.1.6 Feshbach resonances

A Feshbach resonance [35] is an enhancement of the interparticle interaction strength when a certain magnetic field strength is applied.[7] This phenomenon was first studied in the context of ultracold atoms in 1993 [36], experimentally demonstrated in 1998 [37], and is now a staple of cold atom experiments.

[7] Feshbach resonances can also be induced optically and with RF but magnetic resonances are the most commonly used.

The interparticle interaction mentioned above:

$$g = \frac{2\pi\hbar^2 a}{m_r} \tag{2.8}$$

where m_r is the reduced mass of a pair of the interacting particles, is dependent on a parameter a called the *s-wave scattering length*, which characterises low energy collisions between atoms. It is sensitive not only to what species of atoms are colliding, but also to their spin states. For each combination of spins, there is a different inter-atomic potential (called a *channel*) which determines the collision dynamics (Figure 2.1).

The resulting scattering length is sensitive to any bound states of this inter-atomic potential which are near the collision energy. If the channels of different spin states are coupled via the hyperfine interaction[8], then the scattering length is also sensitive to bound states in the channels other than the one the atoms are in when they are far from each other. Due to the Zeeman effect, the energies between the different channels can be shifted with a magnetic field, and so a bound state can be shifted close to the collision energy, which causes the scattering length to diverge.

[8] Requiring that the atoms in question have a nuclear magnetic moment.

The end result is that at certain magnetic field strengths we find that atoms are much more strongly attracted to or repelled from each other.

A particular Feshbach resonance of interest is shown in Figure 2.2, and can be used to enhance the interspecies repulsion between ^{87}Rb and ^{41}K. The use of this resonance is assumed for the simulations in Chapter 5 to trap tracer particles more strongly in vortex cores of a BEC.

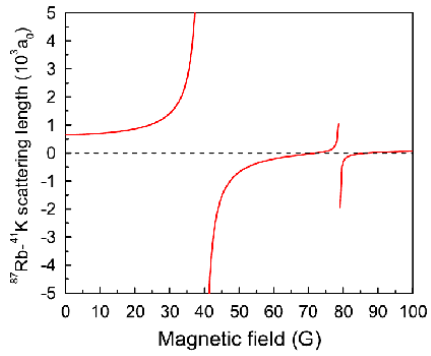

Figure 2.2: Predicted interspecies scattering length [38] as a function of magnetic field strength, for ^{41}K and ^{87}Rb both in their lowest energy hyperfine ground state. The ≈ 34 G resonance is one of the main reasons for this pair of atoms being used in this project. It has a particularly low field strength and large width compared to most Feshbach resonances. Figure reused with permission [38], © American Physical Society 2008.

2.2 Mean-field theory: The Gross–Pitaevskii equation and vortices

Bose-condensates are described well by *mean-field* theory, whereby the many-body wavefunction is approximated by a product of identical single-particle wavefunctions. Indeed, that the majority of the atoms are in the same quantum state is one of the defining features of BEC. The effect of interparticle interactions is included as a nonlinear term in the Schrödinger equation for the single particle wavefunctions, known as the Gross–Pitaevskii equation:

$$i\hbar\frac{\partial}{\partial t}\Psi(\boldsymbol{r},t) = \left[-\frac{\hbar^2}{2m}\nabla^2 + V(\boldsymbol{r}) + g|\Psi(\boldsymbol{r},t)|^2\right]\Psi(\boldsymbol{r},t), \tag{2.9}$$

where g characterises the strength of the interparticle interactions[9], and $\Psi = \sqrt{N}\Psi_{\text{single}}$ is the single-particle wavefunction scaled by the square root of the number of particles[10].

In the hydrodynamic formulation of quantum mechanics [39], the flow velocity of a spatial wavefunction can be defined by considering the probability current to be a product of density and velocity. This allows us to define the superfluid velocity of a BEC as:

$$\boldsymbol{v} = \frac{\hbar}{m}\nabla\phi \tag{2.10}$$

where ϕ is the complex phase of the condensate wavefunction Ψ. Integrating this velocity over any closed path γ gives us the circulation:

$$C = \frac{\hbar}{m}\oint_\gamma \nabla\phi\, ds \tag{2.11}$$

$$= \frac{\hbar}{m}2\pi n. \qquad n = 0, 1, 2 \ldots \tag{2.12}$$

The fact that the circulation is quantised means that vorticity cannot exist in the condensate except in one-dimensional lines, about which the wavefunction's phase winds by a multiple of 2π. These topological defects are the quantised vortices that are central to the vortex tracking simulations in Chapter 5.

[9] The nonlinear constant g is usually positive—having the effect of stabilising BECs by self-repulsion.

[10] Thus giving it the property that $|\Psi|^2$ is the particle density.

At a vortex core, the atom density of a BEC must go to zero. This can be intuitively understood to arise from centrifugal forces, but is also required in order for the wavefunction to be continuous and single-valued across the core. This drop in density in the vicinity of a vortex core is exploited by the method simulated in Chapter 5 to trap atoms within the cores.

2.3 The ^{87}Rb D line

We atomic physicists do our theory work at an intermediate level of abstraction, at which many quantities and systems of interest can be computed and simulated with accurate models using standard quantum mechanics, but with models that are not fully a priori. Instead, the Hamiltonians we feed to the machinery of quantum mechanics encapsulate some of the details we are not interested in or that are too hard to compute, with the link between the underlying layers of reductionism and the higher layer usually provided by experimentally measured values rather than calculations from fundamental physics. In this way we can readily compute results about the atoms we are interested in by treating them as simpler systems than they actually are, with some of the underlying details encapsulated by terms in an effective Hamiltonian for the dynamics that we are interested in.

In this section I'll summarise what the ^{87}Rb D line looks like from the perspective of a cold atom physicist, building up a Hamiltonian containing all 32 sublevels of the ground and first excited state of ^{87}Rb including fine structure, hyperfine structure, interaction with a magnetic field, and optical transitions between states. This Hamiltonian is the starting point for any calculations regarding cooling, trapping, and coherent control of ^{87}Rb, and for other alkali earth metals is much the same.

Much of the details of this section are drawn from references [24, 40–43], but the reader should be aware that there are considerable conventional and notational differences between different literature sources. What is presented here is summarised and framed in a way that I think is useful to an experimentalist looking to use the theory to make concrete calculations about real systems.

2.3.1 Fine structure

The rubidium 87 D line refers to the first excitation available to the sole outer electron of ^{87}Rb. Both the ground and excited state of this transition have the same principal quantum number, but different orbital angular momentum quantum numbers. Upon closer inspection, it is not just one transition between a ground state and an excited state—there are two excited states, and the two resulting transitions are called, in order of their transition frequencies, the D_1 and D_2 lines. Thus the ground and first excited state of ^{87}Rb are actually a ground state plus two non-degenerate excited states, once we take into account fine structure. The ground state is an S state (electronic orbital angular momentum quantum number $L = 0$), called the $S_{1/2}$ state, and the two excited states are P states ($L = 1$), one with the electron spin anti-aligned with its orbital angular momentum (resulting in total angular momentum quantum number $J = 1/2$) and one with the electron spin aligned with its orbital angular momentum ($J = 3/2$), called the $P_{1/2}$ and $P_{3/2}$ states respectively. In all of these states, ^{87}Rb's single outer-shell electron occupies an orbital with principle quantum number $n = 5$, which for brevity we leave out of the notation. The transition between $S_{1/2}$ and $P_{1/2}$ is called the D_1 line, with experimentally measured (angular) transition frequency ω_{D_1}, and the transition between $S_{1/2}$ and $P_{3/2}$ is the D_2 line with angular transition frequency ω_{D_2}. These transition frequencies correspond to optical wavelengths of $\lambda_{D_1} \approx 795\,\text{nm}$ and $\lambda_{D_2} \approx 780\,\text{nm}$ [40].

This fine structure is treated entirely empirically for our purposes, and so our base Hamiltonian for the rubidium D line, taking into account only fine structure, is simply a statement of the experimentally measured energy differences between the states:

$$\hat{H}_{\text{fs}} = \hbar\omega_{\text{D}_2}\hat{1}_{P_{3/2}} \oplus \hbar\omega_{\text{D}_1}\hat{1}_{P_{1/2}} \oplus \hat{0}_{S_{1/2}}, \tag{2.13}$$

where $\hat{1}_{P_{3/2}}$, $\hat{1}_{P_{1/2}}$, and $\hat{0}_{S_{1/2}}$ are identity and zero operators each acting on the subspace of states within the $P_{\frac{3}{2}}$, $P_{\frac{1}{2}}$, and $S_{\frac{1}{2}}$ manifolds respectively, and $\oplus$ is the direct sum.[11] The matrix representation H_{fs} of $\hat{H}_{\text{fs}}$ in the basis in which it is diagonal (which we will call the $\{|L_J\rangle\}$ basis, since L and J are good quantum numbers[12] for specifying one of the three states, which we write with the spectroscopic notation letter—S or P—corresponding to the value of L in place of its numerical value) is

$$H_{\text{fs}} = \begin{bmatrix} \begin{bmatrix} \ddots & & \\ & \hbar\omega_{\text{D}_2} & \\ & & \ddots \end{bmatrix} & & \\ & \begin{bmatrix} \ddots & & \\ & \hbar\omega_{\text{D}_1} & \\ & & \ddots \end{bmatrix} & \\ & & \begin{bmatrix} \ddots & & \\ & 0 & \\ & & \ddots \end{bmatrix} \end{bmatrix}, \tag{2.14}$$

which is a block-diagonal matrix with each block also being a diagonal matrix. We have not yet specified the size of each submatrix—the size of each differs and depends on how many hyperfine and Zeeman sublevels are in that state.

This base Hamiltonian is worth singling out since the energy differences between its three states are orders of magnitude larger than any of the energy differences between hyperfine and Zeeman sublevels within them. When doing any sort of calculations or simulations then, this time-independent Hamiltonian can often be removed from the equations using an interaction picture (see Section 3.2.2), as done in Section 5.5 in simulating laser cooling.

2.3.2 Hyperfine structure

Within each of the $S_{1/2}$, $P_{1/2}$ and $P_{3/2}$ states, the single outer-shell electron's total angular momentum $\hat{\boldsymbol{J}}$ has an interaction with ^{87}Rb's nuclear angular momentum $\hat{\boldsymbol{I}}$. This results in multiple discrete energy levels depending on the relative orientation of the two separate angular momenta. The interaction Hamiltonian for this hyperfine structure is[13] [40, 44]:

$$\hat{H}_{\text{hfs}} = \frac{A_{\text{hfs}}}{\hbar^2}\hat{\boldsymbol{I}}\cdot\hat{\boldsymbol{J}} + \frac{B_{\text{hfs}}}{\hbar^2}\frac{3(\hat{\boldsymbol{I}}\cdot\hat{\boldsymbol{J}})^2 + \frac{3}{2}\hat{\boldsymbol{I}}\cdot\hat{\boldsymbol{J}} - I(I+1)J(J+1)}{2I(2I-1)J(2J-1)}, \tag{2.15}$$

where J is the total angular momentum quantum number of the electron, equal to either $\frac{1}{2}$ or $\frac{3}{2}$ depending on which state in the D line we are considering, $I = \frac{3}{2}$ is the total angular momentum quantum number of the nucleus, and A_{hfs} and B_{hfs} are empirically determined coupling constants. Here we see the boundary between the quantities we can calculate with the machinery of quantum mechanics and those that we determine empirically—this expression applies so long as J and I are good quantum numbers,[14] and the two terms are the dipolar and quadrupolar interactions [44] between two angular momenta, with the coupling constants determined empirically and encapsulating details that are difficult to compute a-priori, such as relativistic effects and the exact shape of the electron orbitals given the presence of inner shell electrons. For the spherically-symmetric $S_{1/2}$ ground state, there is no quadrupolar interaction and so B_{hfs} is only non-zero for the two P excited states.[15] The values of A_{hfs} and B_{hfs} for each of the three L_J states of the D line can be found in [40].

[11] Not to be confused with the Kronecker sum, with which it shares notation. The direct sum concatenates matrices as blocks, producing a larger, block-diagonal matrix with dimension equal to the *sum* of the dimensions of the matrices being direct-summed, whereas the Kronecker sum is the regular sum of matrices after each has been multiplied using the Kronecker-product with identity matrices with sizes of the other matrices in the sum, producing matrices with dimension equal to the *product* of those being summed.

[12] A *good quantum number* is a number that can be used to label (not necessarily uniquely) an energy eigenstate, and on which the energy eigenvalue of that state depends. Saying J and L are good quantum numbers is saying that the eigenstates of the overall Hamiltonian are also eigenstates of $\hat{L}^2$ and $\hat{J}^2$, since eigenstates of these two operators can be specified by stating their quantum numbers L and J.

[13] Note that this expression differs from those in the cited references by a factor of $1/\hbar^2$—this is because I define the $\hat{\boldsymbol{I}}$ and $\hat{\boldsymbol{J}}$ angular momentum operators in SI units, rather than in units of $\hbar^2$.

[14] J is a good quantum number if the hyperfine splitting is small compared to the spacing between the three states of the D line, which it is, and I is a good quantum number if the hyperfine splitting is small compared to the energy difference between the ground state and the first *nuclear* excited state, which it most certainly is.

[15] The quadrupolar term should be explicitly excluded from numerical computations of the hyperfine splitting on the $S_{1/2}$ state, as it contains a division by zero in this case, which may lead to erroneous results even if the term is subsequently multiplied by $B_{\text{hfs}} = 0$.

For a given state of the D line, we can construct the matrix representation $\overset{\mathcal{I}\times\mathcal{J}}{H}_{\text{hfs}}$ of $\hat{H}_{\text{hfs}}$ in the $\mathcal{I}\times\mathcal{J}$ basis—which we define as the basis in which both the z vector components $\hat{I}_z$ and $\hat{J}_z$ of $\hat{\boldsymbol{I}}$ and $\hat{\boldsymbol{J}}$ are diagonal—by constructing matrix representations $\overset{\mathcal{I}\times\mathcal{J}}{\boldsymbol{I}}$ and $\overset{\mathcal{I}\times\mathcal{J}}{\boldsymbol{J}}$ of the operators $\hat{\boldsymbol{I}}$ and $\hat{\boldsymbol{J}}$ in that basis and then applying the expression (2.15). The overset $\mathcal{I}\times\mathcal{J}$ on each of the matrices indicates that the matrix is a representation of its respective operator in the $\mathcal{I}\times\mathcal{J}$ basis, so named because its basis vectors can be obtained via a Cartesian product[16] of the two sets of basis vectors from the $\mathcal{I}$ and $\mathcal{J}$ bases for the two individual nuclear and electronic angular momentum degrees of freedom, in which $\hat{I}_z$ and $\hat{J}_z$ are respectively diagonal.

To construct matrix representations of angular momentum operators in the product space basis $\mathcal{I}\times\mathcal{J}$, we first need their matrix representations $\overset{\mathcal{I}}{\boldsymbol{I}}$ and $\overset{\mathcal{J}}{\boldsymbol{J}}$ in the bases of their respective subspaces, which we will write as $\boldsymbol{I}$ and $\boldsymbol{J}$ for brevity. We can then expand the two operators into the product space by applying a Kronecker product with an appropriate identity matrix to each:

$$\overset{\mathcal{I}\times\mathcal{J}}{\boldsymbol{I}} = \boldsymbol{I} \otimes \overset{\mathcal{J}}{\mathbb{I}} \tag{2.16}$$

$$\overset{\mathcal{I}\times\mathcal{J}}{\boldsymbol{J}} = \overset{\mathcal{I}}{\mathbb{I}} \otimes \boldsymbol{J} \tag{2.17}$$

where $\overset{\mathcal{I}}{\mathbb{I}}$ is the matrix representation of the identity operator in the $\mathcal{I}$ basis of the nuclear spin degree of freedom, equal to a $(2I+1)\times(2I+1)$ identity matrix, with $\overset{\mathcal{J}}{\mathbb{I}}$ defined similarly for the electronic spin degree of freedom.

Each of the two matrices $\boldsymbol{I}$ and $\boldsymbol{J}$ is actually a vector of matrices, one for the angular momentum projection in each of the directions x, y and z. The procedure for constructing such matrices for arbitrary total angular momentum quantum numbers is as follows.[17] I'll show the procedure for constructing J_x, J_y and J_z only for an arbitrary J, the procedure is identical for computing the vector components of $\boldsymbol{I}$.

For a given total angular momentum quantum number J, the vector components of $\boldsymbol{J}$ in the $\mathcal{J}$ basis (the basis in which J_z is diagonal) can be constructed using the raising and lowering operators $\hat{J}_+$ and $\hat{J}_-$. Since the action of the raising and lowering operators on an eigenstate of J_z with angular momentum projection quantum number m_J is to produce an adjacent $(m_J \pm 1)$ eigenstate multiplied by a constant [45, p 192], this fact can be used to compute the non-zero matrix elements of $\hat{J}_+$ and $\hat{J}_-$ in the $\{|m_J\rangle\}$ basis (the basis kets of which we identify with the standard basis vectors[18] for a $2J+1$ dimensional space in order to define the set of concrete basis vectors $\mathcal{J}$):

$$\langle m_J + 1|\hat{J}_+|m_J\rangle = \hbar\sqrt{J(J+1) - m_J(m_J+1)}, \qquad -J \le m_J < J, \tag{2.18}$$

$$\langle m_J - 1|\hat{J}_-|m_J\rangle = \hbar\sqrt{J(J+1) - m_J(m_J-1)}, \qquad -J < m_J \le J, \tag{2.19}$$

and therefore compute explicit matrices for J_+ and J_- in the $\{|m_J\rangle\}$ basis:[19]

$$J_+ = \begin{bmatrix} 0 & \langle J|\hat{J}_+|J-1\rangle & 0 & \dots & & \\ \dots & 0 & \langle J-1|\hat{J}_+|J-2\rangle & 0 & \dots & \\ & & \ddots & \ddots & \ddots & \\ & & \dots & 0 & \langle -J+2|\hat{J}_+|-J+1\rangle & 0 \\ & & & \dots & 0 & \langle -J+1|\hat{J}_+|-J\rangle \\ & & & & \dots & 0 \end{bmatrix}, \tag{2.20}$$

$$J_- = \begin{bmatrix} 0 & \dots & & & & \\ \langle J-1|\hat{J}_-|J\rangle & 0 & \dots & & & \\ 0 & \langle J-2|\hat{J}_-|J-1\rangle & 0 & \dots & & \\ \dots & 0 & \langle J-3|\hat{J}_-|J-2\rangle & 0 & \dots & \\ & & \ddots & \ddots & \ddots & \\ & & \dots & 0 & \langle -J|\hat{J}_-|-J+1\rangle & 0 \end{bmatrix}, \tag{2.21}$$

[16] The different types of products available for matrices, vectors, operators, spaces, and their sets of basis vectors are rife with subfield-specific conventions and overloaded notation, leading to much ambiguity. Although 'Cartesian product' connotes well what I mean here, it is still ambiguous. What I mean is that the $\mathcal{I}\times\mathcal{J}$ basis is the set of basis vectors $\{\boldsymbol{u}\otimes\boldsymbol{v} \mid \boldsymbol{u}\in\mathcal{I}, \boldsymbol{v}\in\mathcal{J}\}$. The notation $\boldsymbol{u}\otimes\boldsymbol{v}$, which is also ambiguous, denotes the Kronecker product of two column vectors, producing a column vector with number of elements equal to the product of the number of elements in each of the two vectors—not the 'outer' or 'tensor' product, which would produce a matrix.

[17] This explicit procedure for constructing the matrix representations of these operators is useful for entering into a programming language to produce programs capable of performing atomic physics calculations for arbitrary total angular momentum quantum number J, without having to explicitly enter the angular momentum operators for each value of J, which can be tedious and prone to human error.

[18] I will hereafter use the terms '$\mathcal{J}$ basis' and '$\{|m_J\rangle\}$ basis', and similarly for other bases, interchangeably, in the understanding that this identification of standard unit vectors with the basis kets is implied.

[19] We're using the standard convention of ordering the eigenkets $\{|m_J\rangle\}$ in descending order of m_J. This is at odds with the computer programming convention of looping over most indices in ascending order, and so care should be taken when constructing these matrices in a computer program.

both of which have non-zero values along only one non-main diagonal adjacent to the main diagonal, and which form a Hermitian conjugate pair (or indeed, a transpose pair, since all elements are real). The matrix representations of $\hat{J}_x$ and $\hat{J}_y$ can then be computed by rearranging the defining expressions for $\hat{J}_+$ and J_-;

$$\hat{J}_+ = \hat{J}_x + \mathrm{i}\hat{J}_y, \tag{2.22}$$

$$\hat{J}_- = \hat{J}_x - \mathrm{i}\hat{J}_y, \tag{2.23}$$

for $\hat{J}_x$ and $\hat{J}_y$, and then applying the result to our matrix representations of $\hat{J}_+$ and $\hat{J}_-$ to obtain matrix representations J_x and J_y of $\hat{J}_x$ and $\hat{J}_y$ in the $\{|m_J\rangle\}$ basis:

$$J_x = \frac{J_+ + J_-}{2}, \tag{2.24}$$

$$J_y = \frac{J_+ - J_-}{2\mathrm{i}}. \tag{2.25}$$

Finally, since $\{|m_J\rangle\}$ is the eigenbasis of J_z with eigenvalues $\{\hbar m_J\}$, the matrix representation of J_z in the $\mathcal{J}$ basis is simply the diagonal matrix of eigenvalues:

$$J_z = \begin{bmatrix} \hbar J & & & \\ & \hbar(J-1) & & \\ & & \ddots & \\ & & & -\hbar J \end{bmatrix}. \tag{2.26}$$

We can also construct the matrix representation J^2 of the total (squared) angular momentum operator $\hat{J}^2$ as

$$J^2 = J_x^2 + J_y^2 + J_z^2, \tag{2.27}$$

or equivalently

$$J^2 = J(J+1)\hbar^2 \overset{\mathcal{J}}{\mathbb{I}}, \tag{2.28}$$

since every m_J state is an eigenstate of the J^2 operator with eigenvalue $J(J+1)\hbar^2$.

The above prescription can be used to produce matrix representations of angular momentum operators J_x, J_y, J_z and J^2 for any integer or half-integer total angular momentum quantum number J. The three components can be considered a vector of matrices, $\boldsymbol{J}$, for the vector angular momentum operator $\hat{\boldsymbol{J}}$. Below is a Python function that computes these matrices as well as the corresponding eigenvectors:

```python
import numpy as np
hbar = 1.054571628e-34

def angular_momentum_operators(J):
    """Construct matrix representations of the angular momentum operators Jx,
    Jy, Jz and J2 in the eigenbasis of Jz for given total angular momentum
    quantum number J. Return them, as well as the number of angular momentum
    projection states, a list of angular momentum projection quantum numbers
    mJ, and a list of their corresponding eigenvectors, in the same order as
    the matrix elements (in descending order of mJ)."""
    n_mJ = int(round(2*J + 1))
    mJlist = np.linspace(J, -J, n_mJ)
    Jp = np.diag([hbar * np.sqrt(J*(J+1) - mJ*(mJ + 1)) for mJ in mJlist if mJ < J], 1)
    Jm = np.diag([hbar * np.sqrt(J*(J+1) - mJ*(mJ - 1)) for mJ in mJlist if mJ > -J], -1)
    Jx = (Jp + Jm) / 2
    Jy = (Jp - Jm) / 2j
    Jz = np.diag([hbar*mJ for mJ in mJlist])
    J2 = Jx**2 + Jy**2 + Jz**2
    basisvecs_mJ = [vec for vec in np.identity(n_mJ)]
    return Jx, Jy, Jz, J2, n_mJ, mJlist, basisvecs_mJ
```

Using the above prescription to construct a matrix representation of the $\hat{J}$ operator with $J = \frac{1}{2}$ for the $S_{1/2}$ and $P_{1/2}$ states, or $J = \frac{3}{2}$ for the $P_{3/2}$ state, and to construct a matrix representation of the $\hat{I}$ operator with $I = \frac{3}{2}$, we can explicitly construct a matrix representation of the hyperfine interaction Hamiltonian for any state of the ^{87}Rb D line. Remaining is to obtain the matrix representations of the two operators in the $\mathcal{I} \times \mathcal{J}$ product space basis using (2.16) and (2.17), and then we can apply (2.15) to our matrices to obtain the matrix representation $\overset{\mathcal{I}\times\mathcal{J}}{H}_{\text{hfs}}$ of $\hat{H}_{\text{hfs}}$ for a given J corresponding to one of the three states on the D line:

$$\overset{\mathcal{I}\times\mathcal{J}}{H}_{\text{hfs}} = \frac{A_{\text{hfs}}}{\hbar^2} \overset{\mathcal{I}\times\mathcal{J}}{\boldsymbol{I}} \cdot \overset{\mathcal{I}\times\mathcal{J}}{\boldsymbol{J}} + \frac{B_{\text{hfs}}}{\hbar^2} \frac{3(\overset{\mathcal{I}\times\mathcal{J}}{\boldsymbol{I}} \cdot \overset{\mathcal{I}\times\mathcal{J}}{\boldsymbol{J}})^2 + \frac{3}{2} \overset{\mathcal{I}\times\mathcal{J}}{\boldsymbol{I}} \cdot \overset{\mathcal{I}\times\mathcal{J}}{\boldsymbol{J}} - I(I+1)J(J+1)}{2I(2I-1)J(2J-1)}, \tag{2.29}$$

where the products of vector components within the dot products are computed with ordinary matrix multiplication. Alternatively, one can use the matrices in their individual subspaces rather than their equivalents in the product space, so long as one interprets the dot products as 'Kronecker dot products':

$$\overset{\mathcal{I}\times\mathcal{J}}{H}_{\text{hfs}} = \frac{A_{\text{hfs}}}{\hbar^2} \boldsymbol{I} \overset{\otimes}{\cdot} \boldsymbol{J} + \frac{B_{\text{hfs}}}{\hbar^2} \frac{3(\boldsymbol{I} \overset{\otimes}{\cdot} \boldsymbol{J})^2 + \frac{3}{2} \boldsymbol{I} \overset{\otimes}{\cdot} \boldsymbol{J} - I(I+1)J(J+1)}{2I(2I-1)J(2J-1)}, \tag{2.30}$$

where $\overset{\otimes}{\cdot}$ is the Kronecker dot product:

$$\boldsymbol{I} \overset{\otimes}{\cdot} \boldsymbol{J} \equiv I_x \otimes J_x + I_y \otimes J_y + I_z \otimes J_z. \tag{2.31}$$

In the above way one can construct an explicit matrix representation of $\hat{H}_{\text{hfs}}$ in the $\mathcal{I} \times \mathcal{J}$ basis for the hyperfine interaction for a given L_J state of the D line.

Column vectors in the $\mathcal{I} \times \mathcal{J}$ basis

Because the matrix representations $\boldsymbol{I}$ and $\boldsymbol{J}$ of the electron and nuclear angular momentum operators were constructed in the $\{|m_I\rangle\}$ and $\{|m_I\rangle\}$ bases of their respective subspaces, the matrices we have constructed are in the basis $\mathcal{I} \times \mathcal{J}$ with basis vectors:

$$\mathcal{I} \times \mathcal{J} = \left\{ |m_I\, m_J\rangle \,\middle|\, |m_I\rangle \in \{|m_I\rangle\}, |m_J\rangle \in \{|m_J\rangle\} \right\}, \tag{2.32}$$

where $|m_I\, m_J\rangle = |m_I\rangle \otimes |m_J\rangle$. The vector representation $\boldsymbol{\psi}$ of a state vector $|\psi\rangle$ in this basis is:

$$\boldsymbol{\psi} = \begin{bmatrix} \langle m_I{=}I\ m_J{=}J|\psi\rangle \\ \langle m_I{=}I\ m_J{=}J{-}1|\psi\rangle \\ \vdots \\ \langle m_I{=}I\ m_J{=}{-}J|\psi\rangle \\ \langle m_I{=}I{-}1\ m_J{=}J|\psi\rangle \\ \langle m_I{=}I{-}1\ m_J{=}J{-}1|\psi\rangle \\ \vdots \\ \langle m_I{=}{-}I\ m_J{=}{-}J|\psi\rangle \end{bmatrix}. \tag{2.33}$$

The hyperfine Hamiltonian is not diagonal in the $\{|m_I\, m_J\rangle\}$ basis. The basis in which it is diagonal—the $\{|F\, m_F\rangle\}$ basis—will be discussed in Section 2.3.4.

2.3.3 Zeeman sublevels

The states differing only in their m_F quantum numbers—called Zeeman sublevels—are degenerate in energy with respect to the hyperfine Hamiltonian, but an external magnetic field lifts this degeneracy. The Zeeman effect [26, 40] results in an energy shift proportional to the external magnetic field $\boldsymbol{B}$ and to a system's magnetic moment $\boldsymbol{\mu}$:

$$V = -\boldsymbol{\mu} \cdot \boldsymbol{B}. \tag{2.34}$$

Our atom is a composite particle, made of a nucleus with its own intrinsic magnetic moment, an electron with its own intrinsic one as well, and a contribution from the orbital motion of the electron about the nucleus. Each magnetic moment is proportional to the angular momentum of the subsystem in question, with the proportionality constants, called Landé g-factors written as dimensionless multiples of $-\mu_B/\hbar$, where μ_B is the Bohr magneton.[20] Since J is a good quantum number so long as energy shifts are smaller than the (large) energy spacing between the three L_J states of the D line, on the level we work we don't consider the electron spin and orbital angular momenta separately, rather we encapsulate them with a single, empirically determined Landé g-factor g_J [40] for the magnetic moment of the electron in each of the three L_J states. Similarly we consider the nucleus as a single spin with an experimentally determined g_I [40], resulting in a Zeeman Hamiltonian:

[20] We are using the sign convention that defines the Landé g-factor g_J for the electron as positive.

$$\hat{H}_Z = -\hat{\boldsymbol{\mu}} \cdot \boldsymbol{B} \tag{2.35}$$

$$= -\left(\hat{\boldsymbol{\mu}}_I + \hat{\boldsymbol{\mu}}_J\right) \cdot \boldsymbol{B} \tag{2.36}$$

$$= \left(\frac{g_I \mu_B}{\hbar}\hat{\boldsymbol{I}} + \frac{g_J \mu_B}{\hbar}\hat{\boldsymbol{J}}\right) \cdot \boldsymbol{B}. \tag{2.37}$$

Separate g_S and g_L values are known and can be used in two terms instead of the one containing $\hat{\boldsymbol{J}}$ above if J is not a good quantum number, but in the regime we work that is not usually the case (and if it were, the fine and hyperfine structure Hamiltonians above would also be inadequate since they assume that J is a good quantum number). If J is a good quantum number then it is more accurate to use the above expression with empirically measured g_J values, since they encapsulate QED effects and corrections due to the multi-electron structure of ^{87}Rb that are not captured by the simple Zeeman Hamiltonian with separate $\hat{\boldsymbol{S}}$ and $\hat{\boldsymbol{L}}$ terms [40].

If the energy shifts from the Zeeman effect are small compared to the hyperfine spitting, then F (the total spin quantum number, defined in the next section) is a good quantum number and a given hyperfine level can be treated as a single magnetic moment subject to the Zeeman Hamiltonian:

$$\hat{H}_{Z\,\mathrm{lin}} = \frac{g_F \mu_B}{\hbar}\hat{\boldsymbol{F}} \cdot \boldsymbol{B}, \tag{2.38}$$

where [40]

$$g_F = g_J \frac{F(F+1) - I(I+1) + J(J+1)}{2F(F+1)} + g_I \frac{F(F+1) + I(I+1) - J(J+1)}{2F(F+1)}. \tag{2.39}$$

The direction in which each Zeeman sublevel shifts in energy for small magnetic fields is depicted in Figure 2.3. Experimentally, Zeeman shifts that depart from this linear regime are not infrequently encountered, and so it is an approximation that cannot always be made.

An explicit matrix representation $\overset{(1\times J)}{H}_Z$ of $\hat{H}_Z$ in the $\{|m_I\ m_J\rangle\}$ basis for each of the three L_J states of the D line can be constructed by applying (2.37) to the matrix representations of $\hat{\boldsymbol{I}}$ and $\hat{\boldsymbol{J}}$ in that basis:

$$\overset{(1\times J)}{H}_Z = -\overset{(1\times J)}{\boldsymbol{\mu}} \cdot \boldsymbol{B}, \tag{2.40}$$

where

$$\overset{(1\times J)}{\boldsymbol{\mu}} = -\frac{g_I \mu_B}{\hbar}\overset{(1\times J)}{\boldsymbol{I}} - \frac{g_J \mu_B}{\hbar}\overset{(1\times J)}{\boldsymbol{J}} \tag{2.41}$$

$$= -\frac{g_I \mu_B}{\hbar}\boldsymbol{I} \otimes \overset{(J)}{\mathbb{I}} - \frac{g_J \mu_B}{\hbar}\overset{(1)}{\mathbb{I}} \otimes \boldsymbol{J}. \tag{2.42}$$

2.3.4 Putting it all together: the $\{|F\ m_F\rangle\}$ basis

So far we have described how to construct a matrix representation of the fine structure Hamiltonian for the three L_J states of the ^{87}Rb D line, as well as matrix representations of each state's hyperfine and Zeeman Hamiltonians, these latter two in the same $\{|m_I\ m_J\rangle\}$ basis. In the subspaces of each of the three L_J states then, we can sum together the hyperfine and Zeeman Hamiltonians to form (a matrix representation of) a Hamiltonian that takes interactions into account:

$$\overset{\mathcal{I}\times\mathcal{J}}{H}_{P_{3/2}} = \overset{\mathcal{I}\times\mathcal{J}}{H}_{\mathrm{hfs}\,P_{3/2}} + \overset{\mathcal{I}\times\mathcal{J}}{H}_{\mathrm{Z}\,P_{3/2}}, \tag{2.43}$$

$$\overset{\mathcal{I}\times\mathcal{J}}{H}_{P_{1/2}} = \overset{\mathcal{I}\times\mathcal{J}}{H}_{\mathrm{hfs}\,P_{1/2}} + \overset{\mathcal{I}\times\mathcal{J}}{H}_{\mathrm{Z}\,P_{1/2}}, \tag{2.44}$$

$$\overset{\mathcal{I}\times\mathcal{J}}{H}_{S_{1/2}} = \overset{\mathcal{I}\times\mathcal{J}}{H}_{\mathrm{hfs}\,S_{1/2}} + \overset{\mathcal{I}\times\mathcal{J}}{H}_{\mathrm{Z}\,S_{1/2}}, \tag{2.45}$$

where the subscripts $P_{3/2}$, $P_{1/2}$, and $S_{1/2}$ on the terms on the right hand side indicate that the expressions for the matrix elements of the hyperfine and Zeeman Hamiltonians are to be evaluated using the specific values of J, A_{hfs}, B_{hfs}, and g_J relevant to that state. The total Hamiltonian for the D line including fine structure, hyperfine structure and the Zeeman interaction is then

$$\hat{H} = \hat{H}_{\mathrm{fs}} + \left(\hat{H}_{P_{3/2}} \oplus \hat{H}_{P_{1/2}} \oplus \hat{H}_{S_{1/2}}\right) \tag{2.46}$$

$$\Rightarrow \hat{H} = \left(\hbar\omega_{D_2}\hat{1}_{P_{3/2}} + \hat{H}_{P_{3/2}}\right) \oplus \left(\hbar\omega_{D_1}\hat{1}_{P_{1/2}} + \hat{H}_{P_{1/2}}\right) \oplus \hat{H}_{S_{1/2}}, \tag{2.47}$$

the matrix representation of which in the $\{|L\ J\ m_I\ m_J\rangle\}$ basis is the block diagonal matrix

$$\overset{\mathcal{I}\times\mathcal{J}}{H}_{\mathrm{D}\,^{87}\mathrm{Rb}} = \begin{bmatrix} \left[\hbar\omega_{D_2} + \overset{\mathcal{I}\times\mathcal{J}}{H}_{P_{3/2}}\right] & & \\ & \left[\hbar\omega_{D_1} + \overset{\mathcal{I}\times\mathcal{J}}{H}_{P_{1/2}}\right] & \\ & & \left[\overset{\mathcal{I}\times\mathcal{J}}{H}_{S_{1/2}}\right] \end{bmatrix}, \tag{2.48}$$

where addition of scalars with matrices implies the addition of a scalar multiple of the appropriately sized identity matrix.

While $\overset{\mathcal{I}\times\mathcal{J}}{H}_{\mathrm{D}\,^{87}\mathrm{Rb}}$ is a block diagonal matrix, each of the three submatrices is not diagonal, since m_I and m_J are not good quantum numbers for the hyperfine interaction (though they are good quantum numbers for the Zeeman interaction and hence $\overset{\mathcal{I}\times\mathcal{J}}{H}_{\mathrm{D}\,^{87}\mathrm{Rb}}$ becomes approximately diagonal at high magnetic field). Once one has constructed $\overset{\mathcal{I}\times\mathcal{J}}{H}_{\mathrm{D}\,^{87}\mathrm{Rb}}$ or its submatrices, there are two other bases one might consider transforming the submatrices into depending on the circumstances. One is the $\Sigma\mathcal{F} = \{|F\ m_F\rangle\}$ basis[21], in which the matrix representation of the hyperfine interaction is diagonal. The matrix representation of the Zeeman Hamiltonian is also approximately diagonal in the $\{|F\ m_F\rangle\}$ basis, so long as one is in the linear Zeeman regime. Furthermore, the transition dipole moments for optical transitions are most easily calculated in the $\{|L_J\ F\ m_F\rangle\}$ basis, as we'll see in the next subsection. For these reasons the $\{|F\ m_F\rangle\}$ basis is the most commonly used and referred to.

The $\{|F\ m_F\rangle\}$ basis is defined as the simultaneous eigenbasis of the $\hat{F}^2$ and $\hat{F}_z$ operators, which are the total (squared) and z component of the total angular momentum operator $\hat{\boldsymbol{F}} = \hat{\boldsymbol{I}} + \hat{\boldsymbol{J}}$, the matrix forms F^2 and F_z of which in the $|m_I\ m_J\rangle$ basis can be

[21] The name $\Sigma\mathcal{F}$ refers to the fact that this is the basis that lends itself to the interpretation of the total space being the direct sum of subspaces, each of which has a well defined F quantum number. So if $\mathcal{I}\times\mathcal{J}$ is a basis for a product space with $(2I+1)\times(2J+1)$ dimensions, then $\Sigma\mathcal{F}$ is a basis for a *sum space* with $\sum_i(2F_i+1)$ dimensions, where F_i ranges from $|I-J|$ to $I+J$ in integer steps. The product and sum bases have the same dimensionality and span the same space, so the difference is only in the identification and composition of their component subspaces.

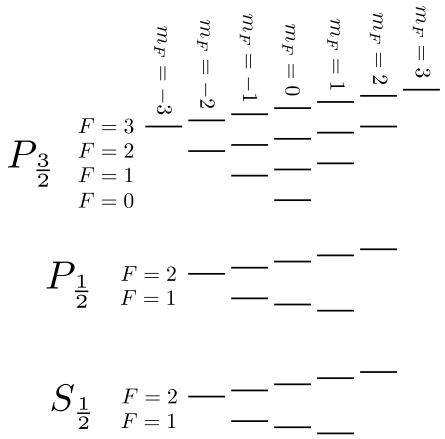

Figure 2.3: The 32 states of the rubidium 87 D line, ordered vertically by energy (not to scale) in a small magnetic field. At zero magnetic field, Zeeman sublevels sharing a common F quantum number and within the same hyperfine level are degenerate. At small magnetic fields this degeneracy is lifted, with state energies shifting in the directions depicted. However, F is no longer a good quantum number at non-zero magnetic field, as most of the non-degenerate sublevels are actually equal to linear combinations of two states of different F quantum numbers. Nevertheless at low magnetic fields the states are labelled using F anyway, since one F state dominates the linear combination, and at higher magnetic fields the states are labelled using an index α, equal to the value of F of the state that would dominate the superposition if the field were smoothly reduced to zero. m_F remains a good quantum number at non-zero magnetic field however, and so at all fields a state can be specified by the numbers L, J, α (equal to F at small field) and m_F.

constructed from the matrix forms of the individual angular momentum operators:

$$\begin{aligned}
\overset{1\times J}{\boldsymbol{F}} &= \overset{1\times J}{\boldsymbol{I}} + \overset{1\times J}{\boldsymbol{J}} \\
&= \boldsymbol{I} \otimes \overset{J}{\mathbb{I}} + \overset{1}{\mathbb{I}} \otimes \boldsymbol{J} && (2.49)\\
\overset{1\times J}{F_z} &= \overset{1\times J}{I_z} + \overset{1\times J}{J_z} \\
&= I_z \otimes \overset{J}{\mathbb{I}} + \overset{1}{\mathbb{I}} \otimes J_z && (2.50)\\
\overset{1\times J}{F^2} &= \overset{1\times J}{\boldsymbol{F}} \cdot \overset{1\times J}{\boldsymbol{F}} && (2.51)
\end{aligned}$$

The $\{|Fm_F\rangle\}$ basis allows the eigenstates of the hyperfine interaction to be labelled with F and m_F quantum numbers. For a state of the ^{87}Rb D line with electron total angular momentum quantum number J, there are $1 + J + I - |(I - J)|$ hyperfine levels, with F quantum numbers running from $|I - J|$ to $I + J$. Within each hyperfine level there are $2F + 1$ degenerate states with different m_F quantum numbers ranging from $-F$ to F. This results in a total of 32 possible states for the rubidium D line, a schematic of which is shown in Figure 2.3.

To transform each submatrix between the $\{|m_I\ m_J\rangle\}$ and $\{|F\ m_F\rangle\}$ bases, we use a unitary matrix whose elements are Clebsch–Gordan coefficients, each defined as the inner product of a $|F\ m_F\rangle$ state with a $|m_I\ m_J\rangle$ state (written in full as $|I\ m_I\ J\ m_J\rangle$), and

calculable as [40]

$$\langle I\, m_I\, J\, m_J | F\, m_F\rangle = (-1)^{I-J+m_F}\sqrt{2F+1}\begin{pmatrix} I & J & F \\ m_I & m_J & -m_F\end{pmatrix} \tag{2.52}$$

where the object in parentheses is a Wigner 3-j symbol. The Clebsch–Gordan coefficients are real: $\langle F\, m_F | I\, m_I\, J\, m_J\rangle = \langle I\, m_I\, J\, m_J | F\, m_F\rangle$, and are zero unless $m_I + m_J = m_F$. Given that the possible range of the F quantum number is from $|I-J|$ to $I+J$, and the possible range of m_F quantum numbers is from $-F$ to F for each F (both in integer steps), an explicit construction of the unitary matrix U_{CG} of Clebsch–Gordan coefficients, in the convention where the column vectors in the $\{|F\, m_F\rangle\}$ basis have F running from its highest value to lowest from top to bottom, and m_F also running from highest to lowest within each value of F, (omitting I and J for brevity) is:

$$U_{\text{CG}} = \begin{bmatrix} \langle m_I{=}I\ m_J{=}J|F{=}I{+}J\ mF{=}F\rangle & \dots & \langle m_I{=}I\ m_J{=}J|F{=}I{+}J\ m_F{=}{-}F\rangle & \dots & \dots & \langle m_I{=}I\ m_J{=}J|F{=}|I{-}J|\ m_F{=}{-}F\rangle \\ \vdots & & & & & \\ \langle m_I{=}{-}I\ m_J{=}J|F{=}I{+}J\ mF{=}F\rangle & & \ddots & & & \vdots \\ \vdots & & & & & \\ \langle m_I{=}{-}I\ m_J{=}{-}J|F{=}I{+}J\ mF{=}F\rangle & & \dots & & & \langle m_I{=}{-}I\ m_J{=}{-}J|F{=}|I{-}J|\ m_F{=}{-}F\rangle \end{bmatrix}. \tag{2.53}$$

This the matrix that takes vectors from the $\{|F\, m_F\rangle\}$ basis to the $\{|m_I\, m_J\rangle\}$ basis, so its Hermitian conjugate $U_{\text{CG}}^\dagger$ is needed for the inverse transformation (which is equal to the transpose since the matrix elements are real). Within the convention we're using to order the matrix elements, the vector representation $\boldsymbol{\psi}$ of a state vector $|\psi\rangle$ in the $\{|F\, m_F\rangle\}$ basis of one of the L_J states is:

$$\boldsymbol{\psi} = \begin{bmatrix} \langle F{=}I{+}J\ m_F{=}F|\psi\rangle \\ \langle F{=}I{+}J\ m_F{=}F{-}1|\psi\rangle \\ \vdots \\ \langle F{=}I{+}J\ m_F{=}{-}F|\psi\rangle \\ \langle F{=}I{+}J{-}1\ m_F{=}F|\psi\rangle \\ \langle F{=}I{+}J{-}1\ m_F{=}F{-}1|\psi\rangle \\ \vdots \\ \langle m_F{=}|I{-}J|\ m_F{=}{-}F|\psi\rangle \end{bmatrix}. \tag{2.54}$$

Each submatrix of the total Hamiltonian for the D line can be transformed into its $\{|F\, m_F\rangle\}$ basis by using a unitary matrix of Clebsch–Gordan coefficients with the appropriate value of J, yielding a total Hamiltonian for the rubidium 87 D line in the $\{|L_J\, F\, m_F\rangle\}$ basis:

$${}^{\Sigma\mathcal{F}}H_{\text{D}\,{}^{87}\text{Rb}} = \begin{bmatrix} \left[\hbar\omega_{\text{D}_2} + U_{\text{CG}\,3/2}^\dagger\, {}^{\mathcal{I}\times\mathcal{J}}H_{P_{3/2}}\, U_{\text{CG}\,3/2}\right] & & \\ & \left[\hbar\omega_{\text{D}_1} + U_{\text{CG}\,1/2}^\dagger\, {}^{\mathcal{I}\times\mathcal{J}}H_{P_{1/2}}\, U_{\text{CG}\,1/2}\right] & \\ & & \left[U_{\text{CG}\,1/2}^\dagger\, {}^{\mathcal{I}\times\mathcal{J}}H_{S_{1/2}}\, U_{\text{CG}\,1/2}\right] \end{bmatrix} \tag{2.55}$$

$$= \begin{bmatrix} \left[\hbar\omega_{\text{D}_2} + {}^{\Sigma\mathcal{F}}H_{P_{3/2}}\right] & & \\ & \left[\hbar\omega_{\text{D}_1} + {}^{\Sigma\mathcal{F}}H_{P_{1/2}}\right] & \\ & & \left[{}^{\Sigma\mathcal{F}}H_{S_{1/2}}\right] \end{bmatrix} \tag{2.56}$$

where the additional subscripts on the unitary matrices indicates the value of J used in its construction. Armed with the $\{|F, m_F\rangle\}$ basis, we have little remaining reason to use the $\{|m_I\, m_J\rangle\}$ basis any more.

Here is a recap of what we have taken into account with our model of the Rubidium D line:

- The two lowest quantum numbers $L = 0$ and $L = 1$ of the electron's orbital angular momentum. This yields the S ground state and P excited state.
- Fine structure: the two possible orientations of the electron's spin with respect to its orbital angular momentum. This splits the P excited state into two states, $P_{1/2}$ and $P_{3/2}$, and leaves the S ground state as the single state $S_{1/2}$. The energies of these three states are determined entirely empirically—without any modelling of the fine structure.
- Hyperfine structure: The possible orientations of the electron's total angular momentum with respect to the nuclear angular momentum. This splits each state so far into $1 + J + I - |(I - J)|$ hyperfine states. The hyperfine interaction is treated semi-empirically, using an analytic form of the hyperfine interaction but with empirically determined coupling constants within each of the three L_J states of the D line.
- Zeeman effect: the possible orientation of the angular momenta $\hat{\boldsymbol{I}}$ and $\hat{\boldsymbol{J}}$—or at low field their sum $\hat{\boldsymbol{F}}$—onto an external magnetic field. The Zeeman Hamiltonian is modelled analytically, but with empirically determined Landé g factors for each of the three L_J states of the D line.

The above considerations allow us to construct a Hamiltonian, as a concrete matrix representation, for all 32 possible states of the rubidium D line in a fixed magnetic field, in a basis in which it is diagonal for the case of zero magnetic field. In a non-zero magnetic field, one can analytically diagonalise the $S_{1/2}$ and $P_{1/2}$blocks of this matrix using the Breit–Rabi formula [41, p. 347; 46], but the same cannot be done for the entire D line, and diagonalisation must be performed numerically for the $P_{3/2}$ block. At non-zero magnetic field, F ceases to be a good quantum number, but states are nonetheless labelled using a variable α, defined as the value of F a state *would* have if the magnetic field were reduced to zero. m_F remains a good quantum number however, and thus at non-zero magnetic field the $\{|\alpha\, m_F\rangle\}$ basis, is the one in which the Hamiltonian is diagonal, with the transformation between it and the $\{|F\, m_F\rangle\}$ basis requiring numerical diagonalisation.

If the magnetic field is static, then this Hamiltonian $\hat{H}_{\mathrm{D}\,^{87}\mathrm{Rb}}$ is a time independent Hamiltonian, to which additional time-dependent Hamiltonians can be added to take into account optical or RF transitions, detailed below. In a dynamic magnetic field, only $\hat{H}_{\mathrm{hfs}}$ is time-independent, in which case the Zeeman Hamiltonian will also be part of the time-dependent part of the Hamiltonian. This has implications for the use of an interaction picture (described in Section 3.2.2), in which the time independent part of a Hamiltonian can be analytically removed from the equations of motion, which can make further computations more tractable both analytically and numerically.

2.3.5 Optical dipole transitions

Optical transitions due to laser fields appear as off-diagonal matrix elements in the $\{|F\, m_F\rangle\}$ basis[22]. A given laser of a certain polarisation may give rise to non-zero transition matrix elements between several different pairs of states, depending on selection rules.

[22] The matrix elements can be obtained in the $\{|\alpha\, m_F\rangle\}$ basis at non-zero magnetic field via the numerically computed transformation mentioned in the previous subsection

The potential the outer electron of rubidium is subject to in a classical electric field is the electric dipole potential [24, 41]:

$$\hat{H}_{\mathrm{d}} = -\hat{\boldsymbol{d}} \cdot \boldsymbol{E}, \tag{2.57}$$

where $\boldsymbol{E}$ is the classical electric field and $\hat{\boldsymbol{d}}$ is the electric dipole operator:

$$\hat{\boldsymbol{d}} = -e\hat{\boldsymbol{r}}, \tag{2.58}$$

which gives the electric dipole of the atom in terms of the electron's charge $-e$ and its position operator (with respect to the nucleus) $\hat{\boldsymbol{r}}$. A single matrix element of $\hat{H}_\mathrm{d}$ for coupling an initial state $|L_J\ F\ m_F\rangle$ to a final state $|L'_J,\ F'\ m'_F\rangle$ is therefore:

$$\langle L'_J,\ F'\ m'_F|\hat{H}_\mathrm{d}|L_J\ F\ m_F\rangle = \langle L'_J,\ F'\ m'_F|e\hat{\boldsymbol{r}}\cdot\boldsymbol{E}|L_J\ F\ m_F\rangle\,, \tag{2.59}$$

which we will abbreviate as

$$\langle n'|\hat{H}_\mathrm{d}|n\rangle = \langle n'|e\hat{\boldsymbol{r}}\cdot\boldsymbol{E}|n\rangle\,, \tag{2.60}$$

writing the initial state $|L_J\ F\ m_F\rangle = |n\rangle$ and final state $|L'_J,\ F'\ m'_F\rangle = |n'\rangle$, encapsulating the set of quantum numbers as a single index[23] for the state.

[23] The choice of the symbol n is unrelated to the principal quantum number, which does not vary between the states of the D line.

Plane waves

The electric field of a plane wave of amplitude E_0, polarisation unit vector $\hat{\boldsymbol{\varepsilon}}$ and angular frequency ω can be written:

$$\boldsymbol{E}(\boldsymbol{r},t) = \frac{E_0}{2}\left(\hat{\boldsymbol{\varepsilon}}\mathrm{e}^{\mathrm{i}(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)} + \hat{\boldsymbol{\varepsilon}}^*\mathrm{e}^{-\mathrm{i}(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)}\right), \tag{2.61}$$

where * is complex conjugation, which in the case of a real valued polarisation unit vector $\hat{\boldsymbol{\varepsilon}}$ does nothing, making the above equal to a cosine wave $E_0\hat{\boldsymbol{\varepsilon}}\cos(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)$ as expected for a linearly polarised plane wave. However, (2.61) also allows for arbitrary circular or elliptical polarisation, if $\hat{\boldsymbol{\varepsilon}}$ is allowed to have complex components. This comes with the caveat, however, that the amplitude E_0 is a misnomer in these cases and no longer corresponds to any actual amplitude. For example, a circularly polarised plane wave propagating in the z direction constructed using complex polarisation vector $\hat{\boldsymbol{\varepsilon}} = \mp(\hat{\boldsymbol{x}} \pm i\hat{\boldsymbol{y}})/\sqrt{2}$) and 'amplitude' E_0 actually has an electric field vector with constant magnitude $E_0/\sqrt{2}$. The 'amplitude' E_0 in (2.61) is actually equal to $\sqrt{2}E_\mathrm{rms}$ and thus only equal to the peak electric field strength in the case of linear polarisation. Using the intensity of the beam[24] $I = \varepsilon_0 c E_\mathrm{rms}^2$ instead, which is less ambiguous and more relatable to experimentalists [42], resolves this potential confusion:[25]

$$\boldsymbol{E}(\boldsymbol{r},t) = \frac{1}{2}\sqrt{\frac{2I}{\varepsilon_0 c}}\left(\hat{\boldsymbol{\varepsilon}}\mathrm{e}^{\mathrm{i}(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)} + \hat{\boldsymbol{\varepsilon}}^*\mathrm{e}^{-\mathrm{i}(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)}\right). \tag{2.62}$$

[24] Not to be confused with the nuclear spin I, distinguishable by context.

[25] One possible confusion of many given the diverse notational and conventional differences throughout the literature.

The spherical basis

Consider one of the three polarisation unit vectors $\hat{\boldsymbol{\varepsilon}}_q$ for which the photon has a well defined angular momentum projection in the z direction:

$$\hat{\boldsymbol{\varepsilon}}_\pm = \mp(\hat{\boldsymbol{x}} \pm i\hat{\boldsymbol{y}})/\sqrt{2}, \tag{2.63}$$

$$\hat{\boldsymbol{\varepsilon}}_0 = \hat{\boldsymbol{z}}, \tag{2.64}$$

where $q = \pm 1$ is denoted with the subscript $_\pm$. These are the basis vectors of the spherical tensor basis [41], which is the basis in which it is easiest to compute matrix elements, as well as the one in which it is easiest to think about which transitions a given laser will drive. Light with polarisation vector $\hat{\boldsymbol{\varepsilon}}_\pm$ is called $\sigma^\pm$ polarised, and excites ground states to excited states that differ in m_F quantum number by ± 1. A single beam with this polarisation would have to be propagating in the z direction, with left or right hand circular polarisation. Light with polarisation vector $\hat{\boldsymbol{\varepsilon}}_0$ is called π polarised, and drives transitions between states of equal m_F quantum numbers. A single beam with this polarisation would have to be propagating with a $\boldsymbol{k}$ vector in the x, y plane and be linearly polarised in the z direction. However, lasers with propagation directions and polarisation

vectors different from these three configurations can be decomposed into the spherical basis. For example, light with linear polarisation vector in the x, y plane drives both σ^+ and σ^- transitions.

The spherical basis is defined somewhat differently than how we are used to for ordinary real-valued vectors. The components A_q of a vector $\boldsymbol{A}$ in the spherical basis are defined as the coefficients of *complex conjugates* of the basis vectors:

$$\boldsymbol{A} = \sum_q A_q \hat{\boldsymbol{\varepsilon}}_q^* = \sum_q (-1)^q A_q \hat{\boldsymbol{\varepsilon}}_{-q}, \tag{2.65}$$

with the second equality resulting from the property of the basis vectors that

$$\hat{\boldsymbol{\varepsilon}}_q^* = (-1)^q \hat{\boldsymbol{\varepsilon}}_{-q}, \tag{2.66}$$

and the dot product in terms of the spherical basis components of two vectors is:

$$\boldsymbol{A} \cdot \boldsymbol{B} = \sum_q (-1)^q A_q B_{-q}. \tag{2.67}$$

The definition (2.65) has the counter-intuitive consequence that the unit vectors themselves, written in terms of their components $\hat{\boldsymbol{\varepsilon}}_q = \left((\varepsilon_q)_-, (\varepsilon_q)_0, (\varepsilon_q)_+\right)$ in the spherical basis are:

$$\hat{\boldsymbol{\varepsilon}}_- = (0, 0, -1), \tag{2.68}$$
$$\hat{\boldsymbol{\varepsilon}}_0 = (0, 1, 0), \tag{2.69}$$
$$\hat{\boldsymbol{\varepsilon}}_+ = (-1, 0, 0), \tag{2.70}$$

that is, the unit vector $\hat{\boldsymbol{\varepsilon}}_q$ has a component of $(-1)^q$ in the $-q$ position, and zeros elsewhere.

Considering a plane wave with polarisation vector equal to one of the spherical basis vectors $\hat{\boldsymbol{\varepsilon}}_q$, we have the electric field:

$$\boldsymbol{E}_q(\boldsymbol{r}, t) = \frac{1}{2}\sqrt{\frac{2I}{\varepsilon_0 c}}\left(\hat{\boldsymbol{\varepsilon}}_q e^{i(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)} + \hat{\boldsymbol{\varepsilon}}_q^* e^{-i(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)}\right), \tag{2.71}$$

which, removing the complex conjugation using (2.66), can be written:

$$\boldsymbol{E}_q(\boldsymbol{r}, t) = \frac{1}{2}\sqrt{\frac{2I}{\varepsilon_0 c}}\left(\hat{\boldsymbol{\varepsilon}}_q e^{i(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)} + (-1)^q \hat{\boldsymbol{\varepsilon}}_{-q} e^{-i(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)}\right). \tag{2.72}$$

The dipole approximation

Substituting (2.72) into (2.59) results in the matrix element of the dipole Hamiltonian for a q polarised plane wave:

$$\langle n'|\hat{H}_d(q, I)|n\rangle = \frac{1}{2}\sqrt{\frac{2I}{\varepsilon_0 c}}\left(\langle n'|e\hat{\boldsymbol{r}}\cdot\hat{\boldsymbol{\varepsilon}}_q e^{i(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)}|n\rangle + (-1)^q \langle n'|e\hat{\boldsymbol{r}}\cdot\hat{\boldsymbol{\varepsilon}}_{-q} e^{-i(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)}|n\rangle\right). \tag{2.73}$$

The *dipole approximation* is the approximation that the spatial extent of the electron's orbital is much smaller than the wavelength of the light, and thus that the factors of $e^{\pm i\boldsymbol{k}\cdot\boldsymbol{r}}$ are approximately constant over the integral and can be taken outside it, with $\boldsymbol{r}$ taken to be the expectation value of the atom's position[26]. For optical wavelengths this is a good

[26] The resulting classical position $\boldsymbol{r}$ of the atom should not be confused with the electron's position operator $\hat{\boldsymbol{r}}$ with respect to the nucleus. Strictly speaking, (2.73) should have $\boldsymbol{r}$ in the exponents replaced with a quantum operator with a different name to signify that it is the absolute position of the electron rather than the position relative to the nucleus, but I suspect this would cause more confusion than it

approximation, yielding our matrix element in the dipole approximation:

$$\langle n'|\hat{H}_{\mathrm{d}}(q,I)|n\rangle = \frac{1}{2}\sqrt{\frac{2I}{\varepsilon_0 c}}\left(\langle n'|e\hat{\boldsymbol{r}}\cdot\hat{\boldsymbol{\varepsilon}}_q|n\rangle\,\mathrm{e}^{\mathrm{i}(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)} + (-1)^q\langle n'|e\hat{\boldsymbol{r}}\cdot\hat{\boldsymbol{\varepsilon}}_{-q}|n\rangle\,\mathrm{e}^{-\mathrm{i}(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)}\right). \tag{2.74}$$

Evaluating the dot products using (2.67), we get $\hat{\boldsymbol{r}}\cdot\hat{\boldsymbol{\varepsilon}}_q = \sum_p(-1)^p\hat{r}_p(\varepsilon_q)_{-p} = \hat{r}_q$ and $\hat{\boldsymbol{r}}\cdot\hat{\boldsymbol{\varepsilon}}_{-q} = \sum_p(-1)^p\hat{r}_p(\varepsilon_{-q})_{-p} = \hat{r}_{-q}$, where $\hat{r}_q$ are the components of the position operator $\hat{\boldsymbol{r}}$ in the spherical basis, the position space representations of which are proportional to the $\ell = 1$ spherical harmonics and the radial coordinate r:

$$\langle r,\theta,\phi|\hat{r}_\pm|r,\theta,\phi\rangle = \mp\frac{r}{\sqrt{2}}\sin\theta\mathrm{e}^{\pm\mathrm{i}\phi}, \tag{2.75}$$

$$\langle r,\theta,\phi|\hat{r}_0|r,\theta,\phi\rangle = r\cos(\theta). \tag{2.76}$$

We now have our final expression for the matrix element $\langle n'|\hat{H}_{\mathrm{d}}(q,I)|n\rangle$ of the dipole Hamiltonian for laser intensity I, angular frequency and wavenumber ω and $\boldsymbol{k}$ respectively, and complex polarisation vector $\hat{\boldsymbol{\varepsilon}}_q$, in the dipole approximation in terms of matrix elements of $e\hat{r}_q$, the calculation of which is detailed below:

$$\langle n'|\hat{H}_{\mathrm{d}}(q,I)|n\rangle = \frac{1}{2}\sqrt{\frac{2I}{\varepsilon_0 c}}\left(\langle n'|e\hat{r}_q|n\rangle\,\mathrm{e}^{\mathrm{i}(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)} + (-1)^q\langle n'|e\hat{r}_{-q}|n\rangle\,\mathrm{e}^{-\mathrm{i}(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)}\right). \tag{2.77}$$

A further approximation often used is to approximate the atom as stationary, replacing the $\boldsymbol{k}\cdot\boldsymbol{r}$ from the exponents with a constant (usually zero if the exact phase of the optical field is not of interest). Alternately, if the atom is moving at an approximately constant velocity, the $\boldsymbol{k}\cdot\boldsymbol{r}$ term can be absorbed into ω resulting in an effective Doppler-shifted angular frequency as in Section 2.1.1. We will leave the $\boldsymbol{k}\cdot\boldsymbol{r}$ terms where they are, but note that they are often absent in most literature sources for the preceding reasons.

The rotating wave approximation

If the plane wave's angular frequency ω is close to the resonant angular frequency ω_{D_1} or ω_{D_2} of the D_1 or D_2 line, then one of the two exponential terms is oscillating at a frequency much further from resonance (approximately 2ω) than the other, and thus does not contribute significantly to coupling between states. Discarding it is known as the *rotating wave approximation* (RWA), so called because in an interaction picture (section 3.2.2) where the evolution of the base Hamiltonian $\hat{H}_{\mathrm{D}\,^{87}\mathrm{Rb}}$ is factored out—the so called *rotating frame*—this fact is more readily apparent and a formal approximation can be made showing that the off-resonant term averages to zero over time scales much shorter than any other dynamics. Which term is discarded depends on whether the transition being considered is from a ground state to excited state, or vice-versa. Our matrix element $\langle n'|\hat{H}_{\mathrm{d}}(q,I)|n\rangle$ in the RWA becomes:

$$\langle n'|\hat{H}_{\mathrm{d}}(q,I)|n\rangle \overset{\text{RWA}}{\approx} \begin{cases} \frac{1}{2}\sqrt{\frac{2I}{\varepsilon_0 c}}\langle n'|e\hat{r}_q|n\rangle\,\mathrm{e}^{\mathrm{i}(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)}, & E_{n'} > E_n \\ \frac{(-1)^q}{2}\sqrt{\frac{2I}{\varepsilon_0 c}}\langle n'|e\hat{r}_{-q}|n\rangle\,\mathrm{e}^{-\mathrm{i}(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)}, & E_{n'} < E_n \end{cases}. \tag{2.78}$$

The spherical basis components $\hat{r}_q$ of the position operator $\hat{\boldsymbol{r}}$ are not Hermitian operators, instead obeying the following relation [41]:

$$\hat{r}_q^\dagger = (-1)^q\hat{r}_{-q}, \tag{2.79}$$

$$\Rightarrow \langle n|e\hat{r}_q|n'\rangle^* = (-1)^q\langle n'|e\hat{r}_{-q}|n\rangle, \tag{2.80}$$

allowing us to simplify (2.78) to:

$$\langle n'|\hat{H}_{\mathrm{d}}(q,I)|n\rangle \overset{\text{RWA}}{\approx} \begin{cases} \frac{\hbar}{2}\Omega_{n'n}(q,I)\,\mathrm{e}^{\mathrm{i}(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)}, & E_{n'} > E_n \\ \\ \frac{\hbar}{2}\Omega^*_{nn'}(q,I)\,\mathrm{e}^{-\mathrm{i}(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)}, & E_{n'} < E_n \end{cases}, \tag{2.81}$$

where $\Omega_{n'n}(q,I)$ is the *Rabi frequency* of the $n \to n'$ transition for polarisation q and intensity (I),

$$\Omega_{n'n}(q,I) \equiv \hbar\sqrt{\frac{2I}{\varepsilon_0 c}}\,\langle n'|e\hat{r}_q|n\rangle\,, \tag{2.82}$$

with the symmetry that $\Omega_{n'n}(q,I) = \Omega^*_{nn'}(q,I)$. Because of this symmetry, the electric dipole Hamiltonian for a plane wave in the rotating wave approximation is Hermitian, and all its matrix elements can be computed considering only transitions from ground state to excited state (with the Rabi frequencies for the reverse transitions being simply the complex conjugates of those of the forward transitions).

The rotating wave approximation is accurate when $\omega - \omega_0 \gg |\omega + \omega_0|$, which is the case for transitions being driven close to resonance. When far off-resonant light is being used however—for example to impart optical dipole forces as in optical traps—it may not be a good approximation, and the full matrix elements (2.77) of the dipole Hamiltonian may need to considered.

Dipole matrix elements

We now move on to how to compute the dipole matrix elements

$$\langle n'|e\hat{r}_q|n\rangle \equiv \langle L'_J,\, F'\; m'_F|e\hat{r}_q|L_J\; F\; m_F\rangle\,. \tag{2.83}$$

Using these matrix elements, one can compute Rabi frequencies for matrix elements of the dipole Hamiltonian (2.81) in the rotating wave approximation, or to calculate the full matrix elements (2.77) of the dipole Hamiltonian.

In the spherical basis we've been working in, most of the integrals required to compute these matrix elements can be treated analytically, and there are a series of reductions that can be made to ultimately reduce the matrix elements down to a product of an analytically computable coefficient depending only an angular momentum quantum numbers, and an experimentally measurable 'reduced' dipole matrix element.

Following [40], the dependence on the photon angular momentum projection quantum number q and atomic angular momentum projection quantum numbers m_F and m'_F can be encapsulated in a Clebsch–Gordan coefficient using the Wigner–Eckart theorem, allowing us to write:

$$\langle L'_J,\, F'\; m'_F|e\hat{r}_q|L_J\; F\; m_F\rangle = \langle F'\; m'_F|F\; 1\; m_F\; q\rangle\;\langle L'_J,\, F'\|e\hat{\boldsymbol{r}}\|L_J\; F\rangle \tag{2.84}$$

$$= (-1)^{F-1+m'_F}\sqrt{2F'+1}\begin{pmatrix} F & 1 & F' \\ m_F & q & -m'_F \end{pmatrix}\langle L'_J,\, F'\|e\hat{\boldsymbol{r}}\|L_J\; F\rangle\,, \tag{2.85}$$

where the object in parentheses is a Wigner 3-j symbol, and $\langle L'_J,\, F'\|e\hat{\boldsymbol{r}}\|L_J\; F\rangle$ is the 'reduced' matrix element for the transition from $|L_J\; F\rangle$ to $|L'_J,\, F'\rangle$ free of dependence on angular momentum projection quantum numbers. Due to fact that the Clebsch–Gordan coefficient is zero unless $m_F + q = m_F$, these matrix elements are zero unless the final and initial states are being coupled by a laser of the correct polarisation to conserve angular momentum projection in the z direction, one of the selection rules of optical transitions.[27]

[27] Observe equation (2.77) however, and note that for the case of $\sigma^{\pm}$ light, both $q = \pm 1$ dipole matrix elements are present, and thus $\sigma^{\pm}$ light couples a ground state to *both* $m'_F = m_F \pm 1$ excited states, with one being far off resonance and discarded by the rotating wave approximation for the case of near-resonant light.

The final reduction is to express this reduced matrix element as yet another analytic coefficient multiplied by a reduced matrix element that doesn't depend on the F quantum numbers:

$$\langle L'_{J'}\, F' \| e\hat{\boldsymbol{r}} \| L_J\, F \rangle = (-1)^{F+J'+1+I} \sqrt{(2F+1)(2J'+1)} \begin{Bmatrix} J' & J & 1 \\ F & F' & I \end{Bmatrix} \langle L'_{J'} \| e\hat{\boldsymbol{r}} \| L_J \rangle\,, \tag{2.86}$$

where I is the nuclear spin quantum number, the object in curly braces is a Wigner 6-j symbol, and $\langle L'_{J'} \| e\hat{\boldsymbol{r}} \| L_J \rangle$ is a reduced matrix element for the transition from $|L_J\rangle$ to $|L'_{J'}\rangle$ free of dependence on F quantum numbers.

This final reduced matrix element can be linked to experimentally known numerical values via the line widths Γ_{D_1} and Γ_{D_2} of D_1 and D_2 lines [41, eq. 7.3.7.4]:

$$\Gamma = \frac{\omega_0^3}{3\pi\varepsilon_0 \hbar c^3} \frac{2J_g+1}{2J_e+1} |\langle L'_{J_g} \| e\hat{\boldsymbol{r}} \| L_{J_e} \rangle|^2\,, \tag{2.87}$$

where Γ and ω_0 are the line width and resonant (angular) frequency respectively of the transition, L'_{J_g} is the ground state ($S_{1/2}$) and L_{J_e} is the excited state (either $P_{1/2}$ or $P_{3/2}$) of the transition. Thus the reduced matrix elements for transitions from excited to ground states can be expressed in terms of known constants. For the reduced matrix elements for transitions from ground to excited states instead, one cannot simply take the complex conjugates of the excited-to-ground reduced matrix elements—the reduced matrix elements are defined instead such that they have the following property [41, eq. 7.3.5.1]:

$$\langle L_J \| e\hat{\boldsymbol{r}} \| L'_{J'} \rangle = (-1)^{J-J'} \sqrt{\frac{2J'+1}{2J+1}} \langle L'_{J'} \| e\hat{\boldsymbol{r}} \| L_J \rangle^*\,. \tag{2.88}$$

Putting these together gives the following reduced matrix elements for the D_1 and D_2 lines in both directions:

$$\langle S_{1/2} \| e\hat{\boldsymbol{r}} \| P_{1/2} \rangle = \sqrt{\frac{3\pi\varepsilon_0 \hbar c^3 \Gamma_{\mathrm{D}_1}}{\omega_{\mathrm{D}_1}^3}} \tag{2.89}$$

$$\langle S_{1/2} \| e\hat{\boldsymbol{r}} \| P_{3/2} \rangle = \sqrt{\frac{6\pi\varepsilon_0 \hbar c^3 \Gamma_{\mathrm{D}_2}}{\omega_{\mathrm{D}_2}^3}} \tag{2.90}$$

$$\langle P_{1/2} \| e\hat{\boldsymbol{r}} \| S_{1/2} \rangle = \langle S_{1/2} \| e\hat{\boldsymbol{r}} \| P_{1/2} \rangle^* \tag{2.91}$$

$$\langle P_{3/2} \| e\hat{\boldsymbol{r}} \| S_{1/2} \rangle = -\frac{1}{\sqrt{2}} \langle S_{1/2} \| e\hat{\boldsymbol{r}} \| P_{3/2} \rangle^* \tag{2.92}$$

The first two of these, having only their absolute values related to the line widths and frequencies via (2.87), have an undetermined phase factor which we have taken to be equal to +1. Since only relative phases are important, if we limit ourselves to only one of the D_1 or D_2 lines at a time, all dipole matrix elements, being proportional to these reduced dipole matrix elements, will have the correct relative phases. However, we are imposing a fixed phase on *both* the reduced dipole matrix elements for the D line. We can verify that a relative phase factor of +1 between the two reduced dipole matrix elements is the correct choice by decomposing [40] the reduced matrix elements into yet another Wigner-6j symbol and a reduced dipole matrix element $\langle L' = \frac{1}{2} \| e\hat{\boldsymbol{r}} \| L = 1 \rangle$, common to both fine structure lines:

$$\langle L'_{J'} \| e\hat{\boldsymbol{r}} \| L_J \rangle = (-1)^{J+L+S+1} \sqrt{(2J+1)(2L'+1)} \begin{Bmatrix} L' & L & 1 \\ J & J' & S \end{Bmatrix} \langle L' \| e\hat{\boldsymbol{r}} \| L \rangle\,, \tag{2.93}$$

where $S = 1/2$ is the electron spin. This expression can be used to verify the ratio:

$$\frac{\langle S_{1/2}||e\hat{\boldsymbol{r}}||P_{3/2}\rangle}{\langle S_{1/2}||e\hat{\boldsymbol{r}}||P_{1/2}\rangle} = \sqrt{2}, \tag{2.94}$$

confirming that the choice of relative phase factor +1 between the two L_J reduced dipole matrix elements is the correct one.

We have now arrived at the ground floor of reductionism for most experimental atomic physicists when it comes to optical transitions, with the values of the transition dipole matrix elements resting on the foundation of empirically measured reduced dipole matrix elements, and calculable by traversing this section in reverse.

2.3.6 Magnetic dipole transitions

After all the complexity of optical dipole transitions, magnetic dipole transitions—for either RF transitions between Zeeman sublevels, or microwave transitions between hyperfine states—are relatively simple. The matrix elements in the $\{|F\ m_F\rangle\}$ basis for a perturbing magnetic field $\boldsymbol{B}(t)$ are simply those of the Zeeman Hamiltonian (2.40):

$$\langle F'\ m'_F|\hat{H}_Z|F\ m_F\rangle = -\langle F'\ m'_F|\hat{\boldsymbol{\mu}}\cdot\boldsymbol{B}|F\ m_F\rangle\,, \tag{2.95}$$

which for a magnetic plane wave with linear polarisation $i \in \{x, y, z\}$ and amplitude B_0, and in the dipole approximation, is:

$$\langle F'\ m'_F|\hat{H}_Z(i, B_0)|F\ m_F\rangle = -\langle F'\ m'_F|\hat{\mu}_i|F\ m_F\rangle\, B_0\cos(\boldsymbol{k}\cdot\boldsymbol{r} - \omega t). \tag{2.96}$$

The magnetic dipole matrix elements $\langle F'\ m'_F|\hat{\mu}_i|F\ m_F\rangle$ can be computed using the known matrix form (2.41) of the magnetic moment operator μ_i in the $\{|m_I\ m_J\rangle\}$ basis, and transforming it into the $\{|F\ m_F\rangle\}$ basis using a unitary of Clebsch–Gordan coefficients as in (2.52).

As with optical dipole transitions, the cosine can be written as a sum of exponentials, revealing co-rotating and counter-rotating terms:

$$\langle F'\ m'_F|\hat{H}_Z(i, B_0)|F\ m_F\rangle = -\frac{B_0}{2}\langle F'\ m'_F|\hat{\mu}_i|F\ m_F\rangle\left(\mathrm{e}^{\mathrm{i}(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)} + \mathrm{e}^{-\mathrm{i}(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)}\right), \tag{2.97}$$

allowing for a rotating wave approximation and definition of a Rabi frequency as with optical dipole transitions:

$$\langle F'\ m'_F|\hat{H}_Z(i, B_0)|F\ m_F\rangle \overset{\text{RWA}}{\approx} \begin{cases} \frac{\hbar}{2}\Omega_{F'\ m'_F\ F\ m_F}(i, B_0)\,\mathrm{e}^{\mathrm{i}(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)}, & E_{F'\ m'_F} > E_{F\ m_F} \\ \frac{\hbar}{2}\Omega^*_{F\ m_F\ F'\ m'_F}(i)\,\mathrm{e}^{-\mathrm{i}(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)}, & E_{F'\ m'_F} < E_{F\ m_F} \end{cases}, \tag{2.98}$$

where $\Omega_{F'\ m'_F\ F\ m_F}(i, B_0) = -\hbar B_0\langle F'\ m'_F|\hat{H}_Z(i, B_0)|F\ m_F\rangle$.

The required linear combinations of the above matrix elements can be used to compute matrix elements for RF waves of different phases, or circular polarisations. It is also often practical, when numerically modelling magnetic dipole transitions, particularly between Zeeman sublevels, to simply include the magnetic field vector as a function as time and evaluate the entire matrix H_Z directly in whichever basis one is working, rather than decomposing it into fixed frequency waves. This is particularly useful for frequency sweeps and other complex magnetic field sequences, which may not be good candidates for the rotating wave approximation, or may not be represented well as single-frequency waves at all. This is most practical for transitions between Zeeman sublevels, since the

energy splittings can be on the order of MHz and therefore well sampled by numerics with nanosecond to microsecond sized timesteps, which are feasible to numerically compute. Microwave transitions, being 6.8 GHz in ^{87}Rb would require sub-nanosecond timesteps to simulate in this direct way, and so is more taxing but still within the range of possibility, as opposed to optical transitions which are in the hundreds of THz and impractical to simulate without rotating wave approximations and the assumption of monochromatic waves as treated in the previous subsection.

2.3.7 Summary

We are now finally at the end of our assembly of the Hamiltonian for the ^{87}Rb D line in the presence of magnetic fields and monochromatic plane waves. Armed with this Hamiltonian and the ability to compute its coupling terms for various lasers, the atomic physicist can predict and simulate much of what is needed for the basics of laser cooling, trapping, and coherent control.

Chapter 3

Quantum mechanics on a computer

This chapter details many of the methods used by cold atom physicists to compute numerical results pertaining to cold atom systems. Many a problem in quantum mechanics is not analytically solvable, especially when the real world of experimental physics rears its ugly head, violating theorists' assumptions of simplicity left and right. In particular, atomic physics experiments are time-dependent, with each run of an experiment generally proceeding in stages. Lasers may turn on and off, magnetic fields may vary in magnitude and direction, RF pulses may be chirped to reliably induce particular transitions [47]. Much of the numerical computation performed by researchers in cold atom physics groups such as ours are accordingly of the time-dependent variety, and are fairly literal simulations of specific experiments that may be carried out in the lab.

This chapter is primarily a pedagogical presentation of existing numerical techniques relevant to computational studies of cold atom physics and quantum mechanics more generally. I explain some fundamentals of representing quantum mechanical problems on a computer and then present my favourite algorithms for propagating state vectors and wavefunctions in time. To spoil the surprise: my favourite timestepping algorithms are the 4th-order split-step method and (unoriginally) 4th-order Runge–Kutta, and my favourite method of evaluating spatial derivatives is moderate (6th or so) order finite differences. I describe the method of Fourier transforms for spatial derivatives, including its limitations compared to finite differences, and I show that the finite-element discrete-variable representation (FEDVR) is, despite appearances, actually less computationally efficient than simple finite differences for producing equally accurate solutions to the spatial Schrödinger equation. I also mention some methods of finding ground states and other stationary states. Throughout I emphasise that the efficacy of a numerical algorithm is strongly tied to its ability to be parallelised—such that a computation is tackled in pieces, distributed over multiple computer cores or cluster nodes. With the ubiquity of cluster computing and the end of rapid increases in single-core computer speed, this criterion is the single most important factor in the practical use of an algorithm in a large numerical simulation.

Section 3.6 presents a new numerical method (a modification of fourth order Runge–Kutta) for timestepping differential equations that allows for larger timesteps for certain problems. This chapter also contains original analysis of existing techniques: the calculation of the optimal submatrix size for splitting methods in Section 3.2.4 is new, as are the arguments in Section 3.2.4 regarding the applicability of split-step methods to nonlinear equations such as the Gross–Pitaevskii equations. The analysis in Section 3.4.3 comparing the finite-element discrete-variable representation method to finite differences with respect to stability and accuracy is new. I also express opinions on the relative merits of different algorithms, these are my own and are hopefully apparent from context.

3.1 From the abstract to the concrete: neglect, discretisation and representation

To numerically simulate a quantum mechanical system, one must evolve a state vector in time according to the Schrödinger equation:

$$i\hbar \frac{\mathrm{d}}{\mathrm{d}t} |\psi(t)\rangle = \hat{H}(t) |\psi(t)\rangle . \tag{3.1}$$

To do this on a computer, one must first decide which degrees of freedom are to be simulated. We necessarily neglect many degrees of freedom as a matter of course; which ones can be neglected is warranted by the specific situation and we do it so often we barely notice. For example, simulating a single component Bose–Einstein condensate entails neglecting the internal degrees of freedom of the atoms—as well as reducing the atom-light interaction to a simple potential such as an optical dipole trap or magnetic dipole interaction (neglecting the quantum degrees of freedom in the electromagnetic field). We may ignore one or more spatial degrees of freedom as well, say, if we are simulating an experiment in which the condensate is confined to one or two dimensions [48–50] by way of a tight trapping potential in one or more directions. Or, when simulating laser cooling (See for example Section 5.5), we may care very much about the electronic state of the atom, but treat its motional state classically. In these cases we are essentially imposing the assumption that the system will only occupy one state with respect to those degrees of freedom ignored (the condensate will remain in lowest excitation level in the direction of the tight trap; the atoms will remain in one specific Zeeman sublevel), or we are assuming those degrees of freedom can be treated classically (the electromagnetic field is well described by classical electromagnetism; the atoms' motional state is described well by Newtonian mechanics). Which degrees of freedom can be neglected and which cannot requires knowledge of the situation at hand, often informed by best-practices of the research community in question and ultimately justified by experiment.[1]

Once the degrees of freedom are known, one must decide on a basis in which to represent them concretely. The basis often cannot be complete, since for many degrees of freedom this would require an infinite number of basis states—for example the electronic state of an atom contains a countably infinite number of states, and a spatial wavefunction in free space has an uncountable number of states (one for each position in $\mathbb{R}^3$). For the internal state of an atom, therefore, we restrict ourselves to only the states we expect can become non-negligibly occupied, given the initial conditions and transitions involved. For example, at low temperature we can expect atoms to be almost completely in their electronic ground states, since energy gaps between ground and excited states are large compared to the thermal energy scale $k_B T$.[2]. We need only include the small number of excited states that might become occupied as a result of optical transitions present in the situation being simulated. This can still be a large number of states if one is studying Rydberg atoms [52, 53] or using ultra-fast (and therefore broad-band) laser pulses [54–56], but is otherwise fairly small. For example, including both the D_1 and D_2 lines of ^{87}Rb, with all hyperfine levels and Zeeman sublevels gives 32 states (Section 2.3).

For spatial degrees of freedom, we usually limit ourselves firstly to a finite region of space (we don't expect the Bose–Einstein condensate to have much probability amplitude on the Moon or anywhere else outside the vacuum system), and then we need to discretise the region of space remaining. To do this one can either discretise space on a grid, or use a set of orthogonal basis functions, and sometimes these can be equivalent, as we will soon see.

Once the degrees of freedom and basis vectors have been chosen, the state vector is then represented on a computer as an array of complex numbers, giving the coefficients of each basis vector required to represent a particular state vector. Matrix elements of the Hamiltonian in the same basis must be calculated, and the Schrödinger equation can

[1] A classic example in the cold atom community of neglected degrees of freedom leading to *disagreement* with experiment is the discovery of polarisation gradient cooling (PGC), the explanation for which requires consideration of Zeeman sublevels of the atoms. The experiment that discovered PGC [51] was designed to measure the effect of Doppler cooling, which does not involve Zeeman sublevels, and it was not until afterwards that theorists determined [29] that transitions between Zeeman sublevels cannot be neglected and indeed are crucial in explaining the lower than predicted temperatures observed.

[2] The D line of rubidium 87 has an energy gap of 0.6 eV, requiring a temperature of $\approx$ 650 K or higher in order for the Boltzmann factor $e^{-\frac{\Delta E}{k_B T}}$ describing the thermal occupation of the excited state to exceed 1×10^{-12}.

then be written:

$$i\hbar\frac{d}{dt}\langle n|\psi(t)\rangle = \sum_m \langle n|\hat{H}(t)|m\rangle\,\langle m|\psi(t)\rangle\,, \tag{3.2}$$

or in standard matrix/vector notation (without Dirac notation):

$$i\hbar\frac{d}{dt}\psi_n(t) = \sum_m H_{nm}(t)\psi_m(t) \tag{3.3}$$

$$\iff i\hbar\frac{d}{dt}\boldsymbol{\psi}(t) = H(t)\boldsymbol{\psi}(t), \tag{3.4}$$

where $\psi_n(t) = \langle n|\psi(t)\rangle$, $H_{nm}(t) = \langle n|\hat{H}(t)|m\rangle$ and $\boldsymbol{\psi}(t)$ and $H(t)$ are the vector and matrix with components and elements $\{\psi_n(t)\}$ and $\{H_{nm}\}$ respectively. This is now something very concrete that can be typed into a computer. Programming languages generally don't know about Dirac kets and operators, and so everything that is to be computed must be translated into matrices and vectors in specific bases. This may seem so obvious as to not be worth mentioning, but was nonetheless a stumbling block in my own experience of getting to grips with quantum mechanics. Once realising that every operator has a matrix representation in some basis, at least in principle (including differential operators), and that every ket is just a list of vector components in some basis (including ones representing spatial wavefunctions), similarly at least in principle, expressions dense in bras and kets become much more concrete as the reader has a feel for exactly how they would type it into a computer. Without a good feel for the mapping between operator algebra and the actual lists of numbers that these objects imply, doing quantum mechanics on paper can seem like an exercise in abstract mumbo-jumbo.

3.2 Solution to the Schrödinger equation by direct exponentiation

As an example of something seemingly abstract being more concrete than first appearances, it is sometimes said that the 'formal' solution to the Schrödinger equation (3.1) is:

$$|\psi(t)\rangle = e^{-\frac{i}{\hbar}\int_{t_0}^{t}\hat{H}(t')\,dt'}\,|\psi(t_0)\rangle\,. \tag{3.5}$$

Saying that this is the 'formal' solution rather than just 'the solution' is presumably intended to emphasise that the arithmetic operations involved in (3.5) might not make immediate sense for the types of mathematical objects they are operating on, and that we have to be careful in defining the operations such that they produce a result that is not only sensible, but also the solution to the Schrödinger equation. If both $\hat{H}(t)$ and $|\psi(t)\rangle$ were single-valued functions of time rather than an operator valued function of time (the values at different times of which don't necessarily commute) and a vector valued function of time, then we would have no problem. However, (3.5) as written with operators and vectors is ambiguous, and we need to elaborate on it in order to ensure it is correct. I will come back to this after considering a simpler case.

If the Hamiltonian is time-independent, then (3.5) reduces to

$$|\psi(t)\rangle = e^{-\frac{i}{\hbar}\hat{H}\Delta t}\,|\psi(t_0)\rangle\,, \tag{3.6}$$

where $\Delta t = t - t_0$. Given the matrix representation H of $\hat{H}$ and vector representation $\boldsymbol{\psi}(t_0)$ of $|\psi(t_0)\rangle$ in a particular basis, this can now be directly typed into a computer as the matrix multiplication:

$$\boldsymbol{\psi}(t) = U(t, t_0)\boldsymbol{\psi}(t_0), \tag{3.7}$$

where

$$U(t, t_0) = \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}H\Delta t} \tag{3.8}$$

is the (matrix representation of the) unitary evolution operator for time evolution from the initial time t_0 to time t, and is computed using a matrix exponential of $-\frac{\mathrm{i}}{\hbar}H\Delta t$. Exponentiation of matrices is defined via the Taylor series of the exponential function:

$$\mathrm{e}^A = \sum_{n=0}^{\infty} \frac{A^n}{n!}, \tag{3.9}$$

which reduces matrix exponentiation to the known operations of matrix multiplication and addition. However, any linear algebra programming library worth the bytes it occupies will have a matrix exponentiation function that should be used instead, as there are other methods of computing matrix exponentials that are more computationally efficient and numerically stable, such as the Padé approximant [57]. It should be noted that there is no known 'best' matrix exponential algorithm, all make compromises and perform poorly for certain types of matrices [58].

3.2.1 Matrix exponentiation by diagonalisation

Regardless of which method is used, matrix exponentiation is computationally expensive. It can be sped up however if a diagonalisation of H is known, since if

$$H = QDQ^\dagger, \tag{3.10}$$

where D is a diagonal matrix and Q is a unitary matrix[3], then

$$\mathrm{e}^{-\frac{\mathrm{i}}{\hbar}H\Delta t} = Q\mathrm{e}^{-\frac{\mathrm{i}}{\hbar}D\Delta t}Q^\dagger. \tag{3.11}$$

This is simple to evaluate because the exponentiation of a diagonal matrix can be performed by exponentiating each diagonal matrix element individually[4]

Even if a diagonalisation of H is not analytically known, numerically diagonalising H (using a linear algebra library function or otherwise) can form the basis for writing your own matrix exponentiation function, if needed. I found this necessary for efficiently exponentiating an array of matrices in Python, since the `scipy` and `numpy` scientific and numeric libraries at the present time lack matrix exponentiation functions that can act on arrays of matrices. Writing a `for` loop in an interpreted language such as Python to exponentiate the matrices individually in many cases is unacceptably slow, so for these cases[5] I use a function such as the one below:

[3] Note that the diagonals of D are the eigenvalues of H, and the columns of Q are its eigenvectors.

[4] The reason for this is clear from the Taylor series definition of matrix exponentiation, since matrix multiplication and addition can both be performed elementwise for diagonal matrices.

[5] Such as simulating the internal state of a large number of atoms, or evolving a spinor Bose–Einstein condensate by exponentiating the Zeeman Hamiltonian with a spatially varying magnetic field.

```python
import numpy as np
from numpy.linalg import eigh

def expmh(M):
    """compute exp(M), where M, shape (..., N, N) is an array of N by N
    Hermitian matrices, using the diagonalisation method. Made this function
    because scipy's expm can't take an array of matrices as input, it can only
    do one at a time."""

    # Diagonalise the matrices:
    evals, evecs = eigh(M)

    # Now we compute exp(M) = Q exp(D) Q^\dagger where Q is the matrix of
    # eigenvectors (as columns) and D is the diagonal matrix of eigenvalues:

    Q = evecs
    Q_dagger = Q.conj().swapaxes(-1, -2) # Only transpose the matrix dimensions
    exp_D_diags = np.exp(evals)

```

```
    # Compute the 3-term matrix product Q*exp_D_diags*Q_dagger using the
    # einsum function in order to specify which array axes of each array to
    # sum over:
    return np.einsum('...ik,...k,...kj->...ij', Q, exp_D_diags, Q_dagger)
```

Matrix diagonalisation (using singular value decomposition or QR decomposition) has computational time complexity $\mathcal{O}(n^3)$, where n is the number of rows/columns in the (square) matrix. Matrix multiplication is (in practice) $\mathcal{O}(n^3)$ and exponentiating a diagonal matrix is only $\mathcal{O}(n)$, so matrix exponentiation of a Hermitian matrix via numerical diagonalisation has total cost $\mathcal{O}(n^3)$. This compares to the Padé approximant, which is also $\mathcal{O}(n^3)$ [58]. So the numerical diagonalisation method is not any worse in terms of computational resources required.

On the other hand, if an analytic diagonalisation is already known, it would seem that exponentiation is just as slow, since the computational cost of matrix multiplication alone is the same order in n as that of numerical diagonalisation. This is true—so there are only constant factors to be saved in computer time by using an analytic diagonalisation in order to exponentiate a matrix using (3.11). However if one's aim—as is often the case—is to ultimately compute

$$\boldsymbol{\psi}(t) = \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}H\Delta t}\boldsymbol{\psi}(t_0), \tag{3.12}$$

for a specific $\boldsymbol{\psi}(t_0)$, then one needs only matrix-vector multiplications and not matrix-matrix multiplications in order to evaluate

$$\boldsymbol{\psi}(t) = U\mathrm{e}^{-\frac{\mathrm{i}}{\hbar}D\Delta t}U^\dagger\boldsymbol{\psi}(t_0), \tag{3.13}$$

from right-to-left, reducing the computational cost to $\mathcal{O}(n^2)$ compared to evaluating it left-to-right.

Whilst matrix exponentiation is a way to efficiently evolve systems with time-independent Hamiltonians, if you only exponentiate a matrix once, you don't much care about the time complexity of doing so. It is mostly of interest because the real power of these exponentiation methods is as a building block for methods of approximate solutions to the Schrödinger equation in the case of time *dependent* Hamiltonians, as we will see in Section 3.2.3.

3.2.2 The interaction picture

Although time-dependent Hamiltonians are much harder to deal with than time-independent ones, often a Hamiltonian can be decomposed into the sum a time-independent and a time-dependent part:

$$\hat{H}(t) = \hat{H}_0 + \hat{H}_1(t), \tag{3.14}$$

in this case, the Schrödinger equation can be transformed into a form that is often more computationally tractable and analytically useful, by treating part of the evolution due to the time-independent part analytically. This transformation is called an *interaction picture* [59, p. 317], or sometimes, particularly in atomic physics, a *rotating frame*, due to the fact that for a spin-$\frac{1}{2}$ system in a constant magnetic field, time evolution due to $\hat{H}_0$ entails literal rotation of the spin vector in three dimensional space.

To analytically remove the effect of time evolution due to $\hat{H}_0$, we write an ansatz for the state vector $|\psi(t)\rangle$ as an *interaction picture state vector* $|\psi_I(t)\rangle$ that has undergone time evolution due to $\hat{H}_0$ for time t:

$$|\psi(t)\rangle = \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}\hat{H}_0 t}|\psi_I(t)\rangle \tag{3.15}$$

Substituting this into the Schrödinger equation yields an equation of motion for the interaction picture state vector $|\psi_I(t)\rangle$:

$$i\hbar\frac{d}{dt}e^{-\frac{i}{\hbar}\hat{H}_0 t}|\psi_I(t)\rangle = \left[\hat{H}_0 + \hat{H}_1(t)\right]e^{-\frac{i}{\hbar}\hat{H}_0 t}|\psi_I(t)\rangle \tag{3.16}$$

$$\Rightarrow i\hbar\frac{d}{dt}|\psi_I(t)\rangle = e^{\frac{i}{\hbar}\hat{H}_0 t}\hat{H}_1(t)e^{-\frac{i}{\hbar}\hat{H}_0 t}|\psi_I(t)\rangle \tag{3.17}$$

$$\Rightarrow i\hbar\frac{d}{dt}|\psi_I(t)\rangle = \hat{H}_I(t)|\psi_I(t)\rangle, \tag{3.18}$$

where $\hat{H}_I(t) = e^{\frac{i}{\hbar}\hat{H}_0 t}\hat{H}_1(t)e^{-\frac{i}{\hbar}\hat{H}_0 t}$ is an *interaction picture Hamiltonian*. Thus the interaction picture defines a time-dependent unitary transformation, with the operator for transforming from the Schrödinger to interaction picture being:

$$\hat{U}_I(t) = e^{\frac{i}{\hbar}\hat{H}_0 t}, \tag{3.19}$$

$$\tag{3.20}$$

and state vectors and operators transforming as:

$$|\psi_I(t)\rangle = \hat{U}_I|\psi(t)\rangle \tag{3.21}$$

$$\hat{A}_I(t) = \hat{U}_I\hat{A}\hat{U}_I^\dagger, \tag{3.22}$$

which creates time dependence for operators that did not have time dependence in the Schrödinger picture.

Usually, calculations are performed in a basis in which $\hat{H}_0$ is diagonal, allowing the unitary transformation to be applied using only scalar exponentials rather than matrix exponentials:

$$\hat{U}(t) = \sum_n e^{\frac{i}{\hbar}E_n t}|n\rangle\langle n| \tag{3.23}$$

$$\Rightarrow \langle n|\psi_I(t)\rangle = e^{\frac{i}{\hbar}E_n t}\langle n|\psi(t)\rangle \tag{3.24}$$

where $\{|n\rangle\}$ and $\{E_n\}$ are the eigenkets and eigenvalues of $\hat{H}_0$.

Use of an interaction picture can remove short timescales from the evolution of a state vector, with the original Schrödinger picture state vectors recoverable via analytic transformations, though this is often not necessary since one often only cares about the magnitude of a component $\langle n|\psi\rangle$ of $|\psi\rangle$—which is preserved by the transformation (3.24)—and not its phase.

In atomic physics calculations an interaction picture can be used to remove the large energy scales (and hence small timescale) separating the three fine structure states of the D_2 line (Section 2.3.1), or it can be used to remove the entire hyperfine interaction Hamiltonian, and even both the hyperfine and Zeeman Hamiltonians (Sections 2.3.2 and 2.3.3) if a time-independent magnetic field is being used. This removes the necessity of small numerical timesteps needed by some numerical methods to resolve very fast phase oscillations due to these time-independent Hamiltonians, as well as aiding human interpretation of the results which would otherwise have more interesting or relevant phase changes drowned out by fast phase oscillations from $\hat{H}_0$.

In Section 3.6 I present a method that exploits the use of an *instantaneous* interaction picture, that is, an interaction picture whose time independent Hamiltonian is based on a time *dependent* Hamiltonian evaluated at a specific moment in time. In this way, the use of an interaction picture can be extended to aid calculations for a certain class of time-dependent Hamiltonians as well.

3.2.3 Time-ordered exponentials and time-ordered products

As hinted to earlier, the solution (3.5) is not the whole picture. It can only be taken at face value if the Hamiltonian at each moment in time commutes with itself at all other times (we will see shortly why this is). If it does—that is, if $[\hat{H}(t_1'), \hat{H}(t_2')] = 0$ for all $t_1', t_2' \in [t_0, t]$, then (3.5) is the solution to the Schrödinger equation, and once represented in a specific basis can be written

$$\psi(t) = U(t, t_0)\psi(t_0), \tag{3.25}$$

with

$$U(t, t_0) = \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}\int_{t_0}^{t} H(t')\,\mathrm{d}t'}, \tag{3.26}$$

i.e. with an integral in the exponent rather than a simple multiplication by a time interval. Since matrix addition can be performed elementwise, so can the integral in the exponent, yielding a matrix which once exponentiated will give the evolution operator $U(t, t_0)$ for the solution to the Schrödinger equation. If the Hamiltonian at each moment in time does not commute with itself at all other times, however, then the unitary evolution operator for the solution to the Schrödinger equation is instead given by the following *time-ordered exponential* [60, p. 193]:

$$U(t, t_0) = \mathcal{T}\left\{\mathrm{e}^{-\frac{\mathrm{i}}{\hbar}\int_{t_0}^{t} H(t')\,\mathrm{d}t'}\right\}. \tag{3.27}$$

In this expression, $\mathcal{T}$ denotes the *time-ordering operator*. The time ordering operator reorders terms within products that contain a time parameter (for us, the time parameter is the argument of the matrix-valued function H), such that the value of the time parameter is smallest in the rightmost term, largest in the leftmost term, and monotonically increasing right-to-left in between. For example:

$$\mathcal{T}\{H(4)H(1)H(2)H(5)H(3)\} = H(5)H(4)H(3)H(2)H(1). \tag{3.28}$$

We see now why the time-ordering can be neglected when $H(t)$ commutes with itself at all times. When it does, all possible reorderings of a product of copies $H(t)$ are already equal, and so a time-ordering operator leaves the actual value of the product unchanged.

Despite appearances, this time-ordered exponential is perfectly concretely defined via the definitions of all the operations involved that we have described so far, and can—with some effort—be typed into a computer and evaluated directly. Even though this is not how I have evaluated time-ordered exponentials in my simulations of atomic systems, I'll quickly elaborate on this just to emphasise the concreteness of all these operations.

"What products is $\mathcal{T}$ reordering?" you might ask, as (3.27) doesn't appear to contain any products of $H(t)$. On the contrary, it does, since exponentiation is defined by its Taylor series, and so

$$U(t, t_0) = 1 + \mathcal{T}\left\{\sum_{n=1}^{\infty} \frac{1}{n!}\left[-\frac{\mathrm{i}}{\hbar}\int_{t_0}^{t} H(t')\,\mathrm{d}t'\right]^n\right\} \tag{3.29}$$

$$= 1 + \sum_{n=1}^{\infty} \frac{1}{n!}\left(-\frac{\mathrm{i}}{\hbar}\right)^n \mathcal{T}\left\{\left[\int_{t_0}^{t} H(t')\,\mathrm{d}t'\right]^n\right\}. \tag{3.30}$$

Each term in this series contains the n^{th} power (and hence a product) of an integral of $H(t)$. The time ordering operator doesn't allow us to evaluate each term by computing the matrix integral once and then raising it to a power—to do so would violate time-ordering

since each integral involves evaluating $H(t)$ at all times. Instead we have to write each product of integrals as the integral of a product:

$$U(t,t_0) = 1 + \sum_{n=1}^{\infty} \frac{1}{n!}\left(-\frac{\mathrm{i}}{\hbar}\right)^n \int_{t_0}^{t} \mathrm{d}t_1' \int_{t_0}^{t} \mathrm{d}t_2' \cdots \int_{t_0}^{t} \mathrm{d}t_n' \cdots \mathcal{T}\left\{H(t_1')H(t_2')\cdots H(t_n')\right\}, \tag{3.31}$$

from which we can see exactly which product of matrices the time ordering operator is acting on.

Now we are close to seeing one might evaluate $U(t,t_0)$ numerically by summing each term in the Taylor series up to some order set by the required accuracy. For the n^{th} term, one needs to evaluate an n-dimensional integral over n time coordinates, with each coordinate having the same limits of integration. This can be computed in the usual way an integral is numerically computed,[6] with the minor change that each time the integrand is evaluated, the terms within it must be re-ordered to respect the required time-ordering. Alternatively, the integration region can be restricted to the region in which the terms are already time-ordered, and then the total integral inferred by symmetry, which gives:

$$U(t,t_0) = 1 + \sum_{n=1}^{\infty} \left(-\frac{\mathrm{i}}{\hbar}\right)^n \int_{t_0}^{t} \mathrm{d}t_n' \cdots \int_{t_0}^{t_3'} \mathrm{d}t_2' \int_{t_0}^{t_2'} \mathrm{d}t_1'\ H(t_n')\cdots H(t_2')H(t_1'). \tag{3.32}$$

[6] By sampling the integrand on a uniform grid, using a quadrature method, or Monte-Carlo integration [61] which is widely used for high dimensional integrals such as these.

This is now a perfectly concrete expression, with each term comprising an integral over an n-simplex[7] of a product of n matrices.

[7] A simplex is the generalisation of a triangle to higher dimensions, i.e. a 3-simplex is a tetrahedron.

This expression for the unitary evolution operator is called the Dyson series [62]. It is not generally used for time-dependent simulations, though it is the basis for time-dependent perturbation theory, and sees use in high energy physics [62, 63] for computing transition amplitudes between incoming and outgoing waves in scattering problems (in which U is called the S-matrix). In these problems, H is an interaction Hamiltonian containing terms for all particle interactions being considered. Accordingly, the integrand for the n^{th} term, being a product of n copies of H evaluated at different times, contains one term for each possible sequence of particle interactions. The integral itself can be considered a sum of transition amplitudes over all possible times that each interaction could have occurred. Indeed, each term in the Dyson series corresponds to a number of Feynman diagrams with n nodes [63].

The Dyson series isn't really suited to time-dependent simulations, though perturbation theory is useful for approximate analytics. For one, the series must be truncated at some point, and the result won't be a U that is actually unitary.[8]. Also, we are typically interested in the intermediate states, not just the final state of a system or an average transition rate, as one might want to compute in a scattering problem.

[8] Although unitarity is not often a strict requirement—we also frequently solve (3.3) directly with fourth order Runge–Kutta, which is also not unitary.

In any case, usually when solving the Schrödinger equation by exponentiation we use the following, alternate expression for a time-ordered exponential:

$$U(t,t_0) = \mathcal{T}\left\{\mathrm{e}^{-\frac{\mathrm{i}}{\hbar}\int_{t_0}^{t} H(t')\,\mathrm{d}t'}\right\} \tag{3.33}$$

$$= \lim_{N\to\infty} \prod_{n=N-1}^{0} \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}H(t_n)\Delta t}, \tag{3.34}$$

where here $\Delta t = (t - t_0)/N$ and $t_n = t_0 + n\Delta t$. Note that the product limits are written in the reverse of the usual order—this is important in order to produce terms with smaller n on the right and larger n on the left of the resulting product. You can convince yourself that (3.32) is equivalent to (3.34) this by replacing the integral in the exponent with a sum—as per the Riemann definition of an integral—and expanding the exponential according to its Taylor series. Expanding each exponential in (3.34) as a

Taylor Series and collecting terms of equal powers of H then reveals that the two Taylor series are identical.

In any case, (3.34) paints an intuitive picture of solving the Schrödinger equation: one evolves the initial state vector in time by evolving it according to constant Hamiltonians repeatedly over small time intervals[9]. This has the desirable property that all intermediate state vectors are computed at the intermediate steps, meaning one can study the dynamics of the system and not just obtain the final state. This is of course useful for comparison with experiments, plenty of which involve time-dependent data acquisition and not just post-mortem analysis of some evolution.

[9] This is the usual definition of the solution to a differential equation, which is why I prefer to think of the product formula (3.34) as the *definition* of the solution to the Schrödinger equation, and the time-ordered exponential as merely a shorthand notation for it.

Numerically, we can't actually take $N \to \infty$, or equivalently $\Delta t \to 0$, and so we instead choose a Δt smaller than the timescale of any time-dependence of H, and step through time using

$$\psi(t_{n+1}) = \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}H(t_n)\Delta t}\psi(t_n) + \mathcal{O}(\Delta t^2) \tag{3.35}$$

$$\Rightarrow \psi(t) = \left(\prod_{n=N-1}^{0} \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}H(t_n)\Delta t}\right)\psi(t_0) + \mathcal{O}(\Delta t). \tag{3.36}$$

Thus the case of a time-dependent Hamiltonian reduces to repeated application of the solution (3.7) for a time-independent Hamiltonian, and is (globally) accurate to order Δt. Note that the entire expression can be evaluated right-to-left to propagate the initial state vector in time without explicitly computing the overall unitary, which ensures the computational complexity is $\mathcal{O}(n^2)$ in the size of the system (when the exponentiation is performed via an analytic or pre-computed diagonalisation) rather than $\mathcal{O}(n^3)$.

3.2.4 The operator product/split-step method

Here I'll review decompositions similar to (3.35), but which use variable timesteps to achieve an accuracy to higher order in Δt. I'll present them at the same time as addressing another problem, which is that Hamiltonians are often not in a simple enough form to be exponentiated efficiently at all, making (3.35) difficult to evaluate. Often Hamiltonians are a sum of non-commuting operators (such as kinetic and potential terms), with time dependence such that any diagonalisation of the overall Hamiltonian at one point in time will not diagonalise it at another point in time, with the system size large enough for numerical diagonalisation to be prohibitively expensive. In these cases, we can use methods called *split-step* or *operator product* methods, which allow one to approximately exponentiate the entire Hamiltonian based on having exact diagonalisations of its component terms. In the case of time-dependent Hamiltonians, this allows us to avoid rediagonalising at every timestep whenever the time-dependence can be expressed as scalar coefficients multiplying time-independent operators:

$$\hat{H}(t) = \alpha(t)\hat{H}_1 + \beta(t)\hat{H}_2 + \cdots, \tag{3.37}$$

since multiplication of a matrix by a scalar merely scales its eigenvalues, leaving its eigenbasis unchanged.

There is little downside to having an only approximate exponentiation of the Hamiltonian when the timestepping is already only approximate, so long as we ensure that neither source of error is much greater than the other. To this end I'll show split-step methods that have (global) error $\mathcal{O}(\Delta t)$, $\mathcal{O}(\Delta t^2)$ and $\mathcal{O}(\Delta t^4)$. In Section 3.4, we'll see how this method applies to the case of a spatial wavefunction obeying the Gross–Pitaevskii equation.

First order split-step

Say we have (the matrix representation of) a Hamiltonian that is the sum of two non-commuting terms:[10]

$$H(t) = H_1(t) + H_2(t). \tag{3.38}$$

[10] From here on, component terms of the Hamiltonian will be written with general time dependence, even though it is understood that these methods are mostly useful for the case where that time dependence can be written as scalar coefficients multiplying otherwise time-independent matrices.

The unitary for the solution to the Schrödinger equation is as before:

$$U(t, t_0) = \mathcal{T}\left\{e^{-\frac{i}{\hbar}\int_{t_0}^{t} H(t')\,dt'}\right\}, \tag{3.39}$$

which, without loss of exactness, we can split into N equal time intervals of size Δt and write

$$U(t, t_0) = \prod_{n=N-1}^{0} U(t_{n+1}, t_n), \tag{3.40}$$

where again $t_n = t_0 + n\Delta t$ and $\Delta t = (t - t_0)/N$, and where

$$U(t_{n+1}, t_n) = \mathcal{T}\left\{e^{-\frac{i}{\hbar}\int_{t_n}^{t_{n+1}} H(t')\,dt'}\right\}. \tag{3.41}$$

Evaluating the integral using a one-point rectangle rule gives

$$U(t_{n+1}, t_n) = e^{-\frac{i}{\hbar}H(t_n)\Delta t + \mathcal{O}(\Delta t^2)}, \tag{3.42}$$

in which we were able to drop the time ordering operator because $H(t)$ is only evaluated at a single time. Using the Taylor series definition of the exponential, we can take the error term out of the exponent and write

$$U(t_{n+1}, t_n) = e^{-\frac{i}{\hbar}H(t_n)\Delta t} + \mathcal{O}(\Delta t^2). \tag{3.43}$$

So far all we've done is justify (3.35). Now we'll acknowledge that H is a sum of two terms and write:

$$U(t_{n+1}, t_n) = e^{-\frac{i}{\hbar}(H_1(t_n)+H_2(t_n))\Delta t} + \mathcal{O}(\Delta t^2). \tag{3.44}$$

Using the Baker–Campbell–Hausdorff formula [60, p. 158], we can expand the exponential as:

$$e^{-\frac{i}{\hbar}(H_1(t_n)+H_2(t_n))\Delta t} = e^{-\frac{i}{\hbar}H_1(t_n)\Delta t}e^{-\frac{i}{\hbar}H_2(t_n)\Delta t}e^{\frac{1}{2\hbar^2}\left[H_1(t_n),H_2(t_n)\right]\Delta t^2 + \mathcal{O}(\Delta t^3)} \tag{3.45}$$

$$\Rightarrow U(t_{n+1}, t_n) = e^{-\frac{i}{\hbar}H_1(t_n)\Delta t}e^{-\frac{i}{\hbar}H_2(t_n)\Delta t} + \mathcal{O}(\Delta t^2). \tag{3.46}$$

So we see that the 'commutation error' in (3.46) caused by treating the two terms as if they commute is of the same order in Δt as the 'integration error' in (3.43) caused by treating the overall Hamiltonian as time-independent over one timestep, resulting in a method with local error $\mathcal{O}(\Delta t^2)$ and global error $\mathcal{O}(\Delta t)$; the first order split-step method:

$$U(t, t_0) = \prod_{n=N-1}^{0} U_1(t_{n+1}, t_n) + \mathcal{O}(\Delta t), \tag{3.47}$$

where

$$U_1(t_{n+1}, t_n) = e^{-\frac{i}{\hbar}H_1(t_n)\Delta t}e^{-\frac{i}{\hbar}H_2(t_n)\Delta t}. \tag{3.48}$$

Using the diagonalisation method of matrix exponentiation discussed in Section 3.2.1, this unitary U_1 can be used to step a state vector through time with only matrix-vector multiplications and scalar exponentiation, where separate diagonalisations of H_1 and H_2 are either analytically known or pre-computed, even though a diagonalisation of the total Hamiltonian H may not be feasible.

Crucially, if H_1 and H_2 act on different subspaces of the overall Hilbert space, with respective dimensionalities n_1 and n_2, that is, $H_1(t) = \tilde{H}_1(t) \otimes \mathbb{I}_{n_2}$ and $H_2(t) = \mathbb{I}_{n_1} \otimes \tilde{H}_2(t)$, where $\mathbb{I}_m$ is the $m \times m$ identity matrix, then the matrix-vector products involved in applying U_1 to a state vector can be much faster ($\mathcal{O}(n_1^2 n_2 + n_1 n_2^2)$ rather than $\mathcal{O}(n_1^2 n_2^2)$) than if the Hamiltonian weren't split into two terms, even if an analytic diagonalisation of the total Hamiltonian were available:

$$U_1(t_{n+1}, t_n) = \left(\mathrm{e}^{-\frac{\mathrm{i}}{\hbar}\tilde{H}_1(t_n)\Delta t} \otimes \mathbb{I}_{n_2}\right)\left(\mathbb{I}_{n_1} \otimes \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}\tilde{H}_2(t_n)\Delta t}\right). \tag{3.49}$$

By applying (3.46) recursively for the case where either H_1 or H_2 is itself the sum of two further terms, (3.48) immediately generalises to the case where H is a sum of an arbitrary number of non-commuting terms:

$$U_1(t_{n+1}, t_n) = \prod_{m=1}^{M} \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}H_m(t_n)\Delta t}, \tag{3.50}$$

where $H(t) = \sum_{m=1}^{M} H_m(t)$.

Second and fourth order split-step

By using a two-point trapezoid rule for the integral in (3.41), evaluating $H(t)$ at the beginning and end of the timestep, one can instead obtain an integration error of $\mathcal{O}(\Delta t^3)$ per step:

$$U(t_{n+1}, t_n) = \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}(H(t_n)+H(t_{n+1}))\frac{\Delta t}{2}} + \mathcal{O}(\Delta t^3) \tag{3.51}$$

$$= \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}(H_1(t_n)+H_2(t_n)+H_1(t_{n+1})+H_2(t_{n+1}))\frac{\Delta t}{2}} + \mathcal{O}(\Delta t^3). \tag{3.52}$$

Then, applying the Baker–Campbell–Hausdorff formula once more, and replacing $H_1(t_{n+1})$ (and similarly for $H_2(t_{n+1})$) wherever it appears in a commutator with the Taylor series $H_1(t_n) + \dot{H}_1(t_n)\Delta t + \mathcal{O}(\Delta t^2)$, one can show that if the individual exponentials are ordered in the following way, then the remaining commutation error is also $\mathcal{O}(\Delta t^3)$:

$$U(t_{n+1}, t_n) = \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}H_2(t_{n+1})\frac{\Delta t}{2}} \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}H_1(t_{n+1})\frac{\Delta t}{2}} \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}H_1(t_n)\frac{\Delta t}{2}} \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}H_2(t_n)\frac{\Delta t}{2}} + \mathcal{O}(\Delta t^3). \tag{3.53}$$

This gives the second order split-step method:

$$U(t, t_0) = \prod_{n=N-1}^{0} U_2(t_{n+1}, t_n) + \mathcal{O}(\Delta t^2), \tag{3.54}$$

where

$$U_2(t_{n+1}, t_n) = \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}H_2(t_{n+1})\frac{\Delta t}{2}} \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}H_1(t_{n+1})\frac{\Delta t}{2}} \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}H_1(t_n)\frac{\Delta t}{2}} \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}H_2(t_n)\frac{\Delta t}{2}}, \tag{3.55}$$

or, for an arbitrary number of terms in the Hamiltonian,

$$U_2(t_{n+1}, t_n) = \left(\prod_{m=M}^{1} \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}H_m(t_{n+1})\frac{\Delta t}{2}}\right)\left(\prod_{m=1}^{M} \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}H_m(t_n)\frac{\Delta t}{2}}\right). \tag{3.56}$$

U_2 can be concisely written in terms of U_1 as:

$$U_2(t_{n+1}, t_n) = U_1^\dagger(t_n + \tfrac{\Delta t}{2}, t_{n+1}) U_1(t_n + \tfrac{\Delta t}{2}, t_n), \tag{3.57}$$

which is to say it is simply two applications of U_1, each of duration half a total timestep, and with the multiplication order of the exponentials reversed in one compared to the other. Two shortcuts are immediately apparent when computing U_2 or its action on a vector: firstly, if there is a time-independent term in the Hamiltonian, it should be assigned to H_1, so that the innermost two exponentials in (3.55) can be collapsed into one; secondly the final exponential of each timestep is identical to the first exponential of the next timestep, and so these can also be collapsed together (being split apart only at points in time when one wishes to sample the state vector).

The fourth order split-step method is much more difficult to derive, with a large number of commutators needing to cancel exactly to ensure the local error cancels out up to fourth order in Δt. It can be stated in terms of the second order split-step method as [64, p. 7; 65,66]:

$$U(t, t_0) = \prod_{n=N-1}^{0} U_4(t_{n+1}, t_n) + \mathcal{O}(\Delta t^4), \tag{3.58}$$

where

$$\begin{aligned} U_4(t_{n+1}, t_n) = \; & U_2(t_{n+1}, t_{n+1} - p\Delta t) \\ & \times U_2(t_{n+1} - p\Delta t, t_{n+1} - 2p\Delta t) \\ & \times U_2(t_{n+1} - 2p\Delta t, t_n + 2p\Delta t) \\ & \times U_2(t_n + 2p\Delta t, t_n + p\Delta t) \\ & \times U_2(t_n + p\Delta t, t_n), \end{aligned} \tag{3.59}$$

where $p = 1/(4 - 4^{1/3})$. U_4 comprises five applications of U_2 with timesteps $p\Delta t$, $p\Delta t$, $(1-4p)\Delta t$, $p\Delta t$, and $p\Delta t$ respectively. The innermost timestep is backwards in time, since $(1 - 4p) < 0$. But this is no problem, one simply evaluates the expression for U_2 with a negative timestep exactly as written, so long as one reads Δt when written within the above expressions for $U_1(t_f, t_i)$ and $U_2(t_f, t_i)$ as referring to $t_f - t_i$, i.e. the difference between the two arguments to the expression, not its absolute value, and not to the timestep of the method in which it is embedded.

Parallelisability and other speedups for banded matrices

Expressions such as (3.55) don't at first glance appear particularly easy to parallelise, that is, to be evaluated in such a way as to leverage the computing power of multiple CPU cores, GPU cores, or multiple computers. A series of Hamiltonian terms must be exponentiated one by one and multiplied together. Whilst each exponential factor could be evaluated independently, and then all of them multiplied together before being applied to a state vector, this explicit construction of U is very costly compared to merely computing its action on a particular state vector, since the latter allows each term to act only in its specific subspace of the overall Hilbert space, and avoids having to pay the $\mathcal{O}(n^3)$ cost of matrix-matrix multiplication.

When one or more terms in the Hamiltonian has a matrix representation which is banded however, that is, all its entries further than a finite number of elements away from the main diagonal are zero, then those terms may be written as a sum of block diagonal matrices, for example:

$$
A = \begin{bmatrix}
c & d & e & & & & & & & & & & & & & \\
b & c & d & e & & & & & & & & & & & & \\
a & b & c & d & e & & & & & & & & & & & \\
 & a & b & c & d & e & & & & & & & & & & \\
 & & a & b & c & d & e & & & & & & & & & \\
 & & & a & b & c & d & e & & & & & & & & \\
 & & & & a & b & c & d & e & & & & & & & \\
 & & & & & a & b & c & d & e & & & & & & \\
 & & & & & & a & b & c & d & e & & & & & \\
 & & & & & & & a & b & c & d & e & & & & \\
 & & & & & & & & a & b & c & d & e & & & \\
 & & & & & & & & & a & b & c & d & e & & \\
 & & & & & & & & & & a & b & c & d & e & \\
 & & & & & & & & & & & a & b & c & d & \\
 & & & & & & & & & & & & a & b & c &
\end{bmatrix}
$$

$$
= \begin{bmatrix}
\cdot & \cdot & \cdot & & & & & & & & & & & & & \\
\cdot & \cdot & \cdot & \cdot & & & & & & & & & & & & \\
\cdot & \cdot & \cdot & \cdot & \cdot & & & & & & & & & & & \\
 & \cdot & \cdot & \cdot & \cdot & \cdot & & & & & & & & & & \\
 & & \cdot & \cdot & c/2 & d/2 & e & & & & & & & & & \\
 & & \cdot & b/2 & c/2 & d & e & & & & & & & & & \\
 & & a & b & c & d & e & & & & & & & & & \\
 & & & a & b & c & d & e & & & & & & & & \\
 & & & & a & b & c/2 & d/2 & \cdot & & & & & & & \\
 & & & & & a & b/2 & c/2 & \cdot & \cdot & & & & & & \\
 & & & & & & \cdot & \cdot & \cdot & \cdot & \cdot & & & & & \\
 & & & & & & & \cdot & \cdot & \cdot & \cdot & \cdot & & & & \\
 & & & & & & & & \cdot & \cdot & c/2 & d/2 & e & & & \\
 & & & & & & & & & \cdot & b/2 & c/2 & d & e & & \\
 & & & & & & & & & & a & b & c & d & & \\
 & & & & & & & & & & & a & b & c &
\end{bmatrix}
+ \begin{bmatrix}
c & d & e & & & & & & & & & & & & & \\
b & c & d & e & & & & & & & & & & & & \\
a & b & c & d & e & & & & & & & & & & & \\
 & a & b & c & d & e & & & & & & & & & & \\
 & & a & b & c/2 & d/2 & \cdot & & & & & & & & & \\
 & & & a & b/2 & c/2 & \cdot & \cdot & & & & & & & & \\
 & & & & \cdot & \cdot & \cdot & \cdot & & & & & & & & \\
 & & & & & \cdot & \cdot & \cdot & \cdot & & & & & & & \\
 & & & & & & \cdot & \cdot & c/2 & d/2 & e & & & & & \\
 & & & & & & & \cdot & b/2 & c/2 & d & e & & & & \\
 & & & & & & & & a & b & c & d & e & & & \\
 & & & & & & & & & a & b & c/2 & d/2 & \cdot & & \\
 & & & & & & & & & & a & b/2 & c/2 & \cdot & \cdot & \\
 & & & & & & & & & & & \cdot & \cdot & \cdot & \cdot & \\
 & & & & & & & & & & & & \cdot & \cdot & \cdot &
\end{bmatrix}
$$

$$
= B + C, \tag{3.60}
$$

where zero-valued matrix elements outside the band are omitted, and those within the band replaced with dots. In this example, we first take the original 16×16 matrix A, and create four 4×4 non-overlapping submatrices whose diagonals lie along A's main diagonal. These submatrices don't quite cover all of A's non-zero elements, however, so we expand each submatrix (except the final one) from its bottom-right by two elements, making it a 6×6 matrix. The submatrices are no longer non-overlapping, so we take every second one and put it in a matrix B, every other one and put it in a matrix C such that B and C are both block diagonal, divide the elements they have in common by two so we don't double count them, and then declare that $A = B + C$. In the general case, the amount of overlap between the submatrices required to encompass all of A is equal to the bandwidth b of A (in this case $b = 2$ since A has two non-main diagonals on each side of its main diagonal).

Having split a term in the Hamiltonian into two terms, we can simply apply the split operator method as normal, with the same order accuracy in Δt as before. But now the matrices that we wish to exponentiate and have act on a vector are *block diagonal*. This means that the exponentiation of each block can be applied (using the diagonalisation method) separately to the vector, independently of each other block. This enables the application of the exponentials to be performed in parallel—each block submatrix modifies different elements of the vector. So one might store different parts of the state vector on different nodes on a cluster computer, and compute $\mathrm{e}^{-\frac{\mathrm{i}}{\hbar}C\Delta t}\boldsymbol{\psi}$ in parallel. Then some data exchange between nodes would be necessary before applying $\mathrm{e}^{-\frac{\mathrm{i}}{\hbar}B\Delta t}$ to the result (See Figure 3.1 for a schematic of how this works).

Whilst this specific banded matrix A is small for the sake of example, in general of course one might have a matrix of any size, allowing for B and C to contain more than two submatrices each. The submatrices have a minimum size of twice the bandwidth b of A, in order to cover all elements of A whilst only sharing elements with their nearest neighbour submatrices (any smaller and they would share elements with their next nearest neighbour submatrices as well, complicating things somewhat). But the only maximum is the size of A itself. So what is the optimal submatrix size? Although the split-step method is still accurate to the same order in Δt no matter how many pieces we split a banded matrix into,

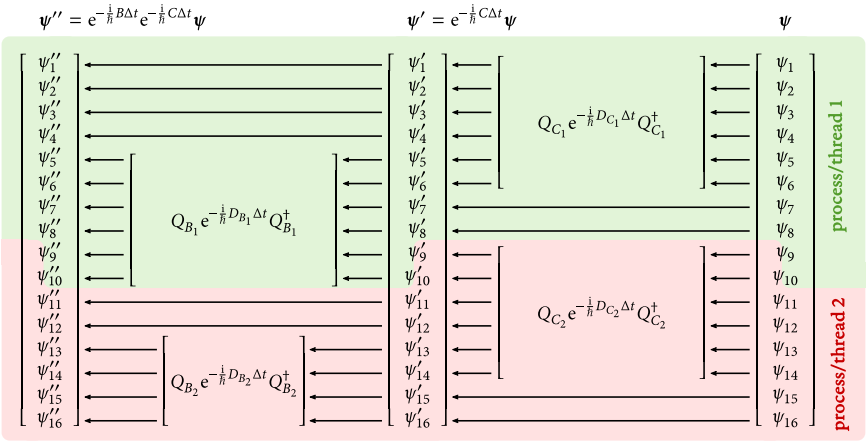

Figure 3.1: Schematic of data flow for parallel split-step method. Computation proceeds right to left. To compute the action of $e^{-\frac{i}{\hbar}B\Delta t}e^{-\frac{i}{\hbar}C\Delta t}$ on a vector $\boldsymbol{\psi}$, one can treat the submatrices B_1, B_2, C_1 and C_2, of the block-diagonal matrices B and C separately. First, the exponentiation of C can be applied to ψ by applying the exponentiations of C_1 and C_2 to ψ. If diagonalisations $C_1 = Q_{C_1}D_{C_1}Q^\dagger_{C_1}$ and $C_2 = Q_{C_2}D_{C_2}Q^\dagger_{C_2}$ are known, then the two operations can be applied as a series of matrix-vector multiplications and the exponentiation of diagonal matrices. Because the two submatrices are non-overlapping, they can be applied to the vector completely independently in separate computer processes, CPU or GPU threads, or cluster nodes. The exponentiation of B can then be applied to the resulting intermediate vector $\boldsymbol{\psi}'$, similarly via two independent operations acting on non-overlapping elements of ψ'. However, because each submatrix of C overlaps each submatrix of B by $b = 2$ elements on either side (b being the bandwidth of the original matrix $A = B + C$), threads/processes must share these elements (here the ninth and tenth elements), sending them to each other whenever they have been updated. For simplicity this example has two compute threads and two submatrices in each term B and C, but in general there can be any number of either. When a single thread must apply the exponentiation of multiple submatrices to the state vector, it is advantageous for it to first compute the result of any submatrix whose output is required by another thread, then all submatrices that are independent of other threads, and finally any submatrix which requires input from another thread. In this way, data required by other threads can be sent as early as possible, and data needed from other threads called upon as late as possible, minimising the time that threads are waiting for each other whilst there is useful work to be done.

we clearly introduce additional 'commutation error' every time we split A into additional submatrices. So one might think that the number of pieces ought to be minimised, and hence the submatrix size maximised. With regard to this, one might decide to split A into $2n_{\text{threads}}$ submatrices (where n_{threads} is the number of independent computational threads available), half of which will reside in B and half in C. This minimises the extra commutation error subject to the constraint that all threads are put to use—splitting A into yet smaller pieces within one computational thread will only yield unnecessary additional error.

Is this the best option? No. Despite the extra commutation error, there is additional benefit to splitting A into more submatrices than required for parallelisation. Let s be the 'nominal' size of each submatrix, that is, the size of the corresponding *non-overlapping* submatrices prior to expanding each one along the diagonal (creating overlap) by a number of elements equal to the bandwidth b. The computational cost of the matrix-vector

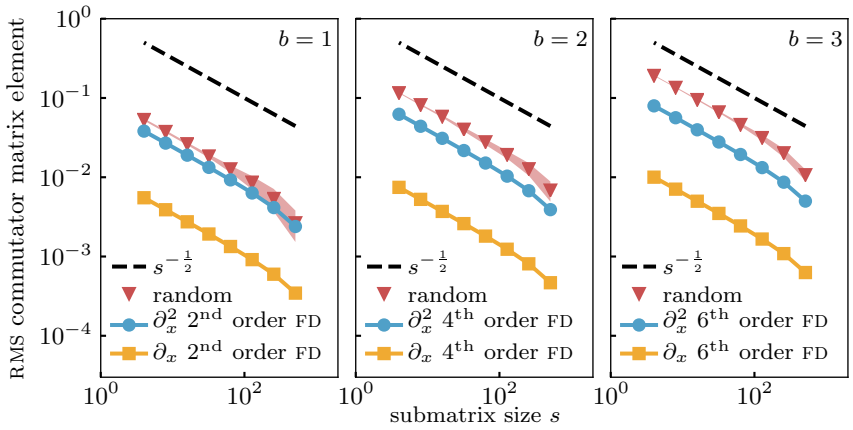

Figure 3.2: Numerical experiment to determine how the commutation error scales with the submatrix size when splitting certain banded matrices into the sum of block-diagonal matrices and using a split-operator method to exponentiate the sum. From left to right, matrices with bandwidth $b = 1$, $b = 1$ and $b = 2$ are considered. For each bandwidth, the matrices for the finite difference schemes of the order resulting in that bandwidth for first and second derivatives are considered, as well as random matrices of the given bandwidth. The random matrices have real and imaginary parts of each element within the band drawn from standard normal distributions. All matrices are 1024×1024. Calculations with random matrices were performed on 20 independent random matrices, and the mean and standard deviation (shaded) of the results plotted. The error metric is the RMS element of the commutator $[B, C]$, where the original matrix A is split into the sum of block-diagonal matrices $B + C$ using a submatrix size of s (described in-text). The RMS error in a vector propagated with the split-operator method is proportional this error metric. The result in all cases is that the commutator error, and therefore the error in the split-operator method scales with the submatrix size as $s^{-\frac{1}{2}}$.

multiplications for computing the action of $e^{-\frac{i}{\hbar}B\Delta t}e^{-\frac{i}{\hbar}C\Delta t}$ on a vector is then $\mathcal{O}((s+b)^2)$ per submatrix, since $s + b$ is the size of the submatrices in terms of their nominal size and the bandwidth. The total number of submatrices required to cover A is ns^{-1}, where n is the size of A, and so the total cost of applying the exponentiations of all submatrices to a vector is $\mathcal{O}(ns^{-1}(s+b)^2)$. The cost *per unit time* of simulation is then $\mathcal{O}(n\Delta t^{-1}s^{-1}(s+b)^2)$. Here we see that splitting into smaller submatrices is desirable from the point of view of speed: because matrix-vector multiplication runs in quadratic time, a larger number of smaller matrices can be multiplied by vectors faster than a smaller number of larger matrices.

But the more submatrices, the more commutation error. Extra error can be made up for by making the timestep smaller, which increases the cost per unit time once more. So is it worth it? The extra commutation error from splitting up A into more and more pieces in general depends on the form of A. I performed a small numerical experiment to see how the commutator $[B, C]$ (which the commutation error is proportional to) scales with s for random banded matrices, as well as those corresponding to 2nd, 4th and 6th order finite differences for first and second derivatives. The result in all cases was that the commutator scaled as $s^{-\frac{1}{2}}$ (see Figure 3.2).

Back to our question—does decreasing the timestep size Δt to compensate for the additional commutation error result in more, or less computational cost per unit time than if we hadn't split A into more pieces than required for parallelisation? Taking into account the nominal submatrix size s and the method's error in terms of Δt, the total

error of integrating using the split-step method (assuming the $s^{-\frac{1}{2}}$ commutator scaling holds) is $\mathcal{O}(\Delta t^a s^{-\frac{1}{2}})$, where a is the order in Δt of the accuracy of the specific split-step method used (1, 2, or 4 for those I've discussed). Using these two pieces of information: the total error, and the total cost per unit time; we can now answer the question "What value of s minimises the computational cost per unit time, assuming constant error?".

For constant error we set $\Delta t^a s^{-\frac{1}{2}} \propto 1$ and get that the computational cost per unit time at constant error is $\mathcal{O}(ns^{-\frac{1}{2a}-1}(s+b)^2)$. For $a > \frac{1}{2}$, this expression has a minimum at $s = b$, which is the smallest possible submatrix size in any case. For our example banded matrix A, decomposition into the smallest possible submatrices looks like this:

$$A = \begin{bmatrix}
c & d & e & & & & & & & & & & & & & & & & & \\
b & c & d & e & & & & & & & & & & & & & & & & \\
a & b & c & d & e & & & & & & & & & & & & & & & \\
 & a & b & c & d & e & & & & & & & & & & & & & & \\
 & & a & b & c & d & e & & & & & & & & & & & & & \\
 & & & a & b & c & d & e & & & & & & & & & & & & \\
 & & & & a & b & c & d & e & & & & & & & & & & & \\
 & & & & & a & b & c & d & e & & & & & & & & & & \\
 & & & & & & a & b & c & d & e & & & & & & & & & \\
 & & & & & & & a & b & c & d & e & & & & & & & & \\
 & & & & & & & & a & b & c & d & e & & & & & & & \\
 & & & & & & & & & a & b & c & d & e & & & & & & \\
 & & & & & & & & & & a & b & c & d & e & & & & & \\
 & & & & & & & & & & & a & b & c & d & e & & & & \\
 & & & & & & & & & & & & a & b & c & d & e & & & \\
 & & & & & & & & & & & & & a & b & c & d & e & & \\
 & & & & & & & & & & & & & & a & b & c & d & e & \\
 & & & & & & & & & & & & & & & a & b & c & d & \\
 & & & & & & & & & & & & & & & & a & b & c &
\end{bmatrix}$$

$$= \begin{bmatrix}
c & d & e & & & & & & & & & & & & & & & & & \\
b & c & d & e & & & & & & & & & & & & & & & & \\
a & b & c/2 & d/2 & \cdot & & & & & & & & & & & & & & & \\
 & a & b/2 & c/2 & \cdot & \cdot & & & & & & & & & & & & & & \\
 & & \cdot & \cdot & c/2 & d/2 & e & & & & & & & & & & & & & \\
 & & & \cdot & b/2 & c/2 & d & e & & & & & & & & & & & & \\
 & & & & a & b & c/2 & d/2 & \cdot & & & & & & & & & & & \\
 & & & & & a & b/2 & c/2 & \cdot & \cdot & & & & & & & & & & \\
 & & & & & & \cdot & \cdot & c/2 & d/2 & e & & & & & & & & & \\
 & & & & & & & \cdot & b/2 & c/2 & d & e & & & & & & & & \\
 & & & & & & & & a & b & c/2 & d/2 & \cdot & & & & & & & \\
 & & & & & & & & & a & b/2 & c/2 & \cdot & \cdot & & & & & & \\
 & & & & & & & & & & \cdot & \cdot & c/2 & d/2 & e & & & & & \\
 & & & & & & & & & & & \cdot & b/2 & c/2 & d & e & & & & \\
 & & & & & & & & & & & & a & b & c/2 & d/2 & \cdot & & & \\
 & & & & & & & & & & & & & a & b/2 & c/2 & \cdot & \cdot & & \\
 & & & & & & & & & & & & & & \cdot & \cdot & c/2 & d/2 & e & \\
 & & & & & & & & & & & & & & & \cdot & b/2 & c/2 & d & e \\
 & & & & & & & & & & & & & & & & a & b & c & d \\
 & & & & & & & & & & & & & & & & & a & b & c
\end{bmatrix} + \begin{bmatrix}
\cdot & \cdot & \cdot & & & & & & & & & & & & & & & & & \\
\cdot & \cdot & \cdot & \cdot & & & & & & & & & & & & & & & & \\
\cdot & \cdot & c/2 & d/2 & e & & & & & & & & & & & & & & & \\
 & \cdot & b/2 & c/2 & d & e & & & & & & & & & & & & & & \\
 & & a & b & c/2 & d/2 & \cdot & & & & & & & & & & & & & \\
 & & & a & b/2 & c/2 & \cdot & \cdot & & & & & & & & & & & & \\
 & & & & \cdot & \cdot & c/2 & d/2 & e & & & & & & & & & & & \\
 & & & & & \cdot & b/2 & c/2 & d & e & & & & & & & & & & \\
 & & & & & & a & b & c/2 & d/2 & \cdot & & & & & & & & & \\
 & & & & & & & a & b/2 & c/2 & \cdot & \cdot & & & & & & & & \\
 & & & & & & & & \cdot & \cdot & c/2 & d/2 & e & & & & & & & \\
 & & & & & & & & & \cdot & b/2 & c/2 & d & e & & & & & & \\
 & & & & & & & & & & a & b & c/2 & d/2 & \cdot & & & & & \\
 & & & & & & & & & & & a & b/2 & c/2 & \cdot & \cdot & & & & \\
 & & & & & & & & & & & & \cdot & \cdot & c/2 & d/2 & e & & & \\
 & & & & & & & & & & & & & \cdot & b/2 & c/2 & d & e & & \\
 & & & & & & & & & & & & & & a & b & c/2 & d/2 & \cdot & \\
 & & & & & & & & & & & & & & & a & b/2 & c/2 & \cdot & \cdot \\
 & & & & & & & & & & & & & & & & \cdot & \cdot & \cdot & \cdot \\
 & & & & & & & & & & & & & & & & & \cdot & \cdot & \cdot
\end{bmatrix}. \tag{3.61}$$

So the conclusion is: use the smallest submatrices possible when decomposing a banded matrix into a sum of two block-diagonal matrices. The decrease in computational costs, even when the increased error is compensated for by a smaller timestep, is worth it.

Limitations and nonlinearity

The split-step method is quite powerful and general. It allows you to approximately decompose the exponentiation of a Hamiltonian into exponentiations of its component terms, in the subspaces that they act on, and avoids exponentiating large banded matrices; saving on computing power immensely compared to exponentiating the full Hamiltonian. It is unitary (when the Hamiltonian is actually Hermitian), and stable—that is, unlike Runge–Kutta methods, the method's truncation error does not grow without limit as the simulation proceeds but is bounded (disregarding floating point rounding error, which is much smaller than the truncation error of either method) [67]. This is extremely appealing. And, whilst higher order split-step methods quickly become unwieldy [67], fourth order accuracy is quite acceptable for many problems.

Applying all the tricks described in the above sections results in $\mathcal{O}(\Delta t)$, $\mathcal{O}(\Delta t^2)$ and $\mathcal{O}(\Delta t^4)$ accurate timestepping methods for solving the Schrödinger equation with total computational cost scaling as $\mathcal{O}(n \sum_i b_i)$, where $n = \prod_i n_i$ (for a product space of subspaces with dimensionalities $\{n_i\}$) is the dimensionality of the total Hilbert space, and $\{b_i\}$ are the bandwidths of the matrix representations of each term in the Hamiltonian in the chosen basis.[11] Furthermore, the method is efficiently parallelisable, provided the

[11] Now that we are here, we can finally say something about the best basis for simulating in: to minimise computational costs, the best basis is the one that minimises precisely this sum of bandwidths. The exception to this is when fast Fourier transforms are involved, which I discuss later.

maximum size-to-bandwidth ratio $\max_i(n_i/b_i)$ of the terms in the Hamiltonian is much larger than the number of parallel computing threads available. Although the constant factors that big-O notation neglects may not be optimal, this scaling would seem to be the best one could hope for—for each of the n elements in the state vector one must consider the $\sum_i b_i$ elements (including itself) that the Hamiltonian couples it with, and no more.[12]

[12] Assuming the banded matrices are otherwise dense within their band—further improvements would be possible if some elements within the band were always zero.

The main downside of this otherwise excellent method of exponentiating Hamiltonians is that the evolution modelled must actually be described by a linear system of equations. One cannot add arbitrary terms and nonlinear operators to the Hamiltonian, as the split-step method requires that one can evaluate each time-dependent term in the Hamiltonian at specific times, including times at which the solution for the state vector is not yet available. This would seem to limit the split-step method strictly to modelling *linear* dynamics, that is, terms in the Hamiltonian must not depend explicitly on the state vector they are operating on. Whilst nature might fundamentally be described by linear dynamics, once approximations of various kinds are made in order to make problems tractable, it's common to end up with a nonlinear pseudopotential or nonlinear effective Hamiltonian. Note that non-*Hermitian*, pseudo-Hamiltonians—leading to non-unitary evolution—are fine. The split-step method has made no assumptions that rules them out, it only assumes the differential equation can be expressed in the form

$$\frac{\mathrm{d}}{\mathrm{d}t}\psi(t) = \sum_n H_n(t)\psi(t), \tag{3.62}$$

where $\{H_n\}$ are a set of linear operators, Hermitian or not.

Fortunately, one specific form of nonlinearity that cold atom physicists are particularly interested in—the nonlinear term in the Gross–Pitaevskii equation—can be incorporated without much difficulty. As mentioned, the problem with nonlinearity is that all but the first-order split step methods require you to evaluate terms in the Hamiltonian at some future time at which the state vector is not yet known, that is the algorithm contains steps akin to $\psi(t_{n+1}) = U(t_{n+1}, t_n; \psi(t_{n+1}))\psi(t_n)$ for some nonlinear unitary matrix $U(t_{n+1}, t_n; \psi(t_{n+1}))$—$U$ cannot be explicitly constructed because it both requires and is required by $\psi(t_{n+1})$.

For the Gross–Pitaevskii effective Hamiltonian, the second-order split-step method (from which the fourth order method is constructed) for a single step might be naïvely written

$$\psi(t_{n+1}) = \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}K\frac{\Delta t}{2}}\mathrm{e}^{-\frac{\mathrm{i}}{\hbar}(g\rho(t_{n+1})+V(t_{n+1}))\frac{\Delta t}{2}}\mathrm{e}^{-\frac{\mathrm{i}}{\hbar}(g\rho(t_n)+V(t_n))\frac{\Delta t}{2}}\mathrm{e}^{-\frac{\mathrm{i}}{\hbar}K\frac{\Delta t}{2}}\psi(t_n), \tag{3.63}$$

where $\psi(t)$ is the state vector[13] represented in a discrete position basis, g is the nonlinear interaction constant, $V(t)$ and $\rho(t) = \psi(t)\psi^\dagger(t)$ are diagonal matrices for the external potential and the density matrix for $\psi(t)$ in the same position basis, and K is a discrete approximation to the kinetic energy operator in the position basis.

[13] Usually when modelling the GPE the single-particle state vector is normalised to the number of particles, rather than unity, and so strictly speaking it cannot be called a state vector, though it otherwise can be treated as one in most respects.

As written, this can't be evaluated because $\psi(t_{n+1})$—required to evaluate $\rho(t_{n+1})$—is not yet known. The order we choose to exponentiate the terms in our Hamiltonian is arbitrary, however (so long as we alternately reverse that order each half-step as required by the second order split-step method), and so swapping the order gives

$$\psi(t_{n+1}) = \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}(g\rho(t_{n+1})+V(t_{n+1}))\frac{\Delta t}{2}}\mathrm{e}^{-\frac{\mathrm{i}}{\hbar}K\Delta t}\mathrm{e}^{-\frac{\mathrm{i}}{\hbar}(g\rho(t_n)+V(t_n))\frac{\Delta t}{2}}\psi(t_n), \tag{3.64}$$

which incidentally has the benefit that since K is (ordinarily) time independent, the two adjacent exponentials containing it can be combined into one. Now $\rho(t_{n+1})$ is contained within the leftmost exponential, and so it is the last operator to be applied in the timestep. Note that since $g\rho(t)$ is real and diagonal in the position basis (as is $V(t)$), this leftmost unitary merely changes the phase of the state vector at each point in space,

having no effect on its density. This means that $\rho(t)$ is, in fact, unaffected by this last unitary evolution operator. Hence, $\rho(t_{n+1})$ can be computed simply as the density matrix of the intermediate state vector that this unitary was to act on:

$$\rho(t_{n+1}) = \psi(t_{n+1})\psi^\dagger(t_{n+1}) = \tilde{\psi}\tilde{\psi}^\dagger, \tag{3.65}$$

where

$$\tilde{\psi} = e^{-\frac{i}{\hbar}K\Delta t} e^{-\frac{i}{\hbar}(g\rho(t_n)+V(t_n))\frac{\Delta t}{2}} \psi(t_n). \tag{3.66}$$

The inclusion of $V(t)$ with $g\rho(t)$ is optional, and if $V(t)$ was not real valued, would not be valid (since in that case the density *would* be affected by the evolution induced by $V(t)$). In such a case the Hamiltonian would have to be split into three terms with $V(t)$ and $g\rho(t)$ treated separately.

So now we can put some conditions on what types of nonlinear operators can be used within the second-order split step method. The first condition is that at most one nonlinear operator can be included, since it must be placed last in the sandwich of exponentials (otherwise its value at the end of the timestep cannot be inferred immediately prior to acting on the state vector). The second condition is that the nonlinear operator must be invariant with respect to the evolution that it itself induces in the state vector. Here we have an operator that depends only on the state vector's absolute value, but for which the corresponding unitary only evolves the state vector's phase. Another example might be an operator that depends only on the state vector's phase gradients, but evolves the state vector's absolute value, and so forth.

Although I find this argument compelling for second-order split-step, it's less obvious that it should hold for fourth-order split-step as well, which, even though it is based on multiple applications of second-order split-step, involves a substep that is backwards in time. "Evaluate the nonlinear operator based on the state vector at this specific time" becomes ambiguous when that moment in time is traversed in both directions by two different sub-steps. However, [68] have verified using computer symbolic algebra that indeed, up to even higher order split-step methods, putting the nonlinear density term last in the splitting and always evaluating it using the value of the intermediate state vector immediately prior results in the method having the same order accuracy in Δt as for linear operators only. Given this and my argument above, as well as the reasoning that the second-order split-step method has no way of 'knowing' whether it is acting backward in time or not when embedded in a higher-order split step method, I would expect the same to hold for all nonlinear operators meeting the above two conditions, though I haven't shown this explicitly.

3.3 For everything else, there's fourth-order Runge–Kutta

Fourth-order Runge–Kutta (RK4) is the enduring workhorse of numerical integration methods. Of the Runge–Kutta methods, it offers a good balance of accuracy and computational cost. When a problem does not have the properties that allow manifestly unitary or error-bounded methods to be used, or when enough computing power can be deployed so as to make these concerns irrelevant, and the programmer's time more important, fourth order Runge–Kutta is a good choice. It downsides are that it is not manifestly unitary, and its error is not bounded. Nonetheless for many problems this is not a concern in practice.

The advantages of fourth-order Runge–Kutta are compelling: it has global error that is fourth order in the integration timestep, involves four function evaluations per timestep, requires only linear arithmetic operations outside of the function evaluations,

and can be applied any problem that can be written in the form

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{x} = f(\boldsymbol{x}, t), \tag{3.67}$$

for some (possibly nonlinear) function f, where $\boldsymbol{x}$ is a vector of dynamical variables, often in our case the components of a state vector, or for classical dynamics the positions and velocities of an ensemble of particles[14]. Note that this formulation allows for initial value problems with coupled ODEs, discretised PDEs, as well as second or higher order differential equations, since an equation of the form

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2}\boldsymbol{x} = f(\boldsymbol{x}, t) \tag{3.68}$$

can be rewritten

$$\frac{\mathrm{d}}{\mathrm{d}t}(\boldsymbol{x}, \dot{\boldsymbol{x}}) = (\dot{\boldsymbol{x}}, f(\boldsymbol{x}, \dot{\boldsymbol{x}}, t)), \tag{3.69}$$

treating the time derivative of each element of $\boldsymbol{x}$ as simply another coupled dynamical variable.

Not unrelated to its ease of implementation, the method itself can be stated concisely. Propagation of the dynamical variables for one timestep from time t to time $t + \Delta t$ is computed as follows:

$$\begin{aligned}
\boldsymbol{k}_1 &= f(\boldsymbol{x}(t), t), \\
\boldsymbol{k}_2 &= f(\boldsymbol{x}(t) + \tfrac{1}{2}\boldsymbol{k}_1\Delta t, t + \tfrac{1}{2}\Delta t), \\
\boldsymbol{k}_3 &= f(\boldsymbol{x}(t) + \tfrac{1}{2}\boldsymbol{k}_2\Delta t, t + \tfrac{1}{2}\Delta t), \\
\boldsymbol{k}_4 &= f(\boldsymbol{x}(t) + \boldsymbol{k}_3\Delta t, t + \Delta t), \\
\boldsymbol{x}(t + \Delta t) &= \boldsymbol{x}(t) + \tfrac{\Delta t}{6}(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4).
\end{aligned} \tag{3.70}$$

Each step is self-contained, that is, the algorithm does not contain any state from previous steps. This is appealing as it means that f can change discontinuously between timesteps without giving rise to Runge's phenomenon [70] in the approximate solutions $\boldsymbol{x}(t)$, as can be the case with multi-step methods which do retain some dependency on previous steps. $\boldsymbol{x}(t)$ can also be modified discontinuously between steps without causing problems, as is required by Monte Carlo wavefunction or quantum jump methods [71, 72], or even in the imaginary time evolution method (see Section 3.5.1) which may require normalisation of the state vector in between steps. Fourth order Runge–Kutta is therefore quite compatible with stochastic processes and many other models which may not be described by a differential equation alone.

One downside is that the timestep used must be much smaller than the timescale on which $\boldsymbol{x}$ changes—even if $\boldsymbol{x}$'s time variation is highly regular. If $\boldsymbol{x}$'s time variation is dominated by the simple accumulation of complex phase at some angular frequency—as is often the case in quantum mechanics—the timesteps used for RK4 must be small enough to resolve these circles about the complex plane. This is in contrast to the exponentiation methods, for which an energy offset (and hence overall change in the angular frequency at which the state vector's elements accumulate phase) is largely irrelevant. Energy offsets or use of an interaction picture can mitigate this problem but requires some foresight, and may not be possible if the required energy offsets change in time or are not known analytically in advance. The method I develop in Section 3.6 is a partial remedy for this specific problem, which in my opinion is the biggest weakness of fourth order Runge–Kutta as applied to quantum state evolution, when compared to the unitary methods.

[14] For classical particle dynamics, often lower order symplectic methods such as the leapfrog method [69] can be preferable, but nonetheless it is hard to overstate the usefulness of a general purpose algorithm such as RK4 that can be deployed as a first attempt, or as a plan B once the assumptions required by another algorithm are violated

3.3.1 Complexity and parallelisability for the Schrödinger equation

Excluding the evaluation of f, RK4 requires a number of arithmetic operations proportional to the number of elements in $\boldsymbol{x}$, and so contributes time-complexity $\mathcal{O}(n)$ to the overall calculation, where n is the number of elements in $\boldsymbol{x}$. Since f itself usually doesn't run in linear time, the computational time complexity of RK4 is usually therefore that of evaluating f. Its parallelisability also comes down to that of f, since equations (3.70) above treat each element of $\boldsymbol{x}$ completely independently, with any couplings computed within f.

For the Schrödinger equation in a concrete basis, f is

$$f(\boldsymbol{\psi}, t) = -\frac{\mathrm{i}}{\hbar} H(t)\boldsymbol{\psi}, \tag{3.71}$$

where $\boldsymbol{\psi}$ is the state vector in the given basis and $H(t)$ is the matrix representation of the Hamiltonian in that basis[15]. Fourth order Runge–Kutta is therefore as computationally expensive and parallelisable as computing the matrix-vector product $H(t)\boldsymbol{\psi}$. In the worst case this multiplication is $\mathcal{O}(n^2)$ in the size of the Hilbert space and barely parallelisable at all, if H is dense. But in the common case (as mentioned in Section 3.2.4), of H being a sum of terms which act on different subspaces, i.e.

$$H(t) = \sum_{i=1}^{N} \mathbb{I}_{n_1} \otimes \cdots \otimes \mathbb{I}_{n_{i-1}} \otimes H_i(t) \otimes \mathbb{I}_{n_{i+1}} \otimes \cdots \otimes \mathbb{I}_{n_N} \tag{3.72}$$

where n_i is the dimensionality of the subspace acted on by the i^{th} term and $\mathbb{I}_m$ is the $m \times m$ identity matrix, then things are much better. If each term is dense, then the overall cost of evaluating the product $H(t)\boldsymbol{\psi}$ is $\mathcal{O}(n \sum_i n_i)$, where $n = \prod_i n_i$ is the dimensionality of the total Hilbert space, which can be considerably less than the $\mathcal{O}(n^2)$ of evaluating the single matrix-vector product for the total Hamiltonian. And if each term is a banded matrix with bandwidth b_i, then the cost of applying a single term in the Hamiltonian becomes $\mathcal{O}(nb_i)$ instead of $\mathcal{O}(nn_i)$, and so the cost of evaluating $H(t)\boldsymbol{\psi}$ becomes $\mathcal{O}(n \sum_i b_i)$. This is identical to the earlier result for the split-operator method once the trick of splitting up banded matrices into block-diagonal matrices was applied (section 3.2.4), but with much less work (in the sense of programmer effort rather than computational complexity). Representing $H(t)$ as a sum of terms over different subspaces is no extra work—this is the form we are likely to be writing Hamiltonians in already, and constructing the total $H(t)$ is often not required if one desires only to propagate a state vector in time.[16] No, we are already applying these operators to different subspaces of the Hilbert space and then summing the results without thinking twice about it, and so the above analysis mostly serves as a reminder that what we are already doing most of the time is in fact very efficient.

Parallelisation, provided one or more of the matrices are banded, is also straightforward, since if A is banded with bandwidth b, then

$$(A\boldsymbol{\psi})_i = \sum_{j=-b}^{b} A_{i,i+j}\psi_j, \tag{3.73}$$

that is, calculation of an element of $A\boldsymbol{\psi}$ requires only the corresponding element of $\boldsymbol{\psi}$ and its nearest b neighbours on each side. One can therefore divide up the state vector into contiguous regions (in the subspace in which A acts), and compute different elements of the product on different compute threads, requiring an exchange of only b elements at each boundary[17] between neighbouring threads. An example of this is to split two dimensional space into a number of pieces in each dimension (resulting in a 2D grid) so as to compute the application of (finite difference approximations to) the x and y second derivative operators on a state vector in parallel.

[15] For the Gross–Pitaevskii equation this would be a nonlinear $H(\boldsymbol{\psi}, t)$, and everything else in this section still applies.

[16] Though constructing the matrix form of say, $\frac{\hat{p}_x^2}{2m} + \frac{\hat{p}_y^2}{2m}$ for some finite difference or pseudospectral representations of $\hat{p}_x$ and $\hat{p}_y$ can be useful if one wants to say, compute the dispersion relation of a particular system by diagonalising it.

[17] A note about minimising the effect of latency: at the start of an RK4 substep, have each thread send the required elements at the edges of its region in each subspace to its neighbouring threads *first*. *Then* compute $H(t)\boldsymbol{\psi}$ on the interior elements. If each thread has a sufficiently large workload, then by the time this is complete, the data from neighbouring regions will have arrived and threads will not have spent any time waiting for each other.

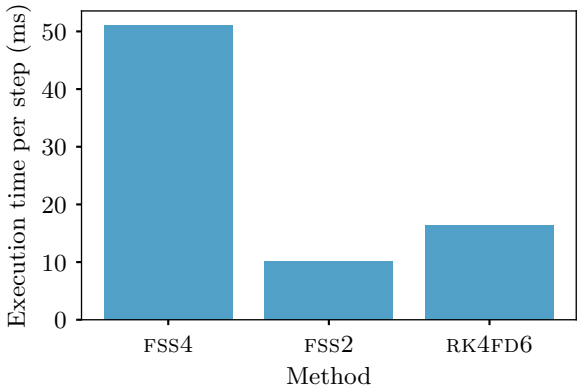

Figure 3.3: The average computation time per step of second and Fourth order Fourier split-step (see Section 3.4.1) and fourth-order Runge–Kutta using sixth order finite differences (see Section 3.4.2) for the simulations detailed in Figure 3.4. It should be noted that the use of finite differences is not significant—as mentioned in Section 3.4.2, it is difficult to observe any difference between the speed of finite differences and Fourier derivatives in practice on present computers. The difference in speed is almost entirely due to the cost of computing exponentials, as can be seen by comparing FSS2 and FSS4, which differ by a factor in five both in speed and in the number of exponentials in their implementations. For fourth order accuracy, RK4 is about three times faster than Fourier split-step.

An additional benefit of parallelised RK4 when compared to split-operator is that the same amount of data needs to be sent between threads regardless of the number of terms in the Hamiltonian. In split-step methods, each term in the Hamiltonian is exponentiated separately (multiple times for the higher order schemes), requiring an exchange of data each time. The amount of data needing to be exchanged per step therefore scales with the number of terms in the Hamiltonian being parallelised, whereas for RK4 it is constant.

The constant factors that big-O notation disregards also favour RK4 when compared to split-operator methods. For one, addition and multiplication are much cheaper than exponentiation, meaning that the exponentials in split-operator methods may add considerable computational cost if the Hamiltonian itself is simple. Furthermore, parallelised or not, each term in fourth order split-operator adds 20 matrix-vector products in the space the term acts, whereas RK4 requires only one additional matrix-vector product per term. In practice due to these properties, RK4 runs considerably faster (see Figure 3.3) than split-operator methods, even for simple systems, and the gap widens as complexity increases. This combined with the relatively minor improvement in accuracy or stability (see Figure 3.4) between the methods make RK4 an enduring choice for cold atom problems.

3.4 Continuous degrees of freedom

The single-particle, non-relativistic, scalar Schrödinger wave equation, as distinct from the general Schrödinger equation (3.1), is:

$$i\hbar\frac{\partial}{\partial t}\psi(\boldsymbol{r},t) = \left[-\frac{\hbar^2}{2m}\nabla^2 + V(\boldsymbol{r})\right]\psi(\boldsymbol{r},t). \tag{3.74}$$

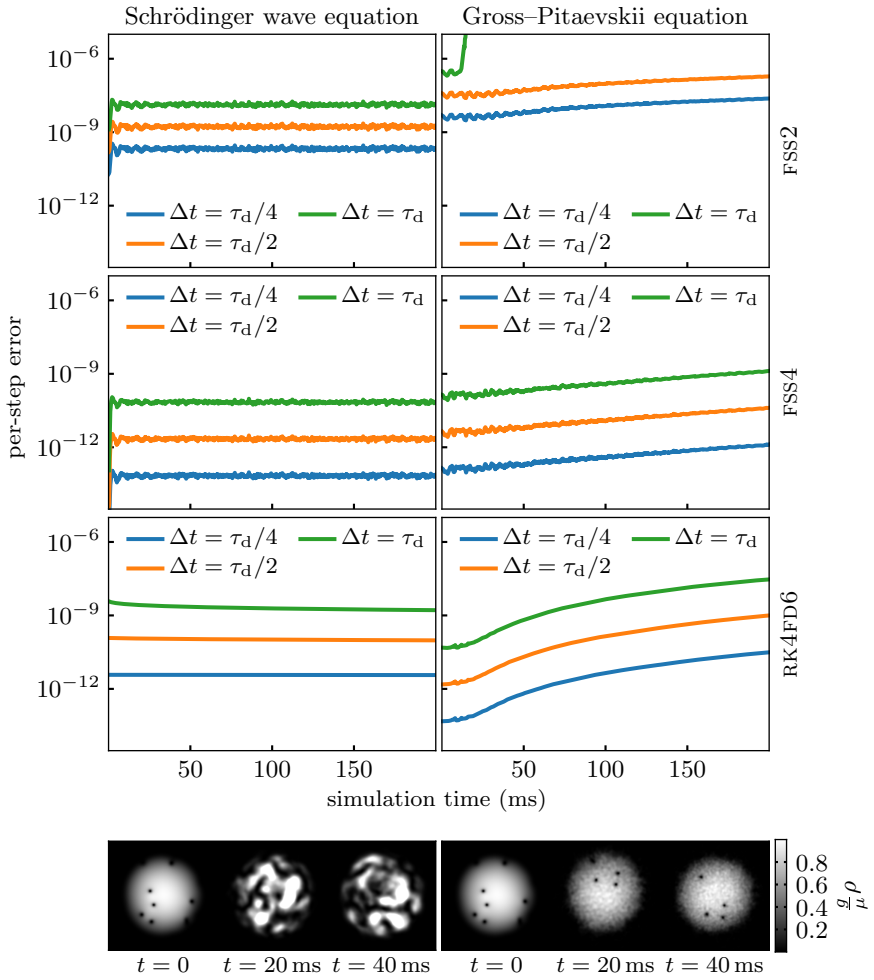

Figure 3.4: A comparison of the RMS per-step accuracy of second and Fourth order Fourier split-step (see Section 3.4.1) and fourth-order Runge–Kutta using sixth order finite differences (see Section 3.4.2). In the left column, the linear Schrödinger equation is solved for a 2D wavefunction and on the right the Gross–Pitaevskii equation for a 2D condensate wavefunction, both for the case of ^{87}Rb. The spatial grid is 256×256 and represents a spatial region of 10 µm. The same initial conditions are used for each, produced in the following way: first a Gross–Pitaevskii ground state (for the sake of comparison, even though this is not a ground state of the Schrödinger equation) in a harmonic trap chosen to give a Thomas–Fermi radius of 7.5 µm is produced using successive over-relaxation (Section 3.5.2). Vortices are created by multiplying by a spatially varying complex phase factor to produce the appropriate phase winding for a number of randomly chosen vortices, as described in the example in Section 3.6. This is then relaxed using imaginary time evolution (Section 3.5.1) of the Gross–Pitaevskii equation for a duration equal the chemical potential timescale $\tau_\mu = \frac{2\pi\hbar}{\mu}$ to produce a physically realistic initial condition. This is then evolved in time using the three methods, with three different timesteps, expressed as multiples of the dispersion timescale (described in Section 3.4.4) $\tau_d = \frac{m\Delta x^2}{\pi\hbar}$ where Δx is the grid spacing. Second order Fourier split-step, although unitary and thus having bounded error, reaches that bound for the smallest timestep. The two fourth order methods have comparable accuracy, with Fourier split step being slightly more accurate and the error in RK4 tending to increase faster with time (for the case of the Gross–Pitaevskii equation). Comparing with Figure 3.3, the difference is even more marginal if we compare by equal computational cost per unit simulation time—which allows RK4 to take timesteps about three times smaller before it consumes the same computational resources as Fourier split step.

Similarly, as mentioned in Section 2.2, the equation for the single-particle wavefunction of an atom in a single-component Bose–Einstein condensate is the Gross–Pitaevskii equation

$$i\hbar\frac{\partial}{\partial t}\psi(\boldsymbol{r},t)=\left[-\frac{\hbar^2}{2m}\nabla^2+V(\boldsymbol{r})+g|\psi(\boldsymbol{r},t)|^2\right]\psi(\boldsymbol{r},t), \tag{3.75}$$

where $\psi(\boldsymbol{r},t)=\sqrt{N}\langle\boldsymbol{r}|\psi(t)\rangle$ is the single-particle wavefunction scaled by the square root number of atoms N.

Both these equations are partial differential equations involving both spatial and temporal derivatives. But in numerical quantum mechanics all state vectors are mapped to column vectors and all operators to matrices. Spatial wavefunctions are no exception to the former and differential operators such as ∇^2 are no exception to the latter–these objects can be thought of as infinite-dimensional vectors and matrices. So the above two equations are specific instances of the general Schrödinger equation (3.1), given specific Hamiltonians, and represented in a concrete—albeit infinite-dimensional—basis. But we can only perform a finite number of computations, so what do these vectors and operators look like once we reduce them to something finite? That depends on whether we choose to discretise space on a grid, or use a functional basis (and on which functional basis we choose). As we'll see, however, spatial discretisation can be a special case of a functional basis, namely the Fourier basis, plus an additional approximation or two. The resulting matrices are *banded*, justifying the previous two sections' attention to dealing with banded matrices efficiently.

3.4.1 Spatial discretisation on a uniform grid: the Fourier basis

Imagine a two dimensional spatial region within which we are solving the single-component Gross–Pitaevskii equation, evolving an initial condensate wavefunction in time. Having specified which degrees of freedom we want to simulate (two continuous degrees of freedom, one for each spatial dimension), the next step according to the method outlined in Section 3.1 is to choose a basis in which to represent this state vector.

Let's say we discretise space in an equally-spaced $n_x\times n_y=7\times 7$ rectangular grid,[18] with spacings Δx and Δy, and only represent the wavefunction at those 49 points in space. The state vector can then be represented by a list of 49 complex numbers, each taken to be the wavefunction's value at the spatial position corresponding to one gridpoint. This 49-vector is now a concrete representation of our state vector (Figure 3.5).

[18] For the sake of example—256 × 256 is a more realistic minimum.

We can also evaluate the potential (and nonlinear term in the case of the Gross–Pitaevskii equation) at each gridpoint and declare this a diagonal operator (Figure 3.6). Finally, we could use finite differences to compute the Laplacian—equivalent to replacing the Laplacian with a matrix

$$L=L_x\otimes\mathbb{I}_{n_y}+\mathbb{I}_{n_x}\otimes L_y \tag{3.76}$$

where L_x and L_y are (banded) matrices for finite-difference approximations to second derivatives in each direction. We might also use discrete Fourier transforms to evaluate the Laplacian, since

$$\nabla^2\psi(\boldsymbol{r})=\mathcal{F}^{-1}\left\{-k^2\mathcal{F}\{\psi(\boldsymbol{r})\}(\boldsymbol{k})\right\}, \tag{3.77}$$

where $k=|\boldsymbol{k}|$.

This is all well and good, and it works. But at what point did we choose a basis just now—what are the basis vectors? This just looks like discretising space at a certain resolution, rather than the formal process of choosing a basis and projecting the state vector

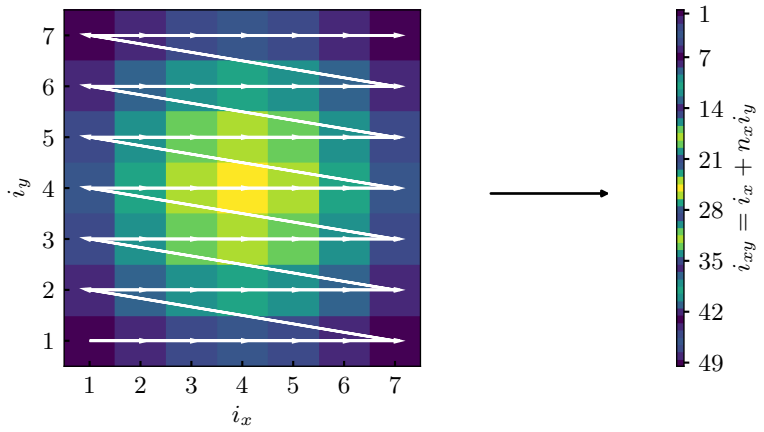

Figure 3.5: Discretising a function over two dimensional space on a grid yields a list of coefficients, one for each gridpoint. These can be arranged as a column vector, and in this way a two dimensional wavefunction approximated by a finite-dimensional state vector. Computationally we don't normally treat this state vector as a column vector—it is more convenient to leave it as a two-dimensional array. But conceptually it is a single vector living in the product space of the discretised x and y spaces.

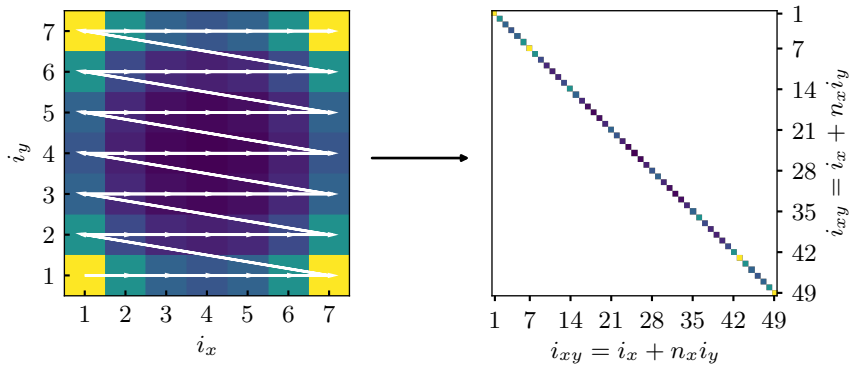

Figure 3.6: A discretised potential energy operator can be formed by evaluating the function for the potential at a set of gridpoints. The resulting matrix is diagonal and square with $n_x \times n_y$ rows and columns. Since multiplying this operator by a state vector entails multiplying each element of the state vector by one diagonal element of the potential operator, both the state vector and the diagonals of the potential operator can be stored in simulation code as 2D arrays and multiplied elementwise, hiding somewhat the fact that the operation is still a matrix-vector product. This way of discretising a potential operator is called the pseudospectral approximation, and is described later in this section

and operators onto each basis vector, as outlined in Section 3.1. Assuming what we've done is equivalent to choosing a basis, that basis has a finite number (49) of basis vectors, which means it cannot be complete, since state vectors we're approximately representing with it require an infinite number of complex numbers to be described exactly.[19] So what do the basis functions look like, and what state vectors have we implicitly excluded from simulation by choosing a basis that is incomplete?

[19] One for each position within the two dimensional space we're representing.

Prior to discretisation, the spatial wavefunction $\psi(\boldsymbol{r}, t) = \langle \boldsymbol{r} | \psi(t) \rangle$ was already the

representation of the abstract state vector $|\psi\rangle$ in the 'spatial basis'—a basis in which the basis vectors $\{|\boldsymbol{r}\rangle\}$ are Dirac deltas positioned at each point in space. The value of the wavefunction $\psi(\boldsymbol{r})$ for a specific $\boldsymbol{r}$ is then simply a coefficient saying how much of the basis vector $|\boldsymbol{r}\rangle$ to include in the overall state vector. What we have *not* done is chosen a subset of these Dirac delta basis functions as our basis. This would be very strange—our representation of the wavefunction would allow it to be non-zero at the gridpoints, but not in between, like a comb. Spatially separated Dirac deltas do not spatially overlap at all; the matrix elements of the kinetic energy operator:

$$\langle \boldsymbol{r}_i|\hat{K}|\boldsymbol{r}_j\rangle = \int \delta(\boldsymbol{r}-\boldsymbol{r}_i)\left(-\frac{\hbar^2}{2m}\nabla^2\right)\delta(\boldsymbol{r}-\boldsymbol{r}_j)\,\mathrm{d}\boldsymbol{r} \tag{3.78}$$

would all be zero for $i \neq j$, disallowing any flow of amplitude from one point in space to another by virtue of it not being able to pass through the intervening points.

Neither have we implicitly chosen a set of two-dimensional boxcar functions centred on each gridpoint with width Δx and Δy in each direction respectively. These cover all space in between gridpoints, but are not twice-differentiable everywhere, and hence the kinetic energy operator's matrix elements cannot be evaluated[20]. No, neither of these bases makes sense. To interpret our spatial grid as a basis, we need a set of functions $\phi_{ij}(\boldsymbol{r})$ (where i and j are the indices of the gridpoints in the x and y directions respectively) that are orthonormal, are non-zero only at one gridpoint and are zero at all others, and are twice differentiable everywhere in our spatial region. Infinite choices are available, differing in which subspace of the original Hilbert space they cover. A sensible heuristic for choosing one is that we want to be able to represent the state vectors whose wavefunctions do not change much between adjacent gridpoints, and we are happy for the necessary incompleteness of our basis to exclude wavefunctions with any sort of structure in between gridpoints.

[20] Delta functions aren't twice differentiable either, so this itself isn't a fatal flaw–but even if one defines the boxcar functions as the limit of twice differentiable functions becoming increasingly square with some parameter, the limit for the kinetic energy matrix elements between adjacent boxcars goes to infinity.

The discrete Fourier transform to the rescue

It turns out that discretising space in this way can indeed be equivalent to choosing a sensible basis. This is made clearer by first discretising in Fourier space instead, and seeing how this can imply a discretisation in real space.

One possible basis for representing all possible state vectors is the Fourier basis $\{|\boldsymbol{k}_{ij}\rangle\}$. With it, any state vector (whose wavefunction is non-zero only within the 2D region) can be represented as the sum of basis vectors whose wavefunctions are 2D plane waves, also localised to the 2D region:

$$\langle \boldsymbol{r}|\boldsymbol{k}_{ij}\rangle = \begin{cases} \frac{1}{\sqrt{A}}\mathrm{e}^{i\boldsymbol{k}_{ij}\cdot\boldsymbol{r}} & (\boldsymbol{r} \text{ within 2D region}) \\ 0 & (\boldsymbol{r} \text{ not within 2D region}), \end{cases} \tag{3.79}$$

where A is the area of the 2D region and the wavevector of each plane wave is

$$\boldsymbol{k}_{ij} = \left[\frac{2\pi i}{L_x}, \frac{2\pi j}{L_y}\right]^{\mathrm{T}}, \tag{3.80}$$

where i and j are (possibly negative) integers. Any state vector whose wavefunction is localised to the 2D region can then be written as the infinite sum:

$$|\psi\rangle = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty}\langle \boldsymbol{k}_{i,j}|\psi\rangle\,|\boldsymbol{k}_{i,j}\rangle \tag{3.81}$$

$$\Rightarrow \psi(\boldsymbol{r}) = \langle \boldsymbol{r}|\psi\rangle = \begin{cases} \sum_{i=0}^{\infty}\sum_{j=0}^{\infty}\langle \boldsymbol{k}_{ij}|\psi\rangle\,\frac{1}{\sqrt{A}}\mathrm{e}^{i\boldsymbol{k}_{ij}\cdot\boldsymbol{r}} & (\boldsymbol{r} \text{ within 2D region}) \\ 0 & (\boldsymbol{r} \text{ not within 2D region}). \end{cases} \tag{3.82}$$

So $\{\langle \boldsymbol{k}_{ij}|\psi\rangle\}$ are simply the coefficients of the 2D Fourier series of $\psi(\boldsymbol{r})$.

What does this have to do with our discretised space? These basis functions $\{\langle \boldsymbol{r}|\boldsymbol{k}_{ij}\rangle\}$ don't have the required properties for a spatial discrete basis. For one, there are an infinite number of them, and we require 49 for our 7×7 example. Secondly, all of them are non-zero everywhere within the 2D region, whereas we require each basis function to be non-zero at exactly one of our 49 gridpoints.

We can solve the first problem by truncating the Fourier series. By only including basis vectors $|\boldsymbol{k}_{ij}\rangle$ for which:

$$\begin{cases} i \in [-\frac{n_x}{2}, \frac{n_x}{2} - 1] & (n_x \text{ even}) \\ i \in [-\frac{n_x-1}{2}, \frac{n_x-1}{2}] & (n_x \text{ odd}) \end{cases} \tag{3.83}$$

and

$$\begin{cases} j \in [-\frac{n_y}{2}, \frac{n_y}{2} - 1] & (n_y \text{ even}) \\ j \in [-\frac{n_y-1}{2}, \frac{n_y-1}{2}] & (n_y \text{ odd}) \end{cases} \tag{3.84}$$

we include only the n_x and n_y longest wavelengths in each respective spatial dimension. This is a sensible truncation with a physically meaningful interpretation. By making it, we are no longer able to represent state vectors with short wavelength components. Because the kinetic energy operator, when represented in the Fourier basis, is:

$$\langle \boldsymbol{k}_{ij}|\hat{K}|\boldsymbol{k}_{i'j'}\rangle = \frac{\hbar^2 k^2}{2m}\delta_{ii'}\delta_{jj'}, \tag{3.85}$$

where $k = |\boldsymbol{k}_{ij}|$, by excluding basis vectors with larger wavevectors, we are excluding state vectors with large kinetic energy. Thus the truncation is a kinetic energy cutoff, and is an accurate approximation whenever a simulation is such that the system is unlikely to obtain kinetic energies above the cutoff.[21] It also matches our earlier intuition that our basis should represent wavefunctions that don't vary much between gridpoints—here we are discarding short wavelengths and hence limiting wavefunctions we can represent to ones that vary slowly in space compared to the cutoff wavelength.

[21] Because a *square* region in Fourier space is being carved out, by limiting each of k_x and k_y to finite ranges rather than the total wavenumber $k = \sqrt{k_x^2 + k_y^2}$, there is no single kinetic energy cutoff so to speak. Nonetheless there is a maximum wavenumber $k_{\text{max}} = \min(\{|k_x|\} \cup \{|k_y|\})$ defining a kinetic energy cutoff $K_{\text{max}} = \hbar^2 k_{\text{max}}^2/(2m)$ below which kinetic energies definitely are representable and above which they may not be.

Now we have a set of basis vectors—a discrete Fourier basis—but their spatial wavefunctions still don't have the property of being non-zero only at a single gridpoint each. On the contrary, each plane wave has a constant amplitude everywhere in space. But consider the following superposition of Fourier basis vectors:

$$|\boldsymbol{r}_{ij}\rangle = \sum_{i'}\sum_{j'} \mathrm{e}^{-\mathrm{i}\boldsymbol{k}_{i'j'}\cdot\boldsymbol{r}_{ij}}\,|\boldsymbol{k}_{i'j'}\rangle \tag{3.86}$$

with $\boldsymbol{r}_{ij} = (i\Delta x, j\Delta y)^{\mathrm{T}}$. The set of vectors $\{|\boldsymbol{r}_{i,j}\rangle\}$ are also an orthonormal basis, related to the discrete Fourier basis by a unitary transformation with matrix elements:

$$U_{\text{DFT2},i'j'ij} = \langle \boldsymbol{k}_{i'j'}|\boldsymbol{r}_{ij}\rangle = \mathrm{e}^{-\mathrm{i}\boldsymbol{k}_{i'j'}\cdot\boldsymbol{r}_{ij}}. \tag{3.87}$$

This unitary transformation is in fact a two-dimensional discrete Fourier transform (hence the subscript), and the basis vectors $\{|\boldsymbol{r}_{ij}\rangle\}$ have spatially localised wavefunctions that are non-zero only at one of the spatial gridpoints. Vectors and matrices can be transformed from their discrete Fourier space representation to their discrete real-space representation and back using the unitary U_{DFT2}:

$$\boldsymbol{\psi}_{\text{real}} = U_{\text{DFT2}}^{\dagger}\boldsymbol{\psi}_{\text{Fourier}} \tag{3.88}$$

$$\boldsymbol{\psi}_{\text{Fourier}} = U_{\text{DFT2}}\boldsymbol{\psi}_{\text{real}} \tag{3.89}$$

$$A_{\text{real}} = U_{\text{DFT2}}^{\dagger}A_{\text{Fourier}}U_{\text{DFT2}} \tag{3.90}$$

$$A_{\text{Fourier}} = U_{\text{DFT2}}A_{\text{Real}}U_{\text{DFT2}}^{\dagger}. \tag{3.91}$$

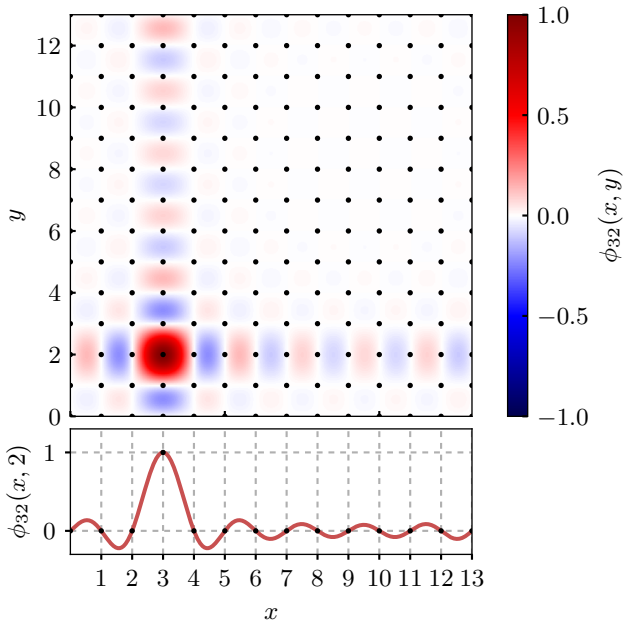

Figure 3.7: An example of the spatial representation of one of the basis vectors obtained by transforming the discrete Fourier basis using the discrete Fourier transform. Since the discrete Fourier transform is a unitary transformation, the set of these basis functions forms an equivalent orthonormal basis for representing state vectors and operators.

where $\boldsymbol{\psi}_{\text{real}}$ is the vector of coefficients $\psi_{\text{real},ij} = \langle \boldsymbol{r}_{ij} | \psi \rangle$ for representing a state vector in the discrete real space basis, $\boldsymbol{\psi}_{\text{Fourier}}$ is the vector of coefficients $\psi_{\text{Fourier},ij} = \langle \boldsymbol{k}_{ij} | \psi \rangle$ for the state vector in the discrete Fourier basis, and A_{real} and A_{Fourier} are the representations of some operator $\hat{A}$ in the discrete real and Fourier bases respectively, with matrix elements $A_{\text{real},iji'j'} = \langle \boldsymbol{r}_{ij} | \hat{A} | \boldsymbol{r}_{i'j'} \rangle$ and $A_{\text{Fourier},iji'j'} = \langle \boldsymbol{k}_{ij} | \hat{A} | \boldsymbol{k}_{i'j'} \rangle$.

The spatial representation of the basis vectors $\{ |\boldsymbol{r}_{ij}\rangle \}$ can be computed using (3.79) as:

$$\phi_{ij}(\boldsymbol{r}) = \langle \boldsymbol{r} | \boldsymbol{r}_{ij} \rangle = \sum_{i'j'} \mathrm{e}^{-\mathrm{i}\boldsymbol{k}_{i'j'} \cdot \boldsymbol{r}_{ij}} \langle \boldsymbol{r} | \boldsymbol{k}_{i'j'} \rangle \tag{3.92}$$

$$\Rightarrow \phi_{ij}(\boldsymbol{r}) = \begin{cases} \sum_{i'j'} \frac{1}{\sqrt{L_x L_y}} \mathrm{e}^{\mathrm{i}\boldsymbol{k}_{i'j'} \cdot (\boldsymbol{r} - \boldsymbol{r}_{ij})} & (\boldsymbol{r} \text{ within 2D region}) \\ 0 & (\boldsymbol{r} \text{ not within 2D region}), \end{cases} \tag{3.93}$$

where L_x and L_y are the spatial extents of the x and y dimensions. An example of one of these basis vectors in the spatial representation is plotted in Figure 3.7.

These functions are sometimes called *periodic sinc functions*, *band-limited delta functions*, or the *Dirichlet kernel* [73, p. 619; 74]. Each of them is zero at all of the gridpoints except one, and they form an orthonormal basis set. They satisfy all of our requirements to be a basis corresponding to our gridded discretisation of space.

One thing to note is that these functions are periodic. By using the Fourier basis in the way we have to restrict our basis to cover only a finite region of both Fourier space and real space, we have necessarily imposed periodicity on the problem. This periodicity shows itself when we compute matrix elements of operators in this basis. If we compute the kinetic energy operator's matrix elements for example, it will couple basis states across the boundary of the region, resulting in spatial periodicity—a wavepacket moving rightward through the right boundary will emerge moving rightward from the left boundary.

Less obviously, the basis is also periodic in Fourier space, and so a wavepacket moving out of the region of Fourier space simulated will also wrap around to the opposite side of Fourier space. In real space, this may appear as a wavepacket undergoing acceleration only to suddenly reverse its velocity as if reflected off a barrier. This effect is unphysical[22] and should be taken as a sign that the spatial grid is not fine enough for the dynamics being simulated.

[22] With the possible exception of the region of Fourier space being simulated corresponding to the first Brillouin zone of a lattice potential, in which case these velocity reversals would correspond to Bloch oscillations.

The discrete Fourier basis we've described is one example of a *spectral basis*, and a numerical method which represented the state vector solely in this basis would be called a *spectral method*. Other choices of basis functions, such as polynomials (see Section 3.4.3) or spherical harmonics lead to other spectral methods.[23] The discrete spatial basis discussed above, despite being related to the Fourier basis by a unitary transformation, is often called a *pseudospectral* basis, however, and a method representing the state vector in this basis a *pseudospectral method*. Although it is 'just another basis', the fact that the basis functions are zero at all spatial points bar one each leads to the possibility of a further approximation when representing some operators, which makes bases with this property especially useful. I describe this in the following section.

[23] The wavefunctions of eigenstates of the 3D harmonic oscillator are products of radial polynomials and spherical harmonics, the combination of which is a good spectral basis for many problems.

Vector and matrix elements in the Fourier and pseudospectral bases

We now have a finite basis $\{|\boldsymbol{r}_{ij}\rangle\}$ that matches our intuitions somewhat for representing a wavefunction at a set of gridpoints. A state vector can be approximated as a linear sum of these basis vectors, with the coefficient for each one being equal to the projection of state vector's wavefunction onto the basis vector's wavefunction:

$$|\psi\rangle \approx \sum_{ij} \psi_{ij} |\boldsymbol{r}_{ij}\rangle \,, \tag{3.94}$$

where

$$\psi_{ij} = \int \phi^*_{ij}(\boldsymbol{r}) \psi(\boldsymbol{r}) \, \mathrm{d}\boldsymbol{r} \,. \tag{3.95}$$

In practice however this integral is rarely done. Instead, ψ_{ij} is simply taken to be the value of the exact wavefunction $\psi(\boldsymbol{r}) = \langle \boldsymbol{r}|\psi\rangle$ at the point $\boldsymbol{r} = \boldsymbol{r}_{ij}$:

$$\psi_{ij} \approx \psi(\boldsymbol{r}_{ij}) \tag{3.96}$$

To see why this is a good approximation, imagine that the approximation (3.94) were exact, that is, $\psi(\boldsymbol{r})$ were exactly equal to a linear sum of the functions $\{\phi_{ij}(\boldsymbol{r})\}$. Since all the basis functions are zero at the point $\boldsymbol{r}_{ij}$ except for ϕ_{ij}, the value of $\psi(\boldsymbol{r}_{ij})$ must come solely from the $\psi(\boldsymbol{r})$'s projection onto the basis function $\phi_{ij}(r)$. Since approximating (3.94) underlies the results of any simulation that discretises a state vector in this way, treating it as exact for the initial projection onto the discrete basis is making an assumption no worse than that already being relied upon.

With a basis and initial discrete state vector in hand, we can now proceed to calculate matrix elements of the Hamiltonian, after which we can proceed to solve the differential equation (3.3) to determine how the coefficients $\{\psi_{ij}\}$ evolve in time.

The specific properties of our Fourier/pseudospectral basis make it quite useful for a range of common Hamiltonians. For example, let's take the single particle Schrödinger Hamiltonian:

$$\hat{H}_{\text{Schrö}} = \frac{\hbar^2 \hat{k}^2}{2m} + V(\hat{\boldsymbol{r}}), \tag{3.97}$$

where $\hat{k} = |\hat{\boldsymbol{k}}|$.

The two terms, kinetic and potential, are each diagonal in different bases. The kinetic term is diagonal in the Fourier basis:

$$\langle \boldsymbol{k}' | \frac{\hbar^2 \hat{k}^2}{2m} | \boldsymbol{k} \rangle = \frac{\hbar^2 k^2}{2m} \delta(\boldsymbol{k} - \boldsymbol{k}'), \tag{3.98}$$

where $k = |\boldsymbol{k}|$, and the potential term is diagonal in the spatial basis:

$$\langle \boldsymbol{r}' | V(\hat{\boldsymbol{r}}) | \boldsymbol{r} \rangle = V(\boldsymbol{r}) \delta(\boldsymbol{r} - \boldsymbol{r}'). \tag{3.99}$$

The kinetic term is also diagonal our discrete Fourier basis:

$$K_{\text{Fourier},i'j'ij} = \langle \boldsymbol{k}_{i'j'} | \frac{\hbar^2 \hat{k}^2}{2m} | \boldsymbol{k}_{ij} \rangle = \frac{\hbar^2 k_{ij}^2}{2m} \delta_{i'i} \delta_{j'j}, \tag{3.100}$$

however—perhaps surprisingly—the potential term is not diagonal in the pseudospectral basis, only approximately so:

$$V_{\text{real},i'j'ij} = \langle \boldsymbol{r}_{i'j'} | V(\hat{\boldsymbol{r}}) | \boldsymbol{r}_{ij} \rangle \approx V(\boldsymbol{r}_{ij}) \delta_{i'i} \delta_{j'j}. \tag{3.101}$$

Nonetheless, equation (3.101) is in practice treated as exact for the purpose of computing matrix elements in the discrete basis of operators that are diagonal in the full spatial basis. As above with projecting a state vector using simply the values of a wavefunction at the gridpoints (rather than doing integrals), treating this approximation as exact is equivalent to making the assumption that the potential $V(\boldsymbol{r})$ is already accurately representable in the discrete basis as a diagonal operator:

$$V(\boldsymbol{r}) \approx \sum_{ij} V(\boldsymbol{r}_{ij}) \phi_{ij}^*(\boldsymbol{r}) \phi_{ij}(\boldsymbol{r}), \tag{3.102}$$

and so similarly is going to be a good approximation whenever the discrete basis is well able to represent the potential. So long as your discrete basis can accurately represent the state vectors and spatially-diagonal operators you will be using, it makes little difference whether you project those vectors and operators onto the basis using integrals or using simply their values at the gridpoints.

This alternate method of projection has a name, it is called *collocation* [60, p. 227]. Using collocation instead of vector projection amounts to treating our basis vectors as a scheme for interpolating state vectors and operators in between gridpoints, given their values at the points, rather than as basis vectors to project upon. Another way to interpret collocation is to say that we are evaluating the vector projections using integrals after all, however, we're numerically computing those integrals using a quadrature scheme, only evaluating the integrand at the gridpoints and performing a discrete sum [60, p. 283]. Collocation is what puts the *pseudo* in pseudospectral—if we evaluated all these operators using integrals instead, we would be treating the discrete spatial basis exactly the same as any spectral basis.

Compared to a purely spectral method, pseudospectral methods are of comparable accuracy [75]. This makes intuitive sense—discrete sums instead of integrals is how we are going to do all inner products once we are in the discrete basis—so it can't be much worse to use the same method to approximate operators and initial state vectors. The sole downside of pseudospectral methods, according to [75], is that the error can lead to instability in the presence of certain nonlinearities. Specifically, if long wavelength waves interact in a way that would produce wavelengths shorter than two grid spacings (the Nyquist wavelength), pseudospectral methods will produce longer wavelength waves instead, whereas in a purely spectral method the interaction would not occur, being due

to couplings to a Fourier mode outside the discrete Fourier basis. This *aliasing* can cause instability, but can be circumvented with smoothing techniques [76]. This is no great downside: if you want to simulate short length scales, you need to choose a grid spacing small enough to represent them, whereas if short wavelengths are produced despite your willingness to ignore them, you must smooth them away before they are aliased into long wavelengths that you do care about.

Finally, armed with a kinetic energy operator in Fourier space and a pseudospectral approximation to the potential operator in real space, we can write all the matrix elements of $\hat{H}_{\text{Schrö}}$ in a single basis, and thus our discretised, pseudospectral two-dimensional Schrödinger wave equation:

$$i\hbar \frac{\mathrm{d}}{\mathrm{d}t}\psi_{i'j'}(t) = \sum_{ij} H_{i'j',ij}(t)\psi_{ij}(t) \tag{3.103}$$

$$\Rightarrow i\hbar \frac{\mathrm{d}}{\mathrm{d}t}\psi_{i'j'}(t) = \sum_{ij} \left[U^{\dagger}_{\text{DFT2}} K_{\text{Fourier}} U_{\text{DFT2}} + V_{\text{real}}(t)\right]_{i'j',ij} \psi_{ij}(t), \tag{3.104}$$

where we have used the discrete Fourier transform to transform the kinetic energy operator into the discrete real space basis, and allowed the potential operator to be possibly time dependent.

The right hand side of this expression can now simply be evaluated, yielding the time derivative of each component $\psi_{ij}(t)$ of the discrete state vector $\boldsymbol{\psi}(t)$:

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\psi}(t) = -\frac{\mathrm{i}}{\hbar}\left[U^{\dagger}_{\text{DFT2}} K_{\text{Fourier}} U_{\text{DFT2}} \boldsymbol{\psi}(t) + V_{\text{real}}(t)\boldsymbol{\psi}(t)\right] \tag{3.105}$$

$$\Rightarrow \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\psi}(t) = -\frac{\mathrm{i}}{\hbar}\left[\text{FFT}_2^{-1}\left\{\frac{\hbar^2 \tilde{\boldsymbol{k}}^{\odot 2}}{2m} \odot \text{FFT}_2\left\{\boldsymbol{\psi}(t)\right\}\right\} + V(\tilde{\boldsymbol{r}}, t) \odot \boldsymbol{\psi}(t)\right], \tag{3.106}$$

where FFT_2 is the two dimensional fast Fourier transform, an efficient implementation of the discrete Fourier transform (taking time $\mathcal{O}(n \log n)$ in the size of each dimension), $\tilde{\boldsymbol{r}}$ is a vector (of vectors) containing each discrete position vector (such that $V(\tilde{\boldsymbol{r}}, t)$ is a vector (of scalars) containing the potential evaluated at each discrete position), $\tilde{\boldsymbol{k}}$ is a vector (of vectors) containing each discrete k-vector, such that $\tilde{\boldsymbol{k}}^{\odot 2}$ is a vector (of scalars) containing the squared magnitude of each discrete k-vector, and $\odot$ represents elementwise multiplication (or exponentiation) of vectors. Other than $\boldsymbol{\psi}$, these vectors are more akin to arrays used in programming languages than to members of a vector space, hence the somewhat clunky notation in (3.106). Comparison with the continuous version of equation (3.106) (i.e. the Schrödinger wave equation (3.74)), since elementwise multiplication of functions is a more common operation in mathematics, might be clarifying:

$$\frac{\partial}{\partial t}\psi(\boldsymbol{r}, t) = -\frac{\mathrm{i}}{\hbar}\left[\mathcal{F}^{-1}\left\{\frac{\hbar^2 k^2}{2m}\mathcal{F}\left\{\psi(\boldsymbol{r}, t)\right\}(\boldsymbol{k})\right\}(\boldsymbol{r}) + V(\boldsymbol{r}, t)\psi(\boldsymbol{r}, t)\right], \tag{3.107}$$

where $\mathcal{F}$ is the continuous Fourier transform, and k as always is $|\boldsymbol{k}|$. So we see that the discretised Schrödinger equation for a single particle in a potential really is the same as evaluating the continuous equation at a set of gridpoints, evaluating spatial derivatives in Fourier space, and replacing the continuous Fourier transform with its discrete equivalent.

Here is an example of how one might compute the RHS of (3.106) in Python code:

```
import numpy as np
from numpy.fft import fft2, ifft2, fftfreq

pi = np.pi
u  = 1.660539e-27 # unified atomic mass unit
```

```
m  = 86.909180*u # 87Rb atomic mass
omega = 15 # Harmonic trap frequency
hbar = 1.054571726e-34 # Reduced Planck's constant

# Space:
nx = ny = 256
x_max = y_max = 100e-6

# Arrays of components of position vectors. The reshaping is to ensure that
# when used in arithmetic with each other, these arrays will be treated as if
# they are two dimensional with repeated values along the dimensions of size
# 1, up to the size of the other array (this is called broadcasting in numpy).
x = np.linspace(-x_max, x_max, nx, endpoint=False).reshape(1, nx)
y = np.linspace(-y_max, y_max, ny, endpoint=False).reshape(ny, 1)

# Grid spacing:
dx = x[0, 1] - x[0, 0]

# Arrays of components of k vectors.
kx = 2*pi*fftfreq(nx, d=dx).reshape(1, nx)
ky = 2*pi*fftfreq(nx, d=dx).reshape(ny, 1)

# The kinetic energy operator in Fourier space (shape ny, nx).
K_fourier = hbar**2 * (kx**2 + ky**2)/(2*m)

# The potential operator in real space (shape ny, nx)
V_real = 0.5 * m * omega**2 * (x**2 + y**2)

def dpsi_dt(t, psi):
    """Return a 2D array for  the time derivative of the 2D array psi
    representing a discretised wavefucntion obeying the Schrodinger wave
    equation"""
    K_real_psi = ifft2(K_fourier * fft2(psi))
    return -1j/hbar * (K_real_psi + V_real * psi)
```

Where the example is for a time-independent potential. If the potential were time dependent, `V_real` within the function `dpsi_dt(t, psi)` would need to be replaced with a call to a function that returned an array for `V_real` at time `t`.

Starting with some initial discrete wavefunction, this could then be solved with a forward differencing scheme like fourth order Runge–Kutta (section 3.3):

```
def rk4(t, t_final, dt, psi, dpsi_dt):
    """Evolve the initial array psi_initial forward in time from time t to
    t_final according to the differential equation dpsi_dt using fourth order
    Runge-Kutta with timestep dt"""
    while t < t_final:
        k1 = dpsi_dt(t, psi)
        k2 = dpsi_dt(t + 0.5 * dt, psi + 0.5 * k1 * dt)
        k3 = dpsi_dt(t + 0.5 * dt, psi + 0.5 * k2 * dt)
        k4 = dpsi_dt(t + dt, psi + k3 * dt)

        psi[:] += dt/6.0 * (k1 + 2 * k2 + 2 * k3 + k4)

        t += dt

    return psi
```

The discretised differential equation (3.105) can also be solved using a split-step method (section 3.2.4), since the Hamiltonian matches the requirements of being written as a sum of terms for which individually an eigenbasis is known (the discrete real space basis for the potential term, and the Fourier basis for the kinetic term). For example, here is how one might implement second or fourth-order split-step (only a single timestep shown):

```
def split_step2(t, psi, dt):
    """Evolve psi in time from t to t + dt using a single step of the second
    order Fourier split-step method with timestep dt"""

```

```
    # First evolve using the potential term for half a timestep:
    psi *= np.exp(-1j/hbar * V_real * 0.5 * dt)

    # Then evolve using the kinetic term for a whole timestep, tranforming to
    # and from Fourier space where the kinetic term is diagonal:
    psi = ifft2(np.exp(-1j/hbar * K_fourier * dt) * fft2(psi))

    # Then evolve with the potential term again for half a timestep:
    psi *= np.exp(-1j/hbar * V_real * 0.5 * dt)

    return psi

def split_step4(t, psi, dt):
    """Evolve psi in time from t to t + dt using a single step of the fourth
    order Fourier split-step method with timestep dt"""
    p = 1/(4 - 4**(1/3.0))

    # Five applications of second-order split-step using timesteps
    # of size p*dt, p*dt, (1 - 4*p)*dt, p*dt, p*dt
    for subdt in [p*dt, p*dt, (1 - 4*p)*dt, p*dt, p*dt]:
        psi = split_step2(t, psi, subdt)
        t += subdt
    return psi
```

In all the above code examples, a nonlinear term as in the case of the Gross-Pitaevskii equation can be included in the potential simply by adding a term `g * np.abs(psi)**2` to the potential `V_real` wherever it appears. As discussed in Section 3.2.4, the nonlinearity poses no problem for the split-step methods so long as the potential term of the Hamiltonian is evaluated as the outermost sandwich of exponentials in the second-order split step method (which comprises the sub-steps of fourth-order split-step).

3.4.2 Finite differences

Fourier split-step, or using discrete Fourier transforms to evaluate the spatial derivatives at each gridpoint in order to time-evolve using Runge–Kutta are effective and versatile numerical methods.

The use of discrete Fourier transforms in the previous section can be seen as replacing the Laplacian operator in the Schrödinger wave equation (3.74) with the equivalent operation in Fourier space:

$$\nabla^2\psi(\tilde{\boldsymbol{r}}) \approx \text{FFT}_2^{-1}\left\{-\tilde{\boldsymbol{k}}^{\odot 2} \odot \text{FFT}_2\left\{\psi(\tilde{\boldsymbol{r}})\right\}\right\}, \tag{3.108}$$

where as before $\tilde{\boldsymbol{r}}$ is a vector (of vectors) containing the discrete positions, $\tilde{\boldsymbol{k}}$ is a vector (of vectors) containing discrete k-vectors such that $\tilde{\boldsymbol{k}}^{\odot 2}$ is a vector (of scalars) containing the squared magnitudes of each k-vector, and $\odot$ represents elementwise multiplication or exponentiation of vectors. More generally for any derivative,

$$\frac{\partial}{\partial x}\psi(\tilde{\boldsymbol{r}}) \approx \text{FFT}_2^{-1}\left\{\mathrm{i}\tilde{\boldsymbol{k}}_x \odot \text{FFT}_2\left\{\psi(\tilde{\boldsymbol{r}})\right\}\right\}, \tag{3.109}$$

where $\tilde{\boldsymbol{k}}_x$ is a vector (of scalars) containing the discrete angular wavenumbers for the x spatial dimension.

Equations (3.108) and (3.109) are exact for any wavefunction $\psi(\boldsymbol{r})$ which is periodic and band-limited to the discrete Fourier space (and thus exactly representable as a vector $\boldsymbol{\psi}$ of its values at each gridpoint), which is why the Fourier method of computing derivatives this way is sometimes said to be accurate to 'infinite order' [77] in the grid spacings Δx and Δy, in contrast to fixed-order approximations to derivatives which are second, fourth, sixth order etc. In practice the Fourier method for derivatives is often used for wavefunctions which are *not* intended to be periodic (the periodicity imposed by using the method is unphysical), and so for these it has merely very high order accuracy, not actually infinite.

	$k=-3$	$k=-2$	$k=-1$	$k=0$	$k=1$	$k=2$	$k=3$
$\Delta x\left(\delta^{(2)}_{\Delta x}\right)_{i,i+k}$			$-\frac{1}{2}$	0	$\frac{1}{2}$		
$\Delta x\left(\delta^{(4)}_{\Delta x}\right)_{i,i+k}$		$\frac{1}{12}$	$-\frac{2}{3}$	0	$\frac{2}{3}$	$-\frac{1}{12}$	
$\Delta x\left(\delta^{(6)}_{\Delta x}\right)_{i,i+k}$	$-\frac{1}{60}$	$\frac{3}{20}$	$-\frac{3}{4}$	0	$\frac{3}{4}$	$-\frac{3}{20}$	$\frac{1}{60}$
$\Delta x^2\left(\delta^{2(2)}_{\Delta x}\right)_{i,i+k}$			1	-2	1		
$\Delta x^2\left(\delta^{2(4)}_{\Delta x}\right)_{i,i+k}$		$-\frac{1}{12}$	$\frac{4}{3}$	$-\frac{5}{2}$	$\frac{4}{3}$	$-\frac{1}{12}$	
$\Delta x^2\left(\delta^{2(6)}_{\Delta x}\right)_{i,i+k}$	$\frac{1}{90}$	$-\frac{3}{20}$	$\frac{3}{2}$	$-\frac{49}{18}$	$\frac{3}{2}$	$-\frac{3}{20}$	$\frac{1}{90}$

Table 3.1: Matrix elements [78] for some finite-differencing schemes for first ($\delta^{(n)}_{\Delta x}$) and second ($\delta^{2(n)}_{\Delta x}$) derivatives using central finite differences of various orders n for uniform grid spacing Δx. All finite difference matrices are banded; each column here shows the matrix elements of k^{th} diagonal, which are all identical. Elements outside of each matrix's band are left blank. Each matrix element is shown multiplied by factors of Δx for clarity.

In any case, such high accuracy is not often necessary—if one is using only an $\mathcal{O}(\Delta t^4)$ accurate timestepping scheme, then the timestepping may be the limiting factor in overall accuracy and it might be wise to decrease the accuracy of computing spatial derivatives if there is otherwise a benefit to doing so.

To that end, the Fourier method of derivatives may be replaced with *finite differences* instead. Although finite differences are usually derived as approximations to derivatives directly from the definition of the derivative without reference to discrete Fourier transforms, they can be considered fixed-order approximations to the Fourier method [77]. Thus operators whose form in Fourier space corresponds to a derivative of some order can be approximated with finite differences:

$$U^\dagger_{\text{DFT2}} k_x U_{\text{DFT2}} \psi(\tilde{\boldsymbol{r}}) = -\mathrm{i}\delta^{(n)}_{\Delta x} \otimes \mathbb{I}_{n_y} \psi(\tilde{\boldsymbol{r}}) + \mathcal{O}(\Delta x^n) \tag{3.110}$$

where k_x is the diagonal matrix of the discrete angular wavenumbers for the x spatial dimension and $\delta^{(n)}_{\Delta x}$ is the matrix representing n^{th} order finite difference approximation to the first derivative using grid spacing Δx. The matrix elements of this and some other central finite differences are shown in Table3.1.

The fact that the finite difference matrices are banded allows them to be computing by applying a 'stencil' to a discrete state vector, computing an approximation to some linear sum of derivative operators at each point of discrete space by consideration of only that point and a small number of surrounding point. For example, a $\mathcal{O}(\Delta x^2)$ approximation (assuming $\Delta x = \Delta y$) to the kinetic energy operator in two dimensions may be evaluated at each point as:

$$K_{\text{real}}\psi(\tilde{\boldsymbol{r}}) = U^\dagger_{\text{DFT2}} \frac{\hbar^2(k_x^2 + k_y^2)}{2m} U_{\text{DFT2}} \psi(\tilde{\boldsymbol{r}}) \tag{3.111}$$

$$= -\frac{\hbar^2}{2m}\left[\delta^{2(n)}_{\Delta x} \otimes \mathbb{I}_{n_y} + \mathbb{I}_{n_x} \otimes \delta^{2(2)}_{\Delta y}\right]\psi(\tilde{\boldsymbol{r}}) + \mathcal{O}(\Delta x^2) \tag{3.112}$$

$$\Rightarrow \left(K_{\text{real}}\psi(\tilde{\boldsymbol{r}})\right)_{ij} = -\frac{\hbar^2}{2m}\Big[-4\psi(x_i, y_j) + \psi(x_{i-1}, y_j) + \psi(x_{i+1}, y_j) + \psi(x_i, y_{j-1}) + \psi(x_i, y_{j+1})\Big] + \mathcal{O}(\Delta x^2). \tag{3.113}$$

Thus the kinetic energy operator, when approximated using finite differences, is an example of an operator that can be written as a sum of banded operators acting on

different subspaces of the total Hilbert space—the identity matrices in (3.112) each leave a part of the Hilbert space untouched. As mentioned in Section 3.2.4, this in principle considerably reduces the computational cost of applying the approximate kinetic energy operator to a discretised state vector. Fourier transforms, when computed with the fast Fourier transform algorithm, are already less computationally expensive than a general matrix-vector multiplication, that is, the fast Fourier transform allows one to multiply the matrix U_{DFT2} in (3.111) by a vector considerably faster than $\mathcal{O}(n_x^2 n_y^2)$, which would be the computational time-complexity for a general $n_x n_y \times n_x n_y$ matrix. Firstly, a two-dimensional discrete Fourier transform can also be written as the sum of two one-dimensional transformations operating on different subspaces:

$$U_{\text{DFT2}} = U_{\text{DFT},x} \otimes \mathbb{I}_{n_y} + \mathbb{I}_{n_x} \otimes U_{\text{DFT},y}, \tag{3.114}$$

where $U_{\text{DFT},x}$ and $U_{\text{DFT},y}$ are the unitaries for one-dimensional discrete Fourier transforms in the x and y dimensions respectively. So even if $U_{\text{DFT},x}$ and $U_{\text{DFT},y}$ were arbitrary matrices, this already would reduce the cost of multiplying U_{DFT2} by a vector to $\mathcal{O}(n_y n_x^2 + n_x n_y^2)$ But they are not arbitrary matrices—each of these one-dimensional Fourier transforms has computational cost $\mathcal{O}(n \log n)$ using the FFT algorithm [79, p. 600], where n is the number of points in the relevant dimension, resulting in an overall cost of $\mathcal{O}(n_y n_x \log n_x + n_x n_y \log n_y)$ for applying the two-dimensional Fourier transform U_{DFT2} to a vector. Finite differences improves on this further. As discussed in Section 3.2.4, since our finite-differences approximation to the kinetic energy operator can written as the sum of banded matrices operating on different subspaces, the computational cost of applying it to a vector is $\mathcal{O}(b n_x n_y)$, where b is the bandwidth of the banded matrices, which depends on which order accuracy is used (for example, for second order finite-differences $b = 1$). This is faster than the Fourier method of computing the kinetic energy operator by a factor of $\mathcal{O}(\log n_x + \log n_y)$. Whilst this seems considerable, the difference is hard to observe in practice. On ordinary computers the number of points needs to be increased so much in order to measure any difference in speed between the algorithms that the data no longer fits in CPU cache and copying the data to and from main memory becomes the bottleneck. Although copying data from memory is a linear-time process, the coefficient of that linear time is large enough to make the asymptotic speed of finite differences vs. Fourier transforms not relevant for ordinary computers at the present time.

No, the practical advantages of finite differences compared to Fourier transforms do not come down to single-core speed. Rather, they are:

1. Being banded matrices, the finite-difference approximations to the kinetic energy operator can be multiplied by a vector, or exponentiated and applied to a vector, in parallel on a cluster computer or GPU using the techniques discussed in Section 3.2.4, whereas the fast Fourier transform is less efficiently parallelisable [80]. The speed of finite differences can have very nice scaling with the number of CPU cores or cluster nodes used, since only b points need to be exchanged between cores at each step when applying or exponentiating the kinetic energy operator. For large problems, 'superscaling' can even be observed, whereby the speedup factor obtained by moving to multiple CPUs or compute nodes on a cluster is larger than the number of CPUs/nodes used. This is counter-intuitive, but comes from more effectively using CPU cache—by spreading the data over multiple cores, one minimises the proportion of the state vector that needs to reside in main memory instead of in (much faster) CPU cache at any one time.

2. One can intervene at the boundaries to impose boundary conditions other than periodicity. Strictly speaking, as an approximation to the Fourier method of computing derivatives, the indices for the matrix elements as given in Table3.1 should

be read as wrapping around to the other side of the spatial region whenever they would go out of bounds—that is, $(\delta^{2(n)}_{\Delta x})_{i,i+k}$ should be read as $(\delta^{2(n)}_{\Delta x})_{i,(i+k) \mod n_x}$. However, as mentioned, periodicity is often an undesired consequence of the Fourier method of derivatives. Alternatively one can simply omit these matrix elements that would couple spatial points across the boundary, which has the result of imposing zero boundary conditions instead of periodic. Judicious deletion of matrix elements at other points in space can also be used to impose zero boundary conditions elsewhere, equivalent to an infinite potential barrier which would otherwise be numerically troublesome if done with the potential energy term of the Hamiltonian. Other interventions in the application of the kinetic energy operator can be used to impose other boundary conditions such as constant-value, constant-gradient, etc., whereas the Fourier method is less flexible in this regard.

3. Finite differences are compatible with non-uniform grids, whereas the Fourier method is limited to uniform grids. Non-uniform grids imply different matrix elements [78] for the finite difference operators, but are otherwise treated exactly the same. This allows more dense placement of gridpoints in regions where wavefunctions may have finer spatial structure, without having to waste computational power on regions of space where the wavefunction is known to have only coarser structure. An example is an electron in a Coulomb potential, an accurate simulation of which would need to capture fine details at small radii but less detail at larger radii. A transformation into spherical coordinates with a non-uniform grid for the radial coordinate could be well treated with finite differences.

4. Finite differences are compatible with the use of split-step methods with some operators that are diagonal in neither Fourier nor real space. For example, the real-space representation of the operator for the z component of angular momentum is $L_z = -i\hbar\left(x\frac{\partial}{\partial y} - y\frac{\partial}{\partial x}\right)$. With diagonal matrices X and Y being used for the $\hat{x}$ and $\hat{y}$ operators in accordance with the pseudospectral method, and finite differences being used to approximate the derivatives, the result is a banded matrix representation of L_z, compatible with the techniques from Section 3.2.4 for reducing the problem to that of many small matrices instead of one large one. There is a little more complexity; L_z cannot be represented as a sum of operators acting on the x and y subspaces separately—instead, each term in L_z is a *product* of operators that act on different subspaces. Consider the first term of a discretised version of L_z using fourth-order finite differences. It can be diagonalised in the following way:

$$-i\hbar X \otimes \delta^{(4)}_{\Delta y} = -i\hbar\left(\mathbb{I}_{n_x} X \mathbb{I}_{n_x}\right) \otimes \left(Q^\dagger_{\delta_y} D_{\delta_y} Q_{\delta_y}\right), \tag{3.115}$$

where Q_{δ_y} and D_{δ_y} are the unitary and diagonal matrices that diagonalise $\delta^{(4)}_{\Delta y}$ (X is already diagonal, and so is 'diagonalised' by the identity matrix for the x subspace). $\mathbb{I}_{n_x}$ and X act on the x subspace, whereas Q_{δ_y} and D_{δ_y} act on the y subspace, and since matrices operating on different subspaces commute, this can be rearranged to:

$$-i\hbar X \otimes \delta^{(4)}_{\Delta y} = \left(\mathbb{I}_{n_x} \otimes Q^\dagger_{\delta_y}\right)\left(-i\hbar X \otimes D_{\delta_y}\right)\left(\mathbb{I}_{n_x} \otimes Q_{\delta_y}\right), \tag{3.116}$$

yielding a diagonalisation of the original matrix that can be used to apply an exponentiation of the original matrix to a vector with matrix-vector multiplications only in the x any y subspaces and not the total Hilbert space. Here, the matrix-vector multiplication in the x subspace is the identity, but more generally the above

idea can be used to exponentiate any operator that can be written as the product of operators that act on different subspaces:

$$\mathrm{e}^{A\otimes B} = \left(Q_A^\dagger \otimes Q_B^\dagger\right) \mathrm{e}^{(D_A \otimes D_B)} \left(Q_A \otimes Q_B\right). \tag{3.117}$$

Since X is already diagonal, $\delta^{(4)}_{\Delta y}$ can be written as the sum of two block-diagonal matrices as described in Section 3.2.4, allowing (3.116) to be evaluated as the sum of two terms $-\mathrm{i}\hbar X \otimes \delta^{(4)}_{\Delta y} = -\mathrm{i}\hbar(X \otimes \delta_{\text{even}} + X \otimes \delta_{\text{odd}})$, one for each of the block-diagonal matrices δ_{even} and δ_{odd}. The diagonalisation of each term then has matrix-vector products taking place in a space of the size of each block, that is, in the expression

$$-\mathrm{i}\hbar X \otimes \delta_{\text{even}} = \left(\mathbb{I}_{n_x} \otimes Q^\dagger_{\text{even}}\right)\left(-\mathrm{i}\hbar X \otimes D_{\text{even}}\right)\left(\mathbb{I}_{n_x} \otimes Q_{\text{even}}\right), \tag{3.118}$$

the unitary Q_{even} is also block-diagonal. This splitting allows for parallel application of the exponentiation of L_z to a vector as well as speeding up single-core computation on account of the smaller matrices.

However If a matrix that were not diagonal were present instead of X, such as another derivative operator, then if we wanted to split *both* matrices each into a sum of two block-diagonal matrices, equation (3.117) would become *four* terms rather than two, and the flow of data for a parallel computation would be somewhat more complicated. For a term in the Hamiltonian that is a product of n operators acting on different subspaces, the number of terms obtained by splitting them all in this way grows exponentially with n. But, when all but one operator in the product is diagonal already, as is the case for angular momentum operators, then splitting can be done as normal.

To conclude this section, there are clear advantages to using finite differences as opposed to Fourier transforms when it comes to parallelisability, boundary conditions, and non-uniform grids, but if these are not a concern then both Fourier transforms and finite differences run at approximately equal speeds in practice, meaning one should use whichever is easiest to implement.

Many of these advantages of finite differences over Fourier methods are also enjoyed by the finite element discrete variable representation (FEDVR) method, discussed in the next section. Whilst it's also claimed that FEDVR has a number of advantages over finite-differences [65, 67], I'll argue that most of the comparisons don't stand up to scrutiny, and that finite differences are often still the right choice for the contexts in which FEDVR is argued to be superior.

3.4.3 Stability and the finite element discrete variable representation

Another method of spatial discretisation is the finite element discrete variable representation (FEDVR). As the name suggests, it is a finite element method, using a set of basis functions within each of a number of spatially distributed *elements*, with adjacent elements linked at their boundaries. Here I will not provide a self-contained description of the FEDVR method itself, more detail can be found in [60, p. 285] and specifically in the context of Bose–Einstein condensation, [65, 67]. I'll instead introduce the points that are relevant to the conclusion that I drew regarding the method for the purposes of time-evolving wavefunctions, which is that all practicalities considered, it is less useful than simple central finite differences for these problems.

As a finite element method, FEDVR divides space into a number of 'elements', joined to their neighbouring elements at their edges, within each of which the function being solved for is represented as a linear sum of basis functions. Finite element methods such as

this are useful for problems with irregular boundary conditions, or requiring variable grid sizes, as the elements need not have the same size (area/volume, etc.) as each other, and parameters on which the accuracy of the numerical method depends can be varied from element to element, such that precision is high where it is needed and low where it is not. In addition, the basis functions used within the elements can be chosen so that conserved properties of the differential equation are also inherently conserved by the simulation resulting in a *geometric integrator*. The FEDVR method though does not make use of this possibility, although it can be used with split-step methods for time propagation, which preserve the wavefunction's norm.

Within each element in the FEDVR method, the wavefunction is represented as a linear sum of a set of polynomial basis functions, specially chosen to have some desirable properties. The polynomials are not orthogonal, but at a set of gridpoints, all but one of them is zero, meaning that as with the Fourier pseudospectral method, they can be used as a pseudospectral basis—computing how much of each polynomial to include in the representation of a wavefunction by using only the wavefunction's value at the gridpoint at which each polynomial is zero, and computing the matrix elements of operators using approximate integration based on a discrete sum taking into account only those same gridpoints.

So far (within each element at least) what I have described does not sound too different to the Fourier pseudospectral method. But in the discrete variable representation—which is what the method used within each element of FEDVR is called when used by itself—the gridpoints are not equally spaced. Rather they are more densely packed toward the edges of the element, and least densely packed in the middle of each element. The locations of the gridpoints are chosen according to a *quadrature rule*, which is a method of approximating the definite integral of a function as a weighted sum of the function's values at a specific set of points. There are many quadrature rules, each with a different set of points and weights, but a common feature to them is that the points are more closely spaced at the edges of the integration region. In the context of the pseudospectral method, the chosen quadrature rule is what we are using when we are evaluating integrals based only on the function's values at discrete points. For equally spaced points, the weights are all equal, but for unequally spaced points they are not (and vary depending on the quadrature rule in use).

The use of unequally spaced points increases the accuracy of integrals evaluated this way, to the extent that the result will be exact if the integrand is a polynomial of degree less than a given degree that depends on the quadrature scheme. One would think that equally spaced points would produce the most accurate approximate integrals, but this is not true: approximating functions as polynomials equal to the value of the function at a discrete set of points is vulnerable to Runge's phenomenon—spurious oscillations in the approximation near the edges of the region[24]. The oscillations are minimised, however, if the density of points increases near the edge, specifically if the density of points approaches $1/\sqrt{1-x^2}$ for the normalised integration region $x \in [-1, 1]$ as the number of points goes to infinity [81].

[24] Note that although finite differences are also based on polynomial interpolation, *central* finite differences are not vulnerable to Runge's phenomenon because the interpolated polynomials at the 'edge' of the region are never used for anything: a different polynomial is used for each point, with each polynomial and its derivatives only ever being evaluated at its central interpolation point.

By choosing a quadrature scheme with two points located exactly on each edge of the integration regions, such as *Gauss–Lobatto* quadrature [65], the elements of the FEDVR method can be joined together, with adjacent elements sharing these edge points (see Figure 3.8). This allows one to represent a wavefunction as a list of coefficients for how much of each basis polynomial in each element contributes to it, with the extra condition that some of these coefficients—those corresponding to the edge points—must be equal in order to ensure the wavefunction is continuous across the boundaries between elements.

With some care taken in regard to the points shared between the elements, the FEDVR method provides one with a means of approximating wavefunctions as a finite sum of basis

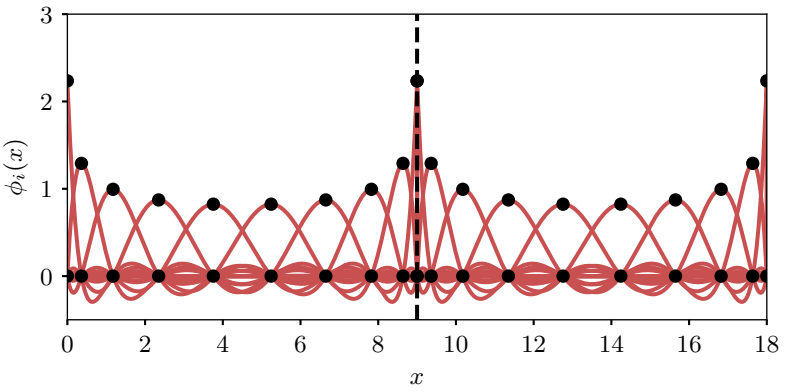

Figure 3.8: An example of the normalised, but non-orthogonal basis polynomials used in the finite-element discrete variable representation, shown here for ten-point Gauss–Lobatto quadrature and two elements. Note that each basis polynomial is non-zero at exactly one point and zero at all others, though it can be non-zero in between points. Outside of its element, a given basis function is defined to be zero, except for the so-called 'bridge functions', which are non-zero in two adjacent elements but zero everywhere else, and surprisingly have a discontinuous derivative at the boundary—necessitating some care in evaluating matrix elements of differential operators.

functions, and of approximately calculating the matrix elements of operators in this basis. After that, one can simply apply the operators to the state vectors in order to compute time derivatives, and propagate in time using fourth order Runge–Kutta (section 3.3) or similar, or one can exponentiate the operators, all at once or one at a time, as part of a split-step method (section 3.2.4). The FEDVR method does not require anything in particular about time propagation—like finite differences or the Fourier pseudospectral method, it is only a way of spatially discretising the differential equation you are trying to solve.

Now we arrive at what makes the FEDVR method exciting for those who want to massively parallelise their simulations. When one calculates the matrix representation of, say, a one-dimensional differential operator in the FEDVR basis, one gets matrices that look like Figure 3.9, with an almost-block-diagonal form.

This is because when the state vector is acted upon by some operator, the result for most points in the resulting vector depends only on the points in the input vector *within the same element*. The exceptions are the points shared between elements—for which the value in the output vector depends on the points in the input vector in *both* adjacent elements, which is why the matrix is not perfectly block diagonal.

Why is this exciting? Well, it means that the operator can be written as a sum of two block diagonal matrices, similar to the splitting of finite-difference matrices in Section 3.2.4, and in the same way allows the matrix or its exponentiation to be efficiently applied to a state vector in parallel. The difference between this and the case of finite differences is the number of points of overlap between adjacent blocks, which corresponds to the amount of data that must be transferred between parallel threads or cluster nodes at each step during a parallel computation. In finite differences, the number of points that must be exchanged at boundaries in each step or sub-step of whichever time propagation method is used is equal to the bandwidth of the matrix. In FEDVR, so long as the boundaries of the regions of space assigned to each thread or cluster node align with boundaries between elements, *only one* point need be exchanged. Increasing the

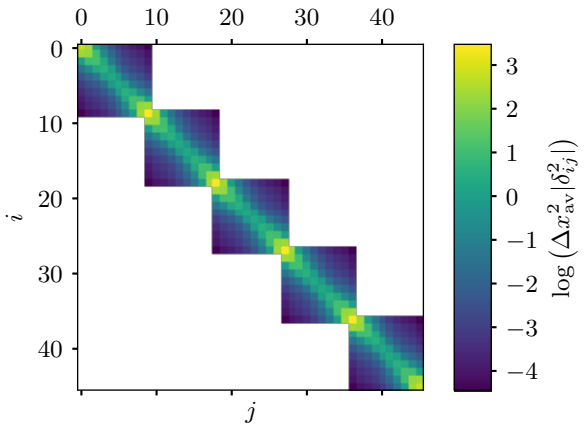

Figure 3.9: The matrix form of the second derivative operator in FEDVR, for five elements each with ten DVR points. The colour scale is logarithmic, showing the log of the absolute values of the matrix elements δ^2_{ij} scaled by Δx^2_{av}, where Δx_{av} is the average spacing between the unequally spaced grid points. Zero matrix elements are shown as white. One can see that the second derivative at each grid point is computed using only the points within the same element, except for the points shared by adjacent elements, at which the second derivative depends on points in both adjacent elements.

number of points within each element—but reducing the number of elements so as to keep the total number of points constant, decreases the discretisation error of the method, but still, only one point needs to be exchanged at boundaries. This contrasts with finite differences, for which increasing the order of the finite difference scheme increases the matrix bandwidth and hence increases the number points required to be exchanged at the boundaries. So it would seem that FEDVR ought to scale much better in parallel implementations, which is a large part of its appeal [65, 67].

However, I've noticed that when using both FEDVR and finite differences to simulate Bose–Einstein condensates using the Gross–Pitaevskii equation, smaller timesteps are required when using the FEDVR method in otherwise comparable setups. By this I mean that, without damping, there is a timestep size at which the GPE simulated using RK4 (which is a conditionally stable algorithm) is unstable and diverges. Similarly in split-step methods, although the error is bounded, there is a timestep size at which the error rapidly grows to that bound and the wavefunction no longer approximately resembles the true solution. Below this threshold timestep, the error scales as $\mathcal{O}(\Delta t^4)$ for RK4, and $\mathcal{O}(\Delta t^4)$, $\mathcal{O}(\Delta t^2)$, $\mathcal{O}(\Delta t)$ for fourth, second, and first order split-step respectively, as expected. But in practice, all these methods are so accurate that one desires to use the largest timestep one can without this blowup (or soft-blowup in the case of the unitary split-step methods) occurring during the time interval one wants to simulate. And my observation was that the threshold timestep is always smaller for FEDVR than for finite differences or the Fourier method for determining spatial derivatives, necessitating smaller timesteps for stability or, in the case of the unitary methods, reasonable accuracy.

So why is this? For RK4, what is the stability criterion and why might FEDVR violate it more easily than finite differences? And how might we understand the sudden decrease in accuracy of the split-step methods at a similar threshold timestep size, despite them being unconditionally stable?

The stability criterion for RK4 when applied to a linear differential equation of the form:

$$\frac{\mathrm{d}\psi}{\mathrm{d}t} = A\psi, \tag{3.119}$$

where A is a matrix with all imaginary eigenvalues (as is the case for Hamiltonian evolution where $A = -\frac{\mathrm{i}}{\hbar}H$), is [82]:

$$\Delta t < \frac{2\sqrt{2}}{\rho(A)} \tag{3.120}$$

where $\rho(A)$ is the *spectral radius* of A, defined as the absolute value of the largest (by absolute value) eigenvalue of A.

The Gross–Pitaevskii equation is not linear, but we'll put that to the side for the moment—it will be relevant shortly. In the linear case of the Schrödinger wave equation, an upper bound for the spectral radius of A can be computed from the absolute eigenvalue from the kinetic term, plus the maximum absolute eigenvalue from the potential term. Using the Fourier method to compute spatial derivatives, the largest eigenvalue of the kinetic part of the Hamiltonian is that of the Nyquist mode with $k_{\mathrm{Nyquist}} = \pi/\Delta x$ for grid spacing Δx, leading to:

$$\rho(A) \leq \left|-\frac{\mathrm{i}}{\hbar}\frac{\hbar^2 k_{\mathrm{Nyquist}}}{2m}\right| + \left|-\frac{\mathrm{i}}{\hbar}V(\boldsymbol{r})\right|_{\max} \tag{3.121}$$

$$\Rightarrow \Delta t < \frac{2\sqrt{2}\hbar}{\frac{\hbar^2\pi^2}{2m\Delta x^2} + |V(\boldsymbol{r})|_{\max}}. \tag{3.122}$$

In the limit of small Δx, the kinetic term dominates and the stability criterion becomes:

$$\Delta t < \frac{4\sqrt{2}m\Delta x^2}{\pi^2\hbar}, \tag{3.123}$$

whereas in the limit of large potential:

$$\Delta t < \frac{2\sqrt{2}\hbar}{|V(\boldsymbol{r})|_{\max}}. \tag{3.124}$$

These results match our intuition somewhat in terms of dynamical phase evolution—eigenvalues of the Hamiltonian are energies and determine how quickly the elements of the state vector accumulate dynamical phase. For each circle around the complex plane that a dynamical phase of 2π entails, we expect to require at least a few timesteps to resolve said circle, and the above shows that 'a few' means $2\sqrt{2}$. When this dynamical phase evolution is in Fourier space, an accumulated phase of 2π will appear in real space as that component having propagated a distance of one wavelength, so the intuition here is that several timepoints are required in the time interval during which the fastest wave (also the Nyquist mode) propagates a distance of one wavelength at its phase velocity.

As a side note, if it is known which term of the Hamiltonian dominates the stability criterion, that term can be removed by use of an *interaction picture* (discussed in Section 3.6), essentially treating the dynamical phase evolution due to that term analytically. And if the term is not constant, but is *almost* constant, one can still treat *most* of the dynamical phase evolution analytically, transforming the differential equation onto one with much smaller eigenvalues for that term, in both cases allowing one to take potentially much larger timesteps due to the above reasoning. Fourth order Runge–Kutta in the interaction picture (RK4IP) [83] uses an interaction picture to treat the kinetic term of the Schrödinger or Gross–Pitaevskii equation analytically, whereas my 'fourth

order Runge–Kutta in an instantaneous, local interaction picture' method presented in Section 3.6 removes (most of) the potential term. Both methods allow larger timesteps to be taken, but in different circumstances depending on which term is dominating the Hamiltonian.

The problem with FEDVR then, is that it requires smaller timesteps than finite differences because its kinetic energy operator has larger eigenvalues than an equally accurate finite difference method. How much larger?

In order to make a fair comparison, we need to know how many DVR basis functions are required per element in order to compute equally accurate second derivatives as a given finite difference scheme. With N points per element, FEDVR represents the state vector within each element in a spectral basis of polynomials up to degree $N-1$. Therefore a polynomial of degree $N-1$ or less can be represented exactly, any other function is subject to truncation error[25]. If one varies the number of points per element, varying the number of elements in order to keep the total number of points constant, the truncation error in representing an arbitrary function in this basis is therefore $\mathcal{O}(\Delta x^N)$, where Δx is either the size of each element, or equivalently the average spacing between grid points (these two differing only by a constant factor if the total number of points is held constant).

[25] Note that this truncation error is distinct from that of evaluating *integrals* using the Gauss–Lobatto quadrature rule, which is exact for integrands that are polynomials of degree $2N-2$ or less.

In FEDVR the derivative operator, regardless of whether its matrix elements are computed with integrals or the quadrature rule, is exact [65]. The relevant error in a derivative of a wavefunction is therefore determined by this truncation error of representing it in the spectral basis in the first place:

$$\psi_{\text{approx}}(x_i) = \psi_{\text{exact}}(x_i) + \mathcal{O}(\Delta x^N) \tag{3.125}$$

$$\Rightarrow \psi''_{\text{approx}}(x_i) = \psi''_{\text{exact}}(x_i) + \mathcal{O}(\Delta x^{N-2}). \tag{3.126}$$

Central finite difference approximations to second derivatives on the other hand have error $\mathcal{O}(\Delta x^{N-1})$ where N is the total number of points used to compute the derivative at each point (for example the three-point central finite difference rule for second derivatives has error $\mathcal{O}(\Delta x^2)$, the five-point rule is accurate to $\mathcal{O}(\Delta x^4)$, etc.). With this knowledge we can translate our question

> Which is less computationally intensive to simulate the GPE or Schrödinger wave equation: m^{th}-order accurate FEDVR or m^{th}-order accurate finite differences?

to

> which is less computationally intensive to simulate the GPE or Schrödinger wave equation: $m+2$ points-per-element FEDVR or $m+1$ point central finite differences?

The argument in favour of FEDVR is that as m grows, finite differences require an increasing number of points to be exchanged at the boundaries between cluster nodes/threads etc. in a parallel implementation, whereas so long as each boundary between spatial regions allocated to different cluster nodes aligns with the boundary between two elements, only one point need be exchanged per timestep in FEDVR, no matter how many points per element there are. Therefore, in the limit of high accuracy, FEDVR wins, it seems.

The problem with this argument is that it compares only the amount of work that needs to be done *per timestep*. But, since the allowed timestep size required for stability (at least for fourth order Runge–Kutta, I'll argue shortly why I think this generalises to other timestepping schemes as well) depends on the spectral radius of the kinetic energy operator, and the kinetic energy operator is not the same as m grows, more work is needed to show which method is least computationally expensive per unit *simulation time*.

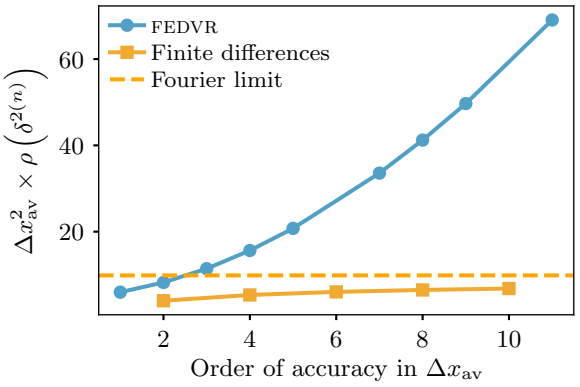

Figure 3.10: Scaling of the spectral radius (maximum absolute eigenvalue) of the second derivative operator with respect to the order of accuracy for finite differences and FEDVR. Results were numerically computed by constructing a 721×721 matrix (chosen to allow various combinations of number of DVR points per element whilst holding total number of points constant) for each order of accuracy and numerically diagonalising. The spectral radius for FEDVR can be seen to scale quadratically with the order of accuracy, whereas for finite differences the maximum eigenvalue approaches that of the Fourier method.

The spectral radius of the kinetic energy operator when approximated using central finite differences does not grow without limit as the number of points used to compute derivatives increases—indeed, its eigenspectrum converges to that of the Fourier method (since the Fourier method can be considered the limit of 'infinite order' finite differences [77]), with maximum eigenvalue equal to the kinetic energy of the Nyquist mode. The kinetic energy operator approximated with FEDVR on the other hand does not have a bounded spectral radius as one increases the number of points per element whilst holding the total number of gridpoints constant. Instead its spectral radius increases quadratically with the number of points (equivalently with the order of accuracy of the derivatives), as shown in Figure 3.10.

This result ought to be expected, at least qualitatively. The density of points in FEDVR is higher toward the edges of the elements than in their centres, and if one increases the number of points per element whilst decreasing the number of elements so as to hold the total number of points approximately constant, the smallest spacing between any two adjacent points will be inversely proportional to the number of points per element. We already know that in high order finite differences or the Fourier method, the spectral radius of the kinetic energy operator is simply the kinetic energy of the Nyquist mode, and being the mode described by a wave with half a wavelength spanning one grid spacing, its kinetic energy is proportional to the square of the grid spacing. It not therefore surprising that in FEDVR when the grid spacing is linearly smaller—albeit locally—that the spectral radius of the kinetic energy operator is quadratically larger. The same result as above could be obtained by asking: "what is the smallest grid spacing, and what is the kinetic energy of the Nyquist mode corresponding to that grid spacing?", and then declaring the stability criterion to be that one must have at least a few points per period of the frequency corresponding to that energy.

Since the spectral radius of FEDVR operators grows faster with increasing order of accuracy in derivative operators than finite differences, FEDVR requires smaller timesteps for stability when used with RK4. Even at low order accuracy, finite difference operators have smaller spectral radii, and so at all levels of accuracy RK4 is stable with finite differ-

ences for larger timesteps than with FEDVR. In the limit of high accuracy then, compared to finite differences FEDVR requires *quadratically* more timesteps to be taken for stability, whereas it requires only *linearly* fewer points to be transferred at the boundaries between threads or cluster nodes per timestep. The larger number of timesteps is therefore the dominant effect, more than cancelling out the benefit of having to transfer fewer points at boundaries per timestep than with finite differences. Indeed, even if transferring data between nodes or threads is the bottleneck of a parallel simulation, *more* points are being transferred all up with FEDVR—they are just spread out over more timesteps.

That's fourth order Runge–Kutta. What about split-step methods? The split-step methods discussed in Section 3.2.4 are unconditionally stable and have bounded error. However, their error can still be large enough to make the results not useful. As shown in (3.46), the leading error term in first-order split-step is $1/2\hbar^2[V,K]\Delta t^2$ where $[V,K]$ is the commutator between the (discretised) kinetic and potential energy operators. The leading term of V is proportional to the discretised $\hat{x}$ operator X, and the kinetic energy operator is proportional to an n^{th} order accurate discretised second derivative operator $\delta^{2(n)}$. Therefore the error per timestep scales with $[X,\delta^{2(n)}]\Delta t^2$. Although this expression is not bounded, the error in first-order split-step is nonetheless bounded because this error appears as the argument of a complex exponential. When expanding the complex exponential as its Taylor series, this error term is the leading term, but when it grows it leads merely to a complex exponential with unbounded phase error, not to unbounded absolute error. Nonetheless unbounded phase error still makes the results of a simulation unlikely to be useful.

The largest (by absolute value) eigenvalue of this commutator sets the rate at which erroneous dynamical phase is accumulated by some eigenstate of the commutator. When this phase becomes comparable to 2π per timestep, the simulation has clearly lost any semblance of accuracy, despite still being technically 'stable'. Therefore as before, the spectral radius of the matrix of this commutator can be used to define a *pseudo*-stability criterion, such that when the spectral radius of $[V,K]\Delta t^2/2\hbar^2$ becomes comparable to unity, the error will dominate the simulation results. Although the value of the spectral radius will depend on the details of the potential energy operator V for the particular problem, we can still ask how it scales with increasing order of accuracy of K given that X is the leading term of V. This is done in Figure 3.11, comparing the spectral radius of the commutator when using derivative operators of various orders in both finite differences and FEDVR approximations.

The result is that as with RK4, the pseudo-stability criterion allows at all orders of accuracy for larger timesteps when using finite differences than when using FEDVR, and that the required timestep is bounded from below when increasing the order of accuracy of finite differences, but can become arbitrarily small for increasing accuracy of FEDVR. However the situation is not so dire for FEDVR when used with split-step as it is with RK4 timestepping. Because the error term in split-step is this commutator multiplied by Δt^2, the timestep required for pseudo-stability scales inversely with the *square root* of the spectral radius of the commutator—not with the spectral radius itself as with RK4. Furthermore the spectral radius of the FEDVR commutator increases only *linearly* with increasing order of accuracy, not quadratically as with RK4. Therefore if transferring data between nodes (with time cost proportional to the amount of data transferred) is the bottleneck of one's simulation, then with increasing accuracy, FEDVR *does* win out. For high enough order of accuracy n, a factor of $\sqrt{n}$ more timesteps are needed with FEDVR compared to finite differences, but a factor of n *fewer* points are sent per timestep. This results in a factor of $1/\sqrt{n}$ fewer points being sent per unit simulation time with FEDVR as compared to finite differences. The question of which method is faster then comes down to which is more costly: extra timesteps, or extra data transfer? Using FEDVR will save you a factor of $\sqrt{n}$ in data transfer at a cost of a factor of $\sqrt{n}$ in the number of timesteps. Given InfiniBand interconnects on cluster computers with tens of gigabits per second

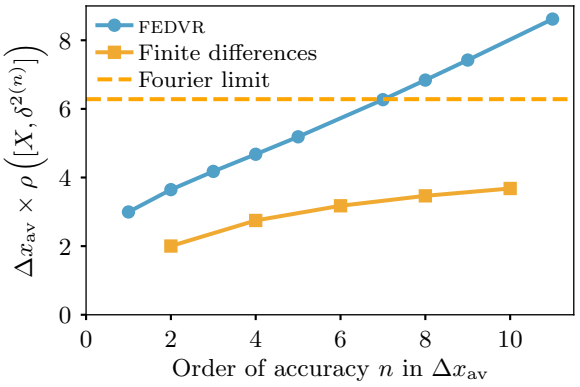

Figure 3.11: Scaling with order of accuracy of the spectral radius of the commutator of position and second derivative operators in the pseudospectral finite difference method and FEDVR. The spectral radius of the commutator grows linearly with increasing order of accuracy for FEDVR, whereas it is bounded for finite differences, approaching that of the Fourier method.

of bandwidth, and latency that does not scale with the amount of data, I suspect that a $\sqrt{n}$ increase in cost of processing within nodes/threads is almost always the larger cost to pay. Therefore I suspect the benefits of FEVDR for parallel simulations of the Schrödinger wave equation or Gross–Pitaevskii equation are limited.

3.4.4 Nonlinearity considerations

The above arguments about stability all disregarded the nonlinearity of the Gross–Pitaevskii equation. Although the stability of a numerical method is very difficult to analyse for nonlinear differential equations, there is a simple argument for heuristically putting an upper bound on the timestep required in order to accurately model the effect of the GPE's nonlinear term. Essentially, density waves in the condensate propagate not at the phase velocity of the condensate wavefunction, but at its group velocity. The group velocity of the shortest possible wavelength in the system therefore sets an upper bound for the propagation of information due to the nonlinear term. In order to correctly model this term, timesteps must be short enough that one's numerical method is evaluating the nonlinear term frequently enough that fast moving density waves can't 'skip' gridpoints in between evaluations. If the smallest wavelength is that of the Nyquist mode, or for non-uniform grids the mode whose wavelength is twice the smallest grid spacing, then the maximum group velocity is:

$$v_{\mathrm{g}}(k_{\mathrm{max}}) = \frac{\partial E_{\mathrm{K}}(k_{\mathrm{max}})}{\partial p(k_{\mathrm{max}})} = \frac{\hbar k_{\mathrm{max}}}{m} \frac{\pi\hbar}{m\Delta x_{\mathrm{min}}} \tag{3.127}$$

Asking how long it takes a wave to move a distance of Δx_{min} at this velocity then results in what I'm calling the *dispersion timescale*:

$$\tau_{\mathrm{d}} = \frac{m\Delta x_{\mathrm{min}}^2}{\pi\hbar}. \tag{3.128}$$

The numerical method being used is now fairly irrelevant: one must evaluate the nonlinear term at least as often as every τ_{d} in order to be able to model it accurately, otherwise density waves may move past each other faster than they can be resolved by the

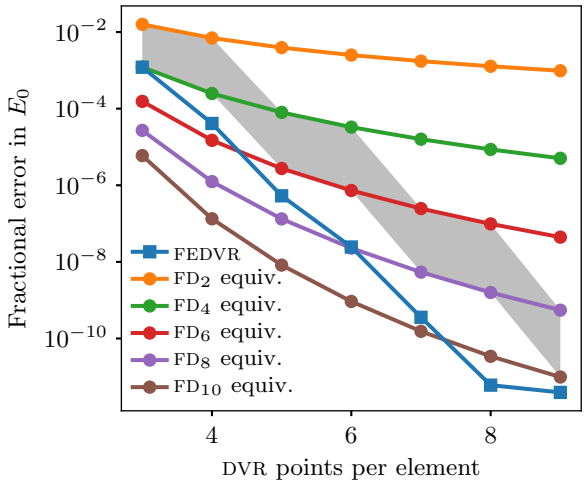

Figure 3.12: Comparison of accuracy of different discretisation methods in representing the ground state of a harmonic oscillator $V(x) = x^2$ in a spatial region $-10 < x < 10$. For the FEDVR results, 20 elements were used with a varying number of DVR points per element, leading to decreasing error in the ground-state energy with increasing number of points. For each number of DVR basis points, finite difference operators are produced using the same total number of points, but on a uniform grid. This is still not a fair comparison however, because it is not equally computationally expensive to apply a FEDVR operator or a finite differences operator to a state vector. As mentioned in earlier sections, the cost of applying operators is dependent on their bandwidth—related to how many surrounding points are coupled to each point by the operator. This comparison is made above by computing the *average* bandwidth of each FEDVR operator (since it is not the same for all points), and shading in the region between the two finite differencing orders that have bandwidths either side of the result. The resulting shaded region shows FEDVR and finite differences are quite comparable when judged by accuracy at fixed cost of applying operators, with the gap small enough to be closed by increasing the bandwidth of a finite differencing scheme only by one.

timestepping, and their interference patterns will be aliased by the too-slow timestepping, resulting in incorrect nonlinear dynamics.

Note that up to constant factors, this criterion for accurate modelling is the same as the stability criterion (3.123) for RK4 when the kinetic term dominates the Hamiltonian. Therefore although first-order split-step as argued above appears to be more forgiving to FEDVR than RK4 is, this is no longer the case when modelling the GPE as opposed to the linear Schrödinger wave equation, and although I didn't extend the argument in the previous section to second or fourth order split-step (figuring out which commutator is the leading term is much more involved), they too are subject to the same requirement to sample the nonlinear term this frequently, even if their linear pseudo-stability region might be larger.

3.4.5 Conclusion

This leaves us with little remaining benefit to FEDVR over finite differences for the Schrödinger wave and Gross–Pitaevskii equations. The argument that FEDVR scales better for parallel computation is unconvincing as argued above. The fact that it allows non-uniform grids is also not unique—central finite differences allow for non-uniform

grids as well [78]. Reference [67] compares the accuracy of finite differences to FEDVR in the context of computing the ground-state energy of a harmonic oscillator using imaginary time evolution (see Section 3.5.1), but only uses second-order finite differences in the comparison whereas higher-order FEDVR schemes are used. If this restriction was in order to hold the number of points transferred at boundaries between parallel computing units constant (since second-order finite differences require, like FEDVR, only one per timestep), then this is not an equal comparison as the number of points transferred does not limit computation time as I've shown—had the authors included comparisons with higher order finite differencing schemes, they would have resulted in comparable accuracy (see Figure 3.12) between FEDVR and finite differences, but with finite differences completing in less time owing to the larger timesteps allowed by the increased range of stability of finite differences.

Perhaps for problems with certain unusual boundary conditions or strangely shaped regions, or where one can construct a geometric integrator that conserves some quantities of interest other than merely the wavefunction norm, FEDVR may still make sense. But unless there is a compelling reason, it seems that simple finite differences, as naïve as it might seem at first, remain a practical choice for highly accurate and potentially massively parallelised simulations of the GPE.

3.5 Finding ground states

For systems with continuous degrees of freedom, such as Bose–Einstein condensates, finding ground states by directly diagonalising the Hamiltonian matrix is impractical due to the size of said matrix, which we normally avoid even constructing for systems with more than one dimension when simply propagating condensate wavefunctions in time. The pseudo-Hamiltonian for the Gross-Pitaevskii equation is not even linear, implying direct diagonalisation would not even return ground states unless one diagonalised matrices repeatedly, each with an improved estimate of the ground state until a self-consistent solution was found.

Below are two methods that are typically used instead to find ground states in these cases, as well as a generalisation (which only applies to linear systems) for finding other excited states of quantum systems.

3.5.1 Imaginary time evolution

Imaginary time evolution [84] is a robust, if somewhat inefficient method of finding ground states of quantum systems, and can be generalised to finding excited states as well (Section 3.5.3, below). The basis idea is to evolve the following differential equation:

$$\hbar \frac{\mathrm{d}\boldsymbol{\psi}}{\mathrm{d}t} = -H\boldsymbol{\psi} \tag{3.129}$$

in time instead of the time-dependent Schrödinger equation (3.3), for some wavefunction or state vector $\boldsymbol{\psi}$ and matrix/linear operator H. The name *imaginary time evolution* is due to the fact that this differential equation can be obtained by making the substitution $t \to -\mathrm{i}t$ in the time-dependent Schrödinger equation, and hence can be thought of as propagating the Schrödinger equation in (negative) imaginary time. This method is capable of evolving an initial guess for $\boldsymbol{\psi}$ into the ground state of the given Hamiltonian. The reason behind this can be seen if we rewrite (3.129) in the eigenbasis of H with eigenvalues $\{E_n\}$ and eigenkets $\{|n\rangle\}$:

$$\hbar \frac{\mathrm{d}}{\mathrm{d}t} \langle n|\psi(t)\rangle = -E_n \langle \phi_n|\psi(t)\rangle \,, \tag{3.130}$$

which has solutions:

$$\langle n|\psi(t)\rangle = \exp\left[-\frac{E_n}{\hbar}t\right]\langle n|\psi(0)\rangle\,, \tag{3.131}$$

that is, the coefficient of each eigenstate exponentially decays with a decay rate proportional its energy. After enough time has elapsed (much larger than the difference in decay timescales between the two lowest lying eigenstates), all coefficients $\langle n|\psi(t)\rangle$ will have decayed, but the coefficient corresponding to the ground state will have decayed by far the least, leading to any initial superposition evolving into one dominated by the ground state.

This method is flexible and robust, and can be used to find ground states subject to imposed conditions such as a phase winding about a point, resulting in a vortex state. It can also be used to 'smooth' candidate wavefunctions, making a guessed initial state of a condensate wavefunction more like a physically realistic one by lowering its energy somewhat, but not propagating far enough to obtain the actual ground state. This smoothing method is used in Section 3.6 to create an approximately correct density profile for a condensate wavefunction after the phase patterns for randomly distributed vortices are artificially imposed, resulting in physically realistic initial conditions for a turbulent condensate.

For practical reasons, exponential decay cannot be carried out indefinitely, as eventually all numbers underflow to zero as they become too small to be represented on a computer. In addition, the wavefunction/state vector obtained after exponential decay will not be normalised. For these reasons, one often normalises the wavefunction/state vector to unit norm in between each step of imaginary time evolution.

Normalisation must be paid attention particularly in the case of the Gross–Pitaevskii equation, since it has a nonlinear pseudo-Hamiltonian. This pseudo-Hamiltonian must always be evaluated with a normalised condensate wavefunction as input, otherwise the imaginary time evolution may, depending on the numerical method employed, produce a final condensate wavefunction which is the ground state not in the presence of the actual nonlinear term, but in the presence of one that has had one timestep worth of exponential decay applied to it. Since the magnitude of the wavefunction matters in the Gross–Pitaevskii equation, one must normalise every time a nonlinear H is evaluated in order to prevent this problem (which can also be mitigated by taking tiny timesteps to minimise the amount of decay within each timestep, but this is wasteful when larger steps could otherwise be used).

An exception to the above normalisation warning is when a first order explicit method such as the first-order split step method (Section 3.2.4) or the Euler method is used—since these methods only ever evaluate the nonlinear pseudo-Hamiltonian at the start of a timestep, normalising the wavefunction once per timestep is sufficient, whereas when using higher order methods that evaluate the pseudo-Hamiltonian at intermediate times one will have to take care to normalise more often.

The imaginary time evolution method is a *relaxation method*, and as such only the final wavefunction/state vector after a sufficient period of evolution is of interest, not the intermediate states. As such, inaccurate but fast methods such as first order split-step or the Euler method may be readily employed, and used with the largest timesteps that do not lead to numerical instability. Second order Runge–Kutta (the explicit midpoint method) can be a good balance due to being faster per-step than more accurate methods, whilst allowing larger timesteps than the Euler method due to better stability characteristics.

A final note is that energies of quantum systems are arbitrary up to a global additive constant, and so energies can be negative or positive or some mix of both in the same system. In this case, imaginary time evolution can have some or all coefficients of energy eigenstates exponentially *growing* rather than decaying. This does not modify the end result at all, as the ground-state energy, if negative, will be more negative than any other

energy and hence its coefficient will exponentially grow faster than all others. If an estimate of the ground-state energy is known ahead of time, it can be advantageous to offset the system Hamiltonian with a constant energy such that the ground-state energy is approximately zero, which minimises the amount of exponential growth/decay and allows for the largest timesteps within the stability range of the numerical method employed. One can even dynamically compute an energy or chemical potential estimate[26] to update the energy offset as the computation proceeds. Imaginary time evolution can be slow enough for non-trivial cases that these considerations can be relevant.

Since imaginary time evolution comprises the evolution of a state vector in time, it is amenable to all the parallel processing techniques we have outlined in this chapter depending on the timestepping and spatial discretisation employed.

[26] Note that the energy of a BEC using the Gross–Pitaevskii pseudo-Hamiltonian cannot be computed simply as the expectation value of the pseudo-Hamiltonian—this double counts the nonlinear term, which should be halved in such energy calculations. The chemical potential however can be computed as this expectation value.

3.5.2 Successive over-relaxation

Successive over-relaxation (SOR) [85] is a much faster method of finding ground states than imaginary time evolution, but slightly less flexible. Strictly speaking (that is, before any ad-hoc modifications), SOR is a method for solving linear systems of the form:

$$A\boldsymbol{\psi} = \boldsymbol{b}, \tag{3.132}$$

for some unknown vector (or discretised function) $\boldsymbol{\psi}$ given a vector (or discretised function) $\boldsymbol{b}$ for the right hand side and a matrix (or discretised linear operator) A for the left hand side. The method is of most benefit when A is diagonally dominant, lending SOR to use with finite difference methods. As with imaginary time evolution, SOR requires an initial guess for $\boldsymbol{\psi}$ that is repeatedly updated to produce an increasingly accurate approximate solution to the given linear system.

Successive over-relaxation proceeds by updating each element ψ_i of $\boldsymbol{\psi}$ one at a time to be more consistent with the linear system and all other estimated elements of $\boldsymbol{\psi}$, by solving the i^{th} linear equation of (3.132) for ψ_i in terms of A, b_i, and the other elements of $\boldsymbol{\psi}$:

$$\psi_i = \frac{b_i - \sum_{j\neq i} A_{ij}\psi_j}{A_{ii}}. \tag{3.133}$$

If one updates every element ψ_i of $\boldsymbol{\psi}$ in this manner, based on the other elements $\psi_{j\neq i}$ of the estimate for $\boldsymbol{\psi}$ at the previous iteration, the method that results is called the Jacobi method, with update rule:

$$\psi_{i\,\text{Jacobi}}^{(k+1)} = \frac{b_i - \sum_{j\neq i} A_{ij}\psi_j^{(k)}}{A_{ii}}, \tag{3.134}$$

where $\boldsymbol{\psi}^{(k)}$ is the k^{th} iterative estimate of $\boldsymbol{\psi}$. If one allows use of already updated elements of $\boldsymbol{\psi}$ when updating subsequent elements of $\boldsymbol{\psi}$, one obtains the more efficient Gauss–Seidel method:

$$\psi_{i\,\text{GS}}^{(k+1)} = \frac{b_i - \sum_{j\neq i} A_{ij}\psi_j^{(\text{latest})}}{A_{ii}}, \tag{3.135}$$

where $\psi_j^{(\text{latest})} = \psi_j^{(k+1)}$ if $\psi_j^{(k+1)}$ has been computed already, and $\psi_j^{(k)}$ if not. Assuming elements of $\boldsymbol{\psi}$ are updated in order of increasing j, one can split the sum into two for these two cases and write:

$$\psi_{i\,\text{GS}}^{(k+1)} = \frac{b_i - \sum_{j<i} A_{ij}\psi_j^{(k+1)} - \sum_{j>i} A_{ij}\psi_j^{(k)}}{A_{ii}}, \tag{3.136}$$

which is how the method is often presented; however, since one can update elements in any order (or equivalently, the order of the basis vectors in $\boldsymbol{\psi}$ is arbitrary), I prefer the form (3.135) as it leaves the iteration order undetermined. One reason to impose an update order other than just that of increasing index in the array storing $\boldsymbol{\psi}$ in computer memory is to minimise communication latency in parallel implementations: one may choose to update points near one or more edges of the spatial region assigned to one CPU core or cluster node first, in order that they become available to other cores/nodes sooner, before updating the interior points that other cores/nodes do not depend on. In this case of parallel processing there isn't even a strict order since elements are being updated independently by separate cores/nodes simultaneously. Gauss–Seidel and successive over-relaxation still work well in this case, they just become a little more like the Jacobi method the more independent updates are occurring on different parts of the vector $\boldsymbol{\psi}$.

The final feature of successive over-relaxation is that elements of $\boldsymbol{\psi}$ are updated not to the values given by the Gauss–Seidel method, but overshoot them by a factor α:

$$\psi_{i\,\mathrm{SOR}}^{(k+1)} = \psi_i^{(k)} + \alpha\left(\frac{b_i - \sum_{j\neq i} A_{ij}\psi_j^{(\mathrm{latest})}}{A_{ii}} - \psi_i^{(k)}\right). \tag{3.137}$$

where $\alpha < 2$ is the convergence criterion in the case of an infinite system.[27]

So what are A and $\boldsymbol{b}$ for quantum mechanics problems? One deviation from pure SOR used in cold atom physics is to allow both A and $\boldsymbol{b}$ to have dependence on $\boldsymbol{\psi}$, which in practice does not prevent the method from finding solutions. For the ground state of the Gross–Pitaevskii equation, we can write:

$$H(\boldsymbol{\psi})\boldsymbol{\psi} = \mu\boldsymbol{\psi} \tag{3.138}$$

where μ is the chemical potential of the condensate in its ground state and $H(\boldsymbol{\psi})$ is the matrix for some discretisation of the Gross–Pitaevskii pseudo-Hamiltonian, [28] to which we can then apply a single iteration of SOR by updating all elements of $\boldsymbol{\psi}$ in some order using (3.137) with $A^{(k)} = H(\boldsymbol{\psi}^{(k)})$ and $\boldsymbol{b}^{(k)} = \mu\boldsymbol{\psi}^{(k)}$ to produce an improved estimate $\boldsymbol{\psi}^{(k+1)}$ of the ground state based on the previous estimate $\boldsymbol{\psi}^{(k)}$:

$$\psi_{i\,\mathrm{SOR}}^{(k+1)} = \psi_i^{(k)} + \alpha\left(\frac{\mu\psi_i^{(k)} - \sum_{j\neq i} H_{ij}(\boldsymbol{\psi}^{(k)})\psi_j^{(\mathrm{latest})}}{A_{ii}} - \psi_i^{(k)}\right). \tag{3.139}$$

$A^{(k+1)}$ and $\boldsymbol{b}^{(k+1)}$ can then be computed based on $\boldsymbol{\psi}^{(k+1)}$ and the process repeated to provide increasingly accurate approximate solutions.

A difference between successive over-relaxation and imaginary time evolution is that SOR finds the state with chemical potential μ (replaceable by the energy E for a linear quantum system), as opposed to the ground state. In this way successive over-relaxation doesn't tell you what the ground-state energy is, it requires that as input. Typically to find a Gross–Pitaevskii ground state I will analytically estimate the chemical potential for a given number of atoms or peak density, or whatever my requirement is, using the Thomas–Fermi approximation, and then find the state with that chemical potential using SOR. This results in minor differences compared to the peak density or atom number originally targeted, but the requirement is not usually strict enough for this to be of consequence. For the Gross–Pitaevskii equation, the ability of the atom number to vary means a condensate wavefunction with the given chemical potential can always be found.

However, sometimes one desires the ground state of the Gross–Pitaevskii pseudo-Hamiltonian given a certain atom number, or one may be solving the time-independent linear Schrödinger equation and not know the eigenstate energies. In the former case

[27] With the deviation from strict SOR by using non-constant A and $\boldsymbol{b}$ mentioned below, and parallel updates as mentioned above, as well as the fact that systems are not infinite, α in the range of 1.6 to 1.8 is realistic to ensure stability in Gross–Pitaevskii equation simulations.

[28] A subtle consequence of A and $\boldsymbol{b}$ depending on $\boldsymbol{\psi}$ is that we cannot simplify this to $(H(\boldsymbol{\psi}) - \mu)\boldsymbol{\psi} = \mathbf{0}$, as this would admit $\boldsymbol{\psi} = \mathbf{0}$ as a solution. As written, (3.138) also admits $\boldsymbol{\psi} = \mathbf{0}$ as a solution, however, $A^{(k)} = H(\boldsymbol{\psi}^{(k)})$ and $\boldsymbol{b}^{(k)} = \mu\boldsymbol{\psi}^{(k)}$ are treated as constants within a single iteration of SOR, preventing the method from approaching zero as a solution within each step compared to if $\boldsymbol{\psi}$ was present only on the left hand side.

one can to normalise ψ to the desired atom number at each step to impose fixed atom number, but then SOR will not converge to a stationary state/eigenstate unless one has by chance guessed the correct chemical potential. SOR will also not converge for the linear Schrödinger equation[29] unless the energy given corresponds to an actual eigenstate.

[29] Because normalisation for the linear Schrödinger equation is arbitrary, one ought to either normalise ψ at every step, or impose a boundary condition at a single point where ψ is non-zero (simply not updating that point in each SOR STEP), and normalise once at the end in order to prevent ψ growing or shrinking without limit.

A strategy that works is to dynamically update the targeted μ (or E) as SOR proceeds, by computing the expectation value of H using the current best estimate of ψ after each SOR step:

$$\psi_{i\,\mathrm{SOR}}^{(k+1)} = \psi_i^{(k)} + \alpha\left(\frac{\mu^{(k)}\psi_i^{(k)} - \sum_{j\neq i} H_{ij}(\psi^{(k)})\psi_j^{(\mathrm{latest})}}{A_{ii}} - \psi_i^{(k)}\right), \tag{3.140}$$

where

$$\mu^{(k)} = \frac{\psi^{*(k)} \cdot H(\psi^{(k)})\psi^{(k)}}{\psi^{*(k)} \cdot \psi^{(k)}} \tag{3.141}$$

This process of dynamically updating the targeted chemical potential or energy can find stationary states and eigenstates, but not necessarily the ground state. Different initial guesses for ψ may result in convergence to different stationary states/eigenstates, and some attention and guidance from the programmer is required in order to craft initial guesses that result in convergence to the desired states.

3.5.3 Generalisation to excited states via Gram–Schmidt orthonormalisation

[30] Realised independently by me, but not original [86].

Directly diagonalising a Hamiltonian can be costly in a spatial basis. Another approach[30] (for the linear Schrödinger equation) is to find the ground state using one of the above techniques, and then repeat the process with a fresh initial guess, subtracting off the wavefunction's projection onto the already found ground state at every step. This yields the lowest energy state that is orthogonal to the first—i.e. the first excited state. Repeating the process, but subtracting at each step the projection onto *both* eigenstates found so far, then yields the second excited state and so forth. This is simply the Gram–Schmidt process for finding orthonormal vectors, with the additional step of relaxing each vector to the lowest possible energy given the orthogonality requirement—this ensures the eigenstates of the Hamiltonian are produced, rather than a different orthonormal basis. Extra conditions can be imposed on the wavefunction at each relaxation step in order to obtain particular solutions in the case of degenerate eigenstates. For example, a phase winding can be imposed in order to obtain a particular harmonic oscillator state, otherwise this process produces an arbitrary superposition of basis states that have equal energy.

For the case of successive over-relaxation used with the linear Schrödinger equation, as mentioned in Section 3.5.2, one often cannot predict which eigenstate results when the targeted energy is not already known—the ground state is not necessarily the one produced by SOR. Nonetheless, relaxation to *some* eigenstate or other, subject to the state vector being orthonormal to all eigenstates found so far, will produce an additional orthonormal eigenstate each time, just not necessarily in order from lowest to highest energy.

3.6 Fourth order Runge–Kutta in an instantaneous local interaction picture

Consider the differential equation for the components of a state vector $|\psi(t)\rangle$ in a particular basis with basis vectors $|n\rangle$. This might simply be the Schrödinger equation, or

perhaps some sort of nonlinear or other approximate, effective or phenomenological equation not corresponding to pure Hamiltonian evolution. Though they may have additional terms, such equations are generally of the form:

$$\frac{\mathrm{d}}{\mathrm{d}t}\langle n|\psi(t)\rangle = -\frac{\mathrm{i}}{\hbar}\sum_m \langle n|\hat{H}(t)|m\rangle\,\langle m|\psi(t)\rangle\,, \tag{3.142}$$

where $\langle n|\hat{H}(t)|m\rangle$ are the matrix elements in that basis of the Hamiltonian $\hat{H}(t)$, which in general can be time dependent, or even a function of $|\psi(t)\rangle$, depending on the exact type of equation in use. If $\hat{H}(t)$ is almost diagonal in the $|n\rangle$ basis, then the solution to (3.142) is dominated by simple dynamical phase evolution, that is:

$$|\psi(t)\rangle \approx \sum_m \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}E_m t}\,|m\rangle\,, \tag{3.143}$$

where E_m is the energy eigenvalue corresponding to the eigenstate $|m\rangle$.

A transformation into an interaction picture (IP) [59, p. 317] is commonly used to treat this part of the evolution analytically, before solving the remaining dynamics with further analytics or numerics. For numerical methods, integration in the interaction picture allows one to use larger integration timesteps, as one does not need to resolve the fast oscillations around the complex plane due to this dynamical phase.

Choosing an interaction picture typically involves diagonalising the time-independent part of a Hamiltonian, and then proceeding in the basis in which that time-independent part is diagonal. However, often one has a good reason to perform computations in a different basis, in which the time independent part of the Hamiltonian is only approximately diagonal,[31] and transforming between bases may be computationally expensive (involving large matrix-vector multiplications). Furthermore, the Hamiltonian may change sufficiently during the time interval being simulated that the original time-independent Hamiltonian no longer dominates the dynamics at later times. In both these cases it would still be useful to factor out the time-local oscillatory dynamics in whichever basis is being used, in order to avoid taking unreasonably small timesteps.

[31] For example, a spatial basis which allows for partitioning the integration region over multiple nodes on a cluster or cores on a GPU.

To that end, suppose we decompose $\hat{H}(t)$ into diagonal and non-diagonal (in the $|n\rangle$ basis) parts at each moment in time:

$$\hat{H}(t) = \hat{H}_{\text{diag}}(t) + \hat{H}_{\text{nondiag}}(t), \tag{3.144}$$

and use the diagonal part at a specific time $t = t'$ to define a time-independent Hamiltonian:

$$\hat{H}_0^{t'} = \hat{H}_{\text{diag}}(t'), \tag{3.145}$$

which is diagonal in the $|n\rangle$ basis. We can then use then use $\hat{H}_0^{t'}$ to define an interaction picture state vector:

$$|\psi_{\mathrm{I}}^{t'}(t)\rangle = \mathrm{e}^{\frac{\mathrm{i}}{\hbar}(t-t')\hat{H}_0^{t'}}\,|\psi(t)\rangle\,, \tag{3.146}$$

which obeys the differential equation:

$$\frac{\mathrm{d}}{\mathrm{d}t}|\psi_{\mathrm{I}}^{t'}(t)\rangle = \mathrm{e}^{\frac{\mathrm{i}}{\hbar}(t-t')\hat{H}_0^{t'}}\frac{\mathrm{d}}{\mathrm{d}t}|\psi(t)\rangle + \frac{\mathrm{i}}{\hbar}\hat{H}_0^{t'}\,|\psi_{\mathrm{I}}^{t'}(t)\rangle\,, \tag{3.147}$$

where:

$$|\psi(t)\rangle = \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}(t-t')\hat{H}_0^{t'}}\,|\psi_{\mathrm{I}}^{t'}(t)\rangle \tag{3.148}$$

is the original Schrödinger picture (SP) state vector.

This transformation is exact, no approximations or assumptions have been made. If indeed the dynamics of $|\psi(t)\rangle$ in the given basis are dominated by fast oscillating dynamical phases, that is, the diagonals of $\hat{H}_{\text{diag}}(t)$ are much greater than all matrix elements of $\hat{H}_{\text{nondiag}}(t)$ in the $|n\rangle$ basis, then solving the differential equation (3.147) for $|\psi_{\text{I}}^{t'}(t)\rangle$ should allow one to use larger integration timesteps than solving (3.142) directly. And if not, then it should do no harm other than the (small) computational costs of computing some extra scalar exponentials.

Equation (3.146) defines an *instantaneous* interaction picture, in that it depends on the dynamics at a specific time $t = t'$, and can be recomputed repeatedly throughout a computation in order to factor out the fast dynamical phase evolution even as the oscillation rates change over time. It is *local* in that $H_0^{t'}$ is diagonal in the $|n\rangle$ basis, which means that transformations between Schrödinger picture and interaction picture state vectors involves ordinary, elementwise exponentiation of vectors, rather than matrix products. Thus (3.146), (3.147) and (3.148) can be written componentwise as:

$$\langle n|\psi_{\text{I}}^{t'}(t)\rangle = \mathrm{e}^{\mathrm{i}(t-t')\omega_n^{t'}}\,\langle n|\psi(t)\rangle\,, \tag{3.149}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\langle n|\psi_{\text{I}}^{t'}(t)\rangle = \mathrm{e}^{\mathrm{i}(t-t')\omega_n^{t'}}\frac{\mathrm{d}}{\mathrm{d}t}\langle n|\psi(t)\rangle + \mathrm{i}\omega_n^{t'}\,\langle n|\psi_{\text{I}}^{t'}(t)\rangle\,, \tag{3.150}$$

and:

$$\langle n|\psi(t)\rangle = \mathrm{e}^{-\mathrm{i}(t-t')\omega_n^{t'}}\,\langle n|\psi_{\text{I}}^{t'}(t)\rangle\,, \tag{3.151}$$

where we have defined:

$$\omega_n^{t'} = \frac{1}{\hbar}\langle n|\hat{H}_0^{t'}|n\rangle \tag{3.152}$$

This is in contrast to fourth order Runge–Kutta in the interaction picture (RK4IP) [83], in which the interaction picture uses the Fourier basis and thus transforming to and from it involves fast Fourier transforms (FFTs). RK4IP was developed to augment computations in which FFTs were already in use for evaluating spatial derivatives, and so its use of FFTs imposes no additional cost. Nonetheless, an interaction picture based on the kinetic term of the Schrödinger equation (which is the term of the Hamiltonian that RK4IP takes as its time-independent part) may not be useful if that term does not dominate the Hamiltonian, as in the case of a Bose–Einstein condensate in the Thomas–Fermi limit. We compare the two methods below.

3.6.1 Algorithm

The *fourth order Runge–Kutta in an instantaneous local interaction picture* RK4ILIP algorithm is now obtained by using (3.146) to define a new interaction picture at the beginning of each fourth-order Runge–Kutta (RK4) integration timestep. The differential equation and initial conditions supplied to the algorithm are in the ordinary Schrödinger picture, and the interaction picture is used only within a timestep, with the Schrödinger picture state vector returned at the end of each timestep. Thus differential equations need not be modified compared to if ordinary RK4 were being used, and the only modification to calling code required is for a function to compute and return $\omega_n^{t'}$.

Being based on fourth order Runge–Kutta integration, this new method enjoys all the benefits of a workhorse method that is time-proven, and—as evidenced by its extremely widespread use—at a sweet-spot of ease of implementation, accuracy, and required computing power [87].

Below is the resulting algorithm for performing one integration timestep. It takes as input the time t_0 at the start of the timestep, the timestep size Δt, an array $\boldsymbol{\psi}_0$ containing the components $\{\langle n|\psi(t_0)\rangle\}$ of the state vector at time t_0, a function $F(t, \boldsymbol{\psi})$ which takes a time and (the components of) a state vector and returns an array containing the time derivative of each component, and a function $G(t, \boldsymbol{\psi})$ which takes the same inputs and returns an array containing the interaction picture oscillation frequency ω_n for each component at that time.

For example, for the case of the Gross–Pitaevskii equation [88] in the spatial basis $\psi(\boldsymbol{r}, t) = \langle \boldsymbol{r}|\psi(t)\rangle$, these would be:

$$F(t, \psi(\boldsymbol{r}, t)) = -\frac{\mathrm{i}}{\hbar}\Bigg[\underbrace{-\frac{\hbar^2}{2m}\nabla^2}_{\hat{H}_{\text{nondiag}}} + \underbrace{V(\boldsymbol{r}, t) + g|\psi(\boldsymbol{r}, t)|^2}_{\hat{H}_{\text{diag}}}\Bigg]\psi(\boldsymbol{r}, t), \tag{3.153}$$

and

$$G(t, \psi(\boldsymbol{r}, t)) = \frac{1}{\hbar}\Big[\underbrace{V(\boldsymbol{r}, t) + g|\psi(\boldsymbol{r}, t)|^2}_{\hat{H}_{\text{diag}}}\Big]. \tag{3.154}$$

Note that each symbol in bold in the algorithm below denotes an array containing one element for each basis vector $|n\rangle$, subscripts denote the different stages of RK4, and all arithmetic operations between arrays are elementwise[32] The only opportunity for non-elementwise operations to occur is within F, which contains the details (via $\hat{H}_{\text{nondiag}}$) of any couplings between basis states for whatever system of equations is being solved, for example, using FFTs or finite differences to evaluate the Laplacian in (3.153).

[32] For example, the expression $\boldsymbol{a} \leftarrow \mathrm{e}^{-\mathrm{i}\boldsymbol{\omega}\Delta t}\boldsymbol{b}$ indicates that for all n, $a_n \leftarrow \mathrm{e}^{-\mathrm{i}\omega_n\Delta t}b_n$, where a_n denotes the n^{th} element of $\boldsymbol{a}$ etc.

Algorithm 1 RK4ILIP

function RK4ILIP(t_0, Δt, $\boldsymbol{\psi}_0$, F)
  $\boldsymbol{f}_1 \leftarrow F(t_0, \boldsymbol{\psi}_0)$ ▷ First evaluation of Schrödinger picture DE
  $\boldsymbol{\omega} \leftarrow G(t_0, \boldsymbol{\psi}_0)$ ▷ Oscillation frequencies: $\hbar\omega_n = \langle n|\hat{H}_{\text{diag}}(t_0)|n\rangle$
  $\boldsymbol{k}_1 \leftarrow \boldsymbol{f}_1 + \mathrm{i}\boldsymbol{\omega}\boldsymbol{\psi}_0$ ▷ Evaluate (3.150) with $t - t' = 0$
  $\boldsymbol{\phi}_1 \leftarrow \boldsymbol{\psi}_0 + \boldsymbol{k}_1\frac{\Delta t}{2}$ ▷ First RK4 estimate of IP state vector, at $t = t_0 + \frac{\Delta t}{2}$
  $\boldsymbol{\psi}_1 \leftarrow \mathrm{e}^{-\mathrm{i}\boldsymbol{\omega}\frac{\Delta t}{2}}\boldsymbol{\phi}_1$ ▷ Convert first estimate back to SP with (3.151)
  $\boldsymbol{f}_2 \leftarrow F(t_0 + \frac{\Delta t}{2}, \boldsymbol{\psi}_1)$ ▷ Second evaluation of Schrödinger picture DE
  $\boldsymbol{k}_2 \leftarrow \mathrm{e}^{\mathrm{i}\boldsymbol{\omega}\frac{\Delta t}{2}}\boldsymbol{f}_2 + \mathrm{i}\boldsymbol{\omega}\boldsymbol{\phi}_1$ ▷ Evaluate (3.150) with $t - t' = \frac{\Delta t}{2}$
  $\boldsymbol{\phi}_2 \leftarrow \boldsymbol{\psi}_0 + \boldsymbol{k}_2\frac{\Delta t}{2}$ ▷ Second RK4 estimate of IP state vector, at $t = t_0 + \frac{\Delta t}{2}$
  $\boldsymbol{\psi}_2 \leftarrow \mathrm{e}^{-\mathrm{i}\boldsymbol{\omega}\frac{\Delta t}{2}}\boldsymbol{\phi}_2$ ▷ Convert second estimate back to SP with (3.151)
  $\boldsymbol{f}_3 \leftarrow F(t_0 + \frac{\Delta t}{2}, \boldsymbol{\psi}_2)$ ▷ Third evaluation of Schrödinger picture DE
  $\boldsymbol{k}_3 \leftarrow \mathrm{e}^{\mathrm{i}\boldsymbol{\omega}\frac{\Delta t}{2}}\boldsymbol{f}_3 + \mathrm{i}\boldsymbol{\omega}\boldsymbol{\phi}_2$ ▷ Evaluate (3.150) with $t - t' = \frac{\Delta t}{2}$
  $\boldsymbol{\phi}_3 \leftarrow \boldsymbol{\psi}_0 + \boldsymbol{k}_3\Delta t$ ▷ Third RK4 estimate of IP state vector, at $t = t_0 + \Delta t$
  $\boldsymbol{\psi}_3 \leftarrow \mathrm{e}^{-\mathrm{i}\boldsymbol{\omega}\Delta t}\boldsymbol{\phi}_3$ ▷ Convert third estimate back to SP with (3.151)
  $\boldsymbol{f}_4 \leftarrow F(t_0 + \Delta t, \boldsymbol{\psi}_3)$ ▷ Fourth evaluation of Schrödinger picture DE
  $\boldsymbol{k}_4 \leftarrow \mathrm{e}^{\mathrm{i}\boldsymbol{\omega}\Delta t}\boldsymbol{f}_4 + \mathrm{i}\boldsymbol{\omega}\boldsymbol{\phi}_3$ ▷ Evaluate (3.150) with $t - t' = \Delta t$
  $\boldsymbol{\phi}_4 \leftarrow \boldsymbol{\psi}_0 + \frac{\Delta t}{6}(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4)$ ▷ Fourth RK4 estimate, at $t = t_0 + \Delta t$
  $\boldsymbol{\psi}_4 \leftarrow \mathrm{e}^{-\mathrm{i}\boldsymbol{\omega}\Delta t}\boldsymbol{\phi}_4$ ▷ Convert fourth estimate back to SP with (3.151)
  return $\boldsymbol{\psi}_4$ ▷ Return the computed SP state vector at $t = t_0 + \Delta t$
end function

Note on imaginary time evolution

When RK4ILIP is used for imaginary time evolution (ITE) [89], the oscillation frequencies $\boldsymbol{\omega}$ may have a large imaginary part. If the initial guess is different enough from the ground state, then the exponentials in (3.149), (3.150) and (3.151) may result in numerical overflow. To prevent this, one can define a clipped copy of $\boldsymbol{\omega}$,

$$\boldsymbol{\omega}_{\text{clipped}} = \text{Re}(\boldsymbol{\omega}) + \text{i}\begin{cases} -\frac{\log X}{\Delta t} & \text{Im}(\boldsymbol{\omega})\Delta t < -\log X \\ \text{Im}(\boldsymbol{\omega}) & -\log X \leq \text{Im}(\boldsymbol{\omega})\Delta t \leq \log X \\ \frac{\log X}{\Delta t} & \text{Im}(\boldsymbol{\omega})\Delta t > \log X \end{cases}, \tag{3.155}$$

where X is very large but less than the largest representable floating-point number, and use $\boldsymbol{\omega}_{\text{clipped}}$ in the exponents instead. In the below results I used RK4ILIP with ITE to smooth initial states of a Bose–Einstein condensate after a phase printing, and performed clipping with [33] $\log X = 400$.

[33] 400 being about half the largest (base e) exponent representable in double-precision floating point.

This clipped version of $\boldsymbol{\omega}$ should be used in all exponents in the above algorithm, but only in exponents—not in the second term of (3.150). If it is used everywhere then all we have done is chosen a different (less useful) interaction picture, and the algorithm will still overflow. By clipping only the exponents, we produce temporarily 'incorrect' evolution[34], limiting the change in magnitude of each component of the state vector to a factor of X per step (remembering that X is very large). This continues for the few steps that it takes ITE to get all components of the state vector to within a factor of X of the ground state, after which no clipping is necessary and convergence to the ground state proceeds as normal, subject to the ordinary limitations on which timesteps may be used with ITE.

[34] Of no concern since we are using ITE as a relaxation method, and are not interested in intermediate states. Only the final state's correctness concerns us.

3.6.2 Domain of improvement over other methods

For simulations in the spatial basis, RK4ILIP treats the spatially local part of the Hamiltonian analytically to first order, and hence can handle larger potentials than ordinary RK4. However, since a global energy offset can be applied to any potential with no physically meaningful change in the results, ordinary RK4 can also handle large potentials—if they are large due to a large constant term which can simply be subtracted off.

So RK4ILIP is only of benefit in the case of large *spatial variations* in the potential. Only one constant can be subtracted off potentials without changing the physics — subtracting a spatially varying potential would require modification of the differential equation in the manner of a gauge transformation in order to leave the system physically unchanged[35].

[35] Though a numerical solution based on analytically gauging away potentials at each timestep might be equally as fruitful as RK4ILIP.

However that's not quite all: large spatial variation in potentials often comes with the prospect of the potential energy turning into kinetic energy, in which case RK4ILIP is also of little benefit, since in order to resolve the dynamical phase due to the large kinetic term, it would require timesteps just as small as those which ordinary RK4 would need to resolve the dynamical phase evolution from the large potential term.

This leaves RK4ILIP with an advantage only in the case of large spatial variations in the potential that do not lead to equally large kinetic energies. Hence the examples I show in the next section are ones in which the condensate is trapped in a steep potential well—the trap walls are high and hence involve large potentials compared to the interior, but do not lead to large kinetic energies because the condensate is trapped close to its ground state.

The Fourier split-step (FSS) method [90] (see Section 3.4.1) also models dynamical phases due to the potential analytically to low order. As such it is also quite capable of modelling large potentials. However, it requires that all operators be diagonal in either the spatial basis or the Fourier basis [90]. Therefore BECs in rotating frames, due to the

Method	RK4	RK4IP	RK4ILIP	FSS
Error	$\mathcal{O}(\Delta t^4)$	$\mathcal{O}(\Delta t^4)$	$\mathcal{O}(\Delta t^4)$	$\mathcal{O}(\Delta t^2)$
FFTs per step	4	4	4	2
Large ΔV	No	No	Yes	Yes
Large kinetic term	No	Yes	No	Yes
Arbitrary operators	Yes	Yes†	Yes	No
Locally parallelisable	Yes	No	Yes	No
Arbitrary boundary conditions	Yes	No	Yes	No

Table 3.2: Advantages and disadvantages of four timestepping methods for simulating Bose–Einstein condensates. *Large* ΔV refers to whether the method can simulate potentials that vary throughout space by an amount larger than the energy scale $2\pi\hbar/\Delta t$ associated with the simulation timestep Δt. *Arbitrary operators* refers to whether the method permits operators that are not diagonal in either the spatial or Fourier basis, such as angular momentum operators. *Locally parallelisable* means the method can be formulated so as to use only spatially nearby points in evaluating operators, and thus is amenable to parallelisation by splitting the simulation over multiple cores in the spatial basis. † Whilst one can include arbitrary operators within the RK4IP method, only operators diagonal in Fourier space can be analytically treated the way RK4IP treats the kinetic term, and so there is no advantage for these terms over ordinary RK4.

Hamiltonian containing an angular momentum operator, are not amenable to simulation with FSS[36].

This use of FFTs in both the FSS and RK4IP methods necessarily imposes periodic boundary conditions on a simulation, which may not be desirable. By contrast, if different boundary conditions are desired, finite differences instead of FFTs can be used to evaluate spatial derivatives in the RK4 and RK4ILIP methods, so long as a sufficiently high-order finite difference scheme is used so as not to unacceptably impact accuracy.

Along with the ability to impose arbitrary boundary conditions, finite differences require only local data, that is, only points spatially close to the point being considered need be known in order to evaluate derivatives there. This makes finite differences amenable to simulation on cluster computers [91, p. 100], with only a small number of points (depending on the order of the scheme) needing to be exchanged at node-boundaries each step. By contrast, FFT based derivatives require data from the entire spatial region. Whilst this can still be parallelised on a GPU, where all the data is available, it cannot be done on a cluster without large amounts of data transfer between nodes [80]. Thus, RK4 and RK4ILIP, being implementable with finite difference schemes, are considerably friendlier to cluster computing.

Table3.2 summarises the capabilities of the four methods considered in the following results section. RK4ILIP is the only method capable of modelling a large spatial variation in the potential term whilst being locally parallelisable, and supporting arbitrary operators and boundary conditions.

[36] Split-step with more than these two bases is however possible in other schemes such as the finite element discrete variable representation [67]—each operator can be diagonalised and exponentiated locally in each element and applied as a (relatively small) matrix multiplication rather than using FFTs.

3.6.3 Results

Here I compare four numerical methods: Fourier split-step (FSS), fourth order Runge–Kutta in the interaction picture (RK4IP), ordinary fourth order Runge–Kutta (RK4), and my new method — fourth order Runge–Kutta in an instantaneous local interaction picture (RK4ILIP).

The example chosen is a 2D simulation of a turbulent Bose–Einstein condensate, in both a rotating and non-rotating frame. For the non-rotating frame the differential equation simulated was equation (3.153), and for the rotating frame the same equation

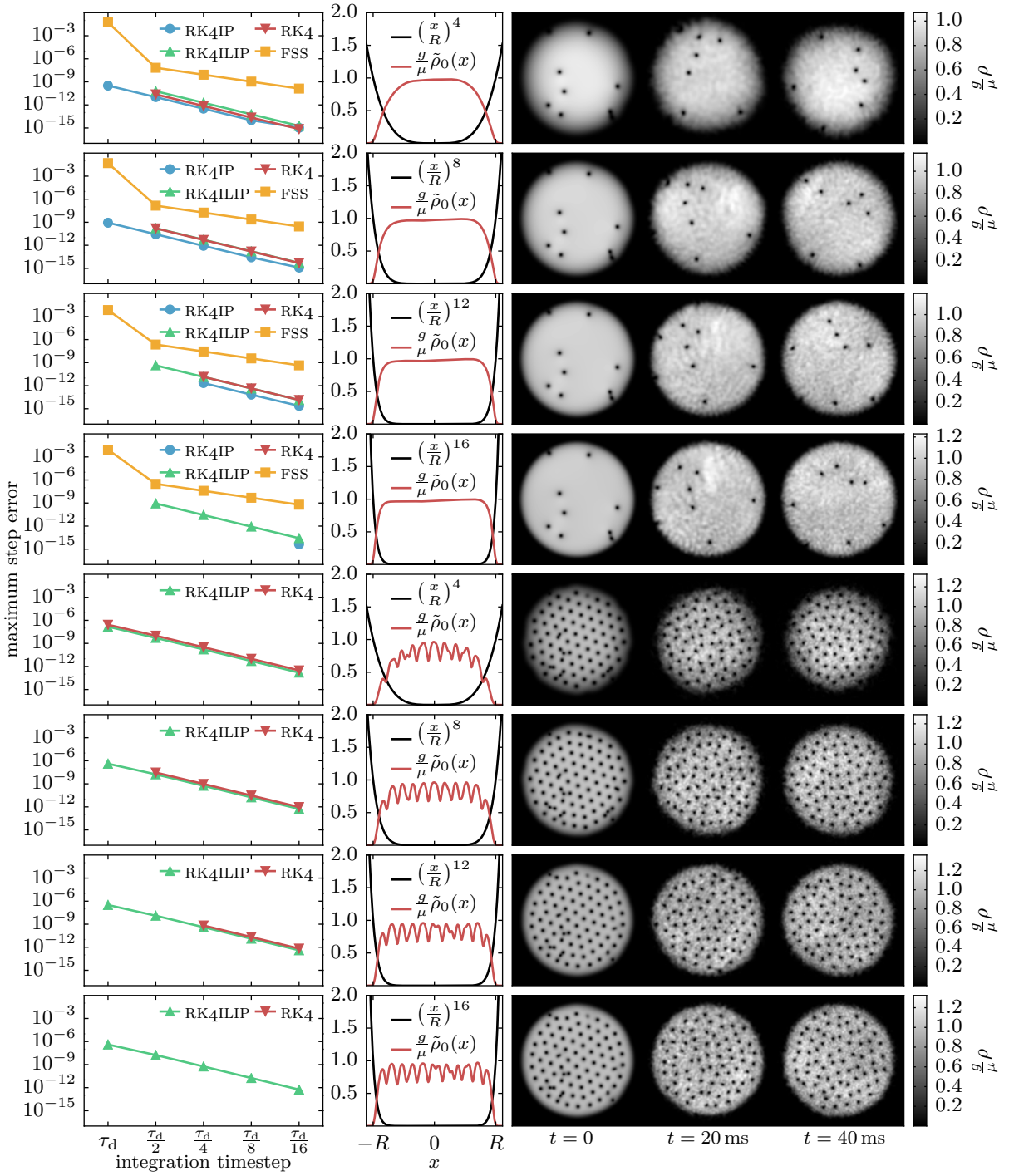

Figure 3.13: Results of simulations to compare RK4ILIP to other timestepping methods. Top four rows: Non-rotating frame simulations with four different radial power-law potentials. Bottom four rows: Rotating frame simulations with same four potentials. Left column: maximum per-step error $\int |\psi - \tilde{\psi}|^2 \, \mathrm{d}\boldsymbol{r} / \int |\tilde{\psi}|^2 \, \mathrm{d}\boldsymbol{r}$ of fourth order Runge–Kutta (RK4), its interaction picture variants (RK4IP and RK4ILIP) and Fourier split-step (FSS) as a function of timestep. Solutions were checked every 100 timesteps against a comparison solution $\tilde{\psi}$ computed using half sized steps for RK4 methods, and quarter sized steps for FSS. Simulations encountering numerical overflow not plotted. Centre column: potential (black) and average density $\tilde{\rho}_0$ of the initial state (red) over a slice of width $R/5$ in the y direction. Right column: Density of solution at initial, intermediate and final times for each configuration simulated (taken from RK4ILIP results). RK4ILIP is the only method usable in rotating frames and not encountering overflow in the steeper traps for the timesteps considered.

was with an additional two terms added to the Hamiltonian:

$$\hat{H}_{\text{rot}} + \hat{H}_{\text{comp}} = -\boldsymbol{\Omega} \cdot \hat{\boldsymbol{L}} + \frac{1}{2}\hbar m^2 \Omega^2 r^2 \tag{3.156}$$

$$= \mathrm{i}\hbar\Omega\left(x\frac{\partial}{\partial y} - y\frac{\partial}{\partial x}\right) + \frac{1}{2}\hbar m^2 \Omega^2 r^2. \tag{3.157}$$

The addition of the first term transforms the original Hamiltonian into a frame rotating at angular frequency Ω in the (x, y) plane, and is equivalent to the Coriolis and centrifugal forces that appear in rotating frames in classical mechanics [92]. The second term is a harmonic potential that exactly compensates for the centrifugal part of this force. In this way the only potential in the rotating frame is the applied trapping potential, and the only effect of the rotating frame is to add the Coriolis force.

Four trapping potentials were used, all radial power laws with different powers. These examples were chosen to demonstrate the specific situation in which RK4ILIP provides a benefit over the other methods for spatial Schrödinger-like equations, as discussed above.

The results of 120 simulation runs are shown in Figure 3.13. Each simulation was of a ^{87}Rb condensate in the $|F = 2, m_F = 2\rangle$ state, in which the two-body s-wave scattering length is $a = 98.98$ Bohr radii [93]. The simulation region was 20 μm in the x and y directions, and the Thomas–Fermi radius of the condensate was $R = 9$ μm. The chemical potential was $\mu = 2\pi\hbar \times 1.91$ kHz, which is equivalent to a maximum Thomas–Fermi density $\rho_{\text{max}} = 2.5 \times 10^{14}\ \text{cm}^{-3}$ and a healing length $\xi = 1.1$ μm. There were 256 simulation grid points in each spatial dimension, which is 14 points per healing length.

Four different potentials were used, all of the form $V(r) = \mu\,(r/R)^{\alpha}$ with $\alpha = 4, 8, 12, 16$. For the rotating frame simulations, the rotation frequency was $\Omega = 2\pi \times 148$ Hz. This is 89% of the effective harmonic trap frequency, defined as the frequency of a harmonic trap that would have the same Thomas–Fermi radius given the same chemical potential.

All ground states were determined using successive over-relaxation (See Section 3.5.2) with sixth-order finite differences for spatial derivatives. For the non-rotating simulations, convergence was reached with $\Delta\mu/\mu < 1 \times 10^{-13}$, with:

$$\Delta\mu = \sqrt{\frac{\langle\psi|(\hat{H} - \mu)^2|\psi\rangle}{\langle\psi|\psi\rangle}}, \tag{3.158}$$

where $\hat{H}$ is the nonlinear Hamiltonian and $\langle \boldsymbol{r}|\psi\rangle$ is the condensate wavefunction, which does not have unit norm. For the rotating frame simulations the ground states converged to $\Delta\mu/\mu \approx 9\times10^{-7}, 2\times10^{-6}, 3\times10^{-6}$ and 2×10^{-6} for $\alpha = 16, 12, 8$, and 4 respectively.

After each ground state was found, it was multiplied by a spatially varying phase factor corresponding to the phase pattern of a number of randomly positioned vortices:

$$\psi_{\text{vortices}}(x, y) = \psi_{\text{groundstate}}(x, y) \prod_{n=1}^{N} \mathrm{e}^{\pm_n \mathrm{i}\,\arctan2(y-y_n, x-x_n)} \tag{3.159}$$

where arctan2 is the two-argument arctan function,[37] $N = 30$, $\pm_n$ is a randomly chosen sign, and (x_n, y_n) are vortex positions randomly drawn from a Gaussian distribution centred on $(0, 0)$ with standard deviation equal to the Thomas–Fermi radius R. The same seed was used for the pseudo-random number generator in each simulation run, and so the vortex positions were identical in each simulation run.

[37] Defined as the principle value of the argument of the complex number $x + iy$: $\arctan2(y, x) = \text{Arg}(x + iy)$.

After vortex phase imprinting, the wavefunctions were evolved in imaginary time [89]. For the non-rotating frame simulations, imaginary time evolution was performed for a time interval equal to the chemical potential timescale $\tau_\mu = 2\pi\hbar/\mu$, and for the rotating

frame simulations, for $\tau_\mu/10$. This was done to smooth out the condensate density in the vicinity of vortices, producing the correct density profile for vortex cores. However, since imaginary time evolution decreases the energy of the state indiscriminately, it also had the side effect of causing vortices of opposite sign to move closer together and annihilate. This decreased the number of vortices, and is the reason the smoothing step in the rotating frame simulations was cut short to $\tau_\mu/10$, as otherwise all vortices had time to annihilate with one of the lattice vortices. A vortex pair in the process of annihilating is visible in Figure 3.13 as a partially filled hole in the initial density profile near the top of the condensate in the $\alpha = 4$, 12, and 16 rotating frame simulations.[38]

[38] The initial states for the four different potentials are not identical, so by chance the corresponding vortex in the $\alpha = 8$ case was not close enough to a lattice vortex to annihilate.

The smoothed, vortex imprinted states were then evolved in time for 40 ms. For each simulation, five different timesteps were used: $\Delta t = \tau_\mathrm{d}, \tau_\mathrm{d}/2, \tau_\mathrm{d}/4, \tau_\mathrm{d}/8, \tau_\mathrm{d}/16$, where $\tau_\mathrm{d} = m\Delta x^2/\pi\hbar \approx 2.68\,\mu\mathrm{s}$ is the dispersion timescale associated with the grid spacing Δx, defined as the time taken to move one gridpoint at the group velocity of the Nyquist mode.

For the non-rotating frame simulations, spatial derivatives for the RK4 and RK4ILIP methods were determined using the Fourier method (Section 3.4.1). This was to ensure a fair comparison with the other two methods, which necessarily use Fourier transforms to perform computations pertaining due to the kinetic term.

For the rotating frame simulations, sixth-order finite differences with zero boundary conditions were used instead for the kinetic terms of the RK4 and RK4ILIP methods, which were the only two methods used for those simulations (due to the other methods being incompatible with the angular momentum operator required for a rotating frame). This choice was fairly arbitrary, but did allow the condensate to be closer to the boundary than is otherwise possible with the periodic boundary conditions imposed by use of the Fourier method for spatial derivatives. This is because the rotating frame Hamiltonian is not periodic in space, and so its discontinuity at the boundary can be a problem if the wavefunction is not sufficiently small there.

As shown in Figure 3.13, all methods tested generally worked well until they didn't work at all, with the per-step error of RK4-based methods being either small and broadly the same as the other RK4-based methods, or growing rapidly to the point of numerical overflow (shown as missing datapoints). The break down of FSS was less dramatic, though it too had a clear jump in its per-step error for larger timesteps. Comparing methods therefore came down to mostly whether or not a simulation experienced numerical overflow during the time interval being simulated.

The main result was that RK4ILIP and FSS remained accurate over the widest range of timesteps and trap steepnesses, with RK4 and RK4IP requiring ever smaller timesteps in order to not overflow as the trap steepness increased.

For the rotating frame simulations, which were only amenable to the RK4 and RK4ILIP methods, the same pattern was observed, with RK4 only working at smaller timesteps as the trap steepness was increased, and ultimately diverging for all timesteps tested at the maximum trap steepness. By contrast, RK4ILIP remained accurate over the entire range of timesteps at the maximum trap steepness.

3.6.4 Discussion

As mentioned, RK4ILIP is mostly useful for continuum quantum mechanics only when there are large spatial differences in the potential, which cannot give rise to equally large kinetic energies[39]. Furthermore, the advantage that RK4ILIP has over other methods with that same property is that it is does not require a particular form of Hamiltonian or a particular method of evaluating spatial derivatives. The former means it is applicable in rotating frames or to situations with unusual Hamiltonians, and the latter means is can be used with finite differences or FEDVR [67] and thus is amenable to parallelisation on a cluster computer.

[39] This is essentially due to such a situation violating the condition we laid out at the beginning of this section — that the simulation basis must be nearly an eigenbasis of the total Hamiltonian.

The ability to model large spatial variations in the potential provides only a narrow domain of increased usefulness over other methods. If a large kinetic energy results from the large potential, then the method requires just as small timesteps as any other. And if the large potential is supposed to approximate an infinite well, then an actual infinite well may be modelled using zero boundary conditions, negating the need for something like RK4ILIP. However, when potential wells are steep, but not infinitely steep, here RK4ILIP provides a benefit. The only other model that can handle these large potentials—Fourier split-step—has the disadvantage that it cannot deal with arbitrary operators such as those arising from a rotating frame, and is not parallelisable with local data. The benefits of parallelisability are obvious, and the above results demonstrate RK4ILIP's advantage at simulating BECs in tight traps and rotating frames.

Note that whilst the *Fourier* split-step method can't handle Hamiltonian terms such as $\hat{\boldsymbol{r}} \cdot \hat{\boldsymbol{p}}$ that are not diagonal in either real space or Fourier space [60, p. 315], a split-step method based on an approximation to the momentum operator as a banded matrix, such as that obtained with finite differences, can. Using the techniques discussed in Section 3.2.4, such a scheme is also parallelisable. The remaining limitations then, when compared to fourth-order Runge–Kutta are the restriction on the types of nonlinearity that can be included, and the complexity of implementation.

For systems with discrete degrees of freedom, RK4ILIP may be useful in the case where an approximate diagonalisation of the Hamiltonian is analytically known, and when the Hamiltonian's eigenvalues vary considerably in time (making a single interaction picture insufficient to factor out dynamical phases throughout the entire simulation). In this situation an analytic transformation into the diagonal basis can be performed at each timestep (or the differential equation analytically re-cast in that basis in the first place), and RK4ILIP can be used to factor out the time-varying dynamical phase evolution at each timestep. An example may be an atom with a magnetic moment in a time-varying magnetic field which varies over orders of magnitude. The transformation into the spin basis in the direction of the magnetic field can be analytically performed, and if the field varies by orders of magnitude, so do the eigenvalues of the Hamiltonian. Although the eigenvalues in this case and other similar cases can be computed analytically too, unless all time dependence of the Hamiltonian is known in advance of the simulation, it would be difficult to incorporate this into a re-casting of the differential equation in a time-dependent interaction picture. RK4ILIP may be useful in these cases to automate this process and evolve the system in the appropriate interaction picture at each timestep.

Chapter 4

Software for experiment control and analysis

Software underlies a huge part of physicists' work, whether experimental or theoretical. On the experimental side, increasingly complex and precise experiments in atomic physics require increasingly sophisticated control of the lasers, magnetic coils, frequency synthesisers, cameras, etc. that interact with the quantum systems being studied. Use of these devices necessitates some kind of interface between the experimentalist and each device, and whilst interfaces of the past were more likely to be knobs and dials on the front of the device, they are increasingly taking the form of software. Software is needed to convert from a smooth ramp of voltages designed to ramp up a magnetic field slowly into a finite list of voltages and times that a device can output with precise timing to make it happen. Software is needed to transmit this data to the device in question, using a communications protocol and data format compatible with the device. Software is required to extract the images from cameras and voltage time series from acquisition devices and store them in computer memory or on disk. And finally, software is required to compute meaningful results from this raw data.

A significant fraction of my PhD was spent developing, maintaining and improving the laboratory control system software suite that has emerged from the Quantum Fluids group at Monash: the *labscript suite*. Originally envisioned as a Python [94] library for generating instructions for programmable hardware via a LabVIEW [95] graphical interface, the software suite grew to encompass most aspects of day-to-day control and analysis in our labs. At present it comprises several separate programs/libraries—depending on how one chooses to draw the borders between them—that control every aspect of a cold atom physics experiment, from setting parameters to analysing results. An overview of the process is shown in Figure 4.1.

The types of experiments the labscript suite addresses are 'shot-based'—ones in which precise timing is required over hardware during some interval while a sequence of instructions is executed (a 'shot'), after which the hardware is idle until the next shot. Many repetitions of similar shots are often performed to build up measurement statistics or investigate the response of a system to a change in one or more parameters. This general method of hardware control and data acquisition is common to many experiments in cold quantum gases and trapped ions [96, 97], quantum computation [98, 99], and quantum simulation [100, 101].

In this chapter, I'll first give a quick overview of each program and what it does. Then I'll outline the design and development approaches we undertook with the labscript suite and comment on the effects these choices have had on the course of the project in subsequent years. Then I'll summarise developments since the publication of our paper on the

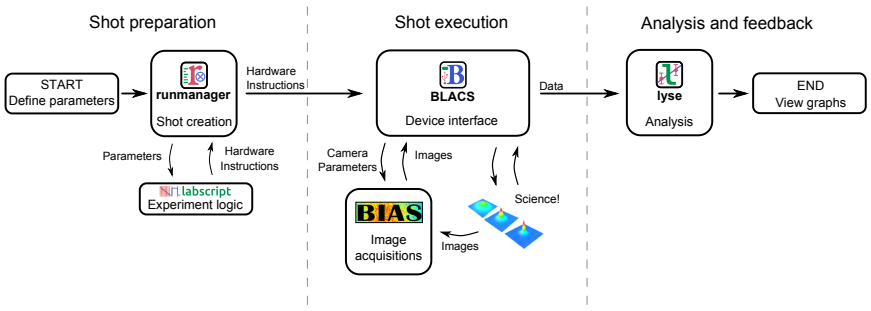

Figure 4.1: The labscript suite comprises a number of libraries and programs allowing one to perform precisely timed experiments, each realisation of which we call a 'shot', using commodity hardware such as devices from SpinCore, NovaTech, National Instruments and others. Experiment logic is described by the user in the form Python code using the `labscript` Python module, which produces from the user's code a set of low-level instructions appropriate for being programmed into the hardware. The program `runmanager` provides a graphical interface for inputting parameters into this experiment logic, and allows this process to be repeated to produce multiple sets of instructions for repeated execution of the shot with different parameters. Not shown in this flowchart is `runviewer`, a graphical program displaying plots of the instructions that have been generated by `labscript`. Once the instructions have been generated, the program `BLACS` (Better Lab Apparatus Control System) is responsible for communicating with the hardware: programming in the generated instructions, beginning the experiment, and saving any acquired data to file. An auxiliary LabVIEW program called `BIAS` (`BEC` Image Acquisition System) is used for communication with cameras in Monash Quantum Fluids group laboratories, though other groups use a number of alternate programs in its place, including a stripped-down derivative called `unBIASed`, as well as several other Python-based 'camera servers'. After `BLACS` is finished with a shot, the data is passed to `lyse`, which executes user-written Python scripts on each shot to analyse the results, and also executes scripts that operate on sets of data over multiple shots, producing dynamically updating plots. In addition to providing a graphical interface for setting parameters, `runmanager` provides a Python library for compiling shots programatically, allowing the flowchart to close into a loop and produce shots with parameters based on analysis results. This can be used to optimise experiment outcomes with respect to a given figure of merit. Figure reused with permission from Starkey et al. [15], © American Institute of Physics 2013.

software, *A scripted control system for autonomous hardware-timed experiments* [15], which is reproduced at the end of this chapter. I will also discuss our development roadmap. Further details on the role of each program in the suite and the design underlying it are available in the paper, and a more thorough presentation of the software, its design principles, comparisons with other laboratory control software, and recent and future developments are available in Philip Starkey's thesis [102].

The labscript suite has been adopted by world-leading research groups at the National Institute of Standards and Technology, the University of Maryland, National Research Laboratories, US Army Research Laboratory, Stanford University, JILA, the University of Rochester, Dartmouth College, Universität Tübingen, Bates College, Universität Basel, and Technische Universität Darmstadt. It continues to grow as a collaborative open-source software project benefiting the experimental physics community.

4.1 The labscript suite

4.1.1 labscript

labscript is a library: that is, it is a set of classes, functions and methods that can be called from user-written code. We call labscript a *compiler*, because the functions, classes and methods within it generate tables of low-level instructions appropriate for programming into devices to execute the experiment described by the user. Thus the user writes a line of code such as `MOT_beams.constant(t=3, 100, 'mW')` and this will add an entry to the table of instructions for whichever digital-to-analogue converter (DAC) is controlling the MOT beams to output 3 mW at $t = 3$ s after the beginning of the experiment. This is a simple example, but has advantages over having a human write the table directly.[1] After one has told the labscript compiler with this line that the MOT beams should have their control voltage[2] set to a particular value, it knows that at all later times the same state should remain, until the user says otherwise. Thus the user doesn't need to also change all future rows of the table: it is enough to declare a change once.

labscript automates much of the tedious, repetitive work of generating those lists of voltages, frequencies, and digital values required to control an apparatus during a shot. This tedium mostly comes from the fact that devices share a common timebase, trigger, or clock. Synchronised output is achieved using a method called *pseudoclocking*, in which timing pulses are produced by a 'pseudoclock' device whenever a state change is required by one or more output devices. These pulses are received by several devices, necessitating that all devices sharing a pseudoclock have an entry in their corresponding tables in order to output the correct value (possibly the same as the previous value they were outputting) at that time, lest they get too far ahead and output a value that was meant for a later time. labscript takes high level descriptions of what voltages etc. are required at different times, puts them on a common timing base and generates the correct tables of values. It also collects any other instructions such as camera exposure durations, or the position a translation stage should move to at the start of the experiment even though it is not capable of moving quickly during the experiment. These instructions are processed by labscript and saved to a file in the Hierarchical Data Format, version 5 [103] (HDF5). HDF5 is a convenient, standardised, cross-platform, and self-documenting format with widespread adoption across many disciplines, and compatibility with a wide range of programming and analysis environments, providing a high degree of interoperability between the data files produced by the labscript suite and other software tools.

[1] Most existing control systems in our field are more-or-less in the form of a large table with each row corresponding to a particular interval during the shot, with the user editing values in the table directly, or typing mathematical expressions in the table describing signal generation as a function of time between one row of the table and the next.

[2] In the example, the user provided a power in units of milliwatts. labscript device objects can include user-provided unit conversion functions, allowing instructions to be given in more physically meaningful units for the common case of an output voltage ultimately controlling a laser power or some other physical quantity via a known conversion function.

4.1.2 runmanager

A thirty second or so experiment shot (a typical duration for a BEC experiment, though ion trapping experiments are often much shorter) is not the only timescale on which experimentalists require automation. Commonly, the same brief experiment is repeated over a range of input parameters spanning some (possibly multi-dimensional) parameter space. In addition, there are many quantities involved in an experiment that do not vary often, but nonetheless need to be managed. runmanager is a program providing a graphical user interface (GUI) for entering and managing such parameters and describing the parameter spaces over which they vary. Users can enter simple numbers or expressions (including expressions for non-numerical variables) into the interface, or lists of numbers that can optionally be considered a description of a dimension of a parameter space. These dimensions may be combined in an outer product resulting in a larger space, or equal length dimensions may be looped over in tandem, if two or more variables are intended to vary together rather than independently.

The GUI of runmanager is shown in Figure 4.2. The user specifies in runmanager's interface which Python file contains their 'experiment script, i.e. a Python script describing

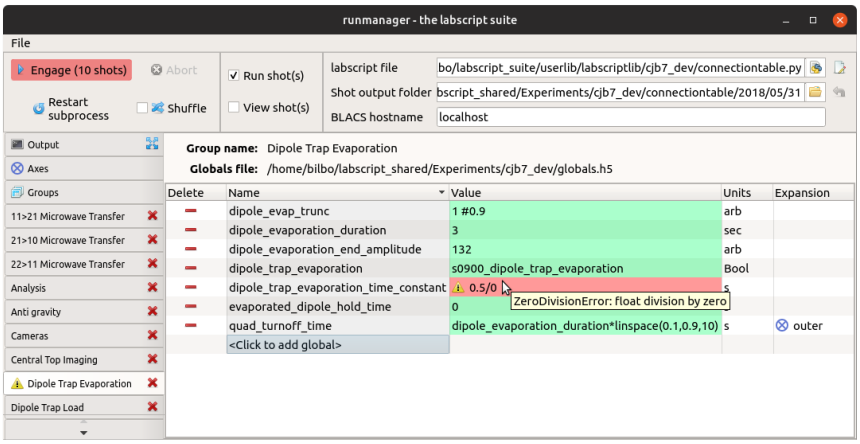

Figure 4.2: The `runmanager` GUI, showing the interface for entering 'globals', so called because they appear to the user's code as global variables. Boolean globals can be turned on and off with a checkbox, and expressions resulting in an error are highlighted in red. The 'expansion' column is where the user specifies whether a global should be considered a list of values to loop over, re-running the experiment each time, and if so if that loop should be combined with other such globals to loop over the resulting product space ('outer') or whether the globals should be looped over together ('zip'[3]). Zipped globals can be grouped together by typing a name in the expansion column to identify which 'zip group' the global belongs to. Globals in the same zip group will loop together, and multiple zip groups will form separate axes of a product space. Globals can be entered as Python expressions, including expressions containing the names of other globals. The evaluated result of an expression, or the error message if it could not be evaluated, is displayed on mouse-over for each global.

the experiment logic using the `labscript` library. When the user clicks the 'engage' button, `runmanager` produces one HDF5 file—each containing a set of parameters—for each point in the parameter space described by those parameters currently active in the `runmanager` interface. For each HDF5 file `runmanager` initialises the `labscript` library such that these values become global variables from the perspective of the user's experiment script, which then runs. For this reason we call the parameters 'globals'. After the user's instructions are processed, the `labscript` library writes the resulting low-level hardware instructions to that same HDF5 file.

We refer to the process of passing the HDF5 file to the user's code and running it as 'compilation', and the resulting HDF5 file containing both globals and hardware instructions a 'compiled shot file'. Compilation occurs in a separate process from the `runmanager` graphical interface, allowing a clean separation between user code and `runmanager`, so that even the most low-level crashes of the user's code cannot crash `runmanager` and only require a restart of its subprocess. This type of separation is a repeated theme in the labscript suite and has been invaluable for making robust programs that can continue to operate in the case of inevitable crashes of user code, or of bugs within labscript suite or third-party code.

[3] The term for this operation—called 'convolution' in computer science—comes from the name of the function in a number of programming languages for converting a tuple of sequences into a sequence of tuples. The Python `zip()` function is commonly used with sequences such as `numpy` arrays for this purpose.

4.1.3 `runviewer`

`runviewer` is a program for viewing the results of `labscript` compilation in the form of graphical plots of the voltages, digital values, frequencies etc. that comprise the hardware

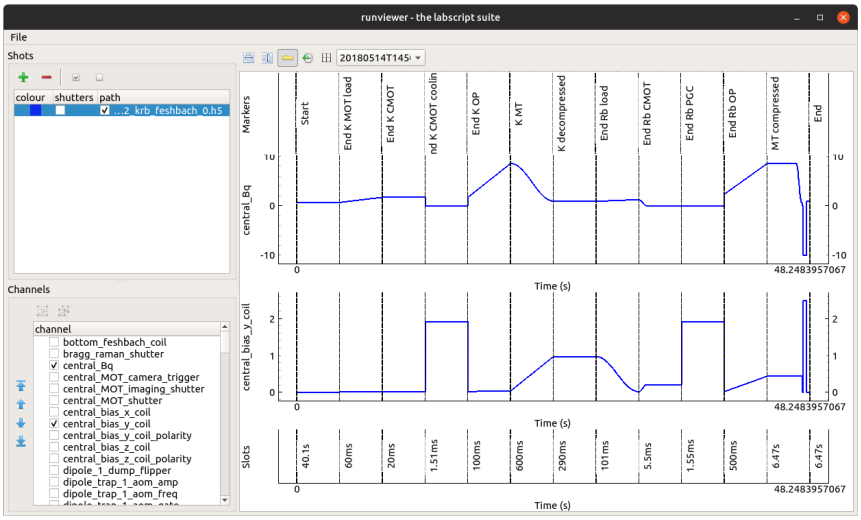

Figure 4.3: The interface of `runviewer`, showing the recently-added 'nonlinear time' feature, in which different intervals of the shot—as declared with `labscript` code—can be meaningfully shown on the same time axis in equal proportion despite being of very different duration.

instructions produced. This is useful for debugging experiment design and timing of instructions, as well as verifying that a newly made `labscript` device class (the 'driver' code for each device that converts `labscript`'s intermediate description of hardware instructions into the actual format required for a given device) is functioning as intended. The GUI of `runviewer` is shown in Figure 4.3, illustrating the display of the output values of a number of devices varying over the course of the shot.

4.1.4 BLACS

BLACS (Better Lab Apparatus Control System) is a graphical program responsible for queueing experiment shots as compiled by `labscript` in tandem with `runmanager`, and executing them one after the other on the hardware. As such, BLACS interacts with a number of Python classes that serve as *drivers* for each device, containing code that uses the required software libraries, hardware drivers or communication protocols to communicate with each device. BLACS executes the code for communicating with each device in a separate process in order to isolate them from each other, so that communication failures, software bugs, or other failures that may occur in the interactions with one device will not stop BLACS from continuing to function in other respects. Errors are presented graphically and each device process may be restarted with the click of a button if something goes wrong, re-initialising communication with the offending device. This is useful both for responding to an unexpected failure, and for debugging when developing a driver for a new device (or new features for an existing device) to be integrated with the labscript suite.

Upon receiving an HDF5 file from `runmanager`, BLACS adds it to the queue of shots to be executed on the hardware. It then executes these shots in order, by programming the instructions stored in the HDF5 file into each device, and then giving the top-level device the command to begin the experiment. Devices are programmed in parallel in their separate processes, saving time.[4] Once the shot is complete, each device process is given the command to write any data acquired to the HDF5 file.

[4] Particularly since many delays in programming the devices are communication delays, during which the process is simply idle.

BLACS can also repeat shots, by copying and then 'cleaning' an HDF5 file after it has already run to produce a new shot file ready to be run on the hardware. It can repeat either all the shots, or just the last one in the queue. This ability to always keep running by repeating the last shot in the queue is crucial for experiments using alkali metal dispensers ('getters') or ultraviolet light-induced atom desorption [104], as these processes must run on an approximately fixed duty cycle to maintain a stable atomic vapour pressure, otherwise experiments need to 'warm up' after being idle to reach a stable pressure.

When processing of the shot queue is paused by the user or when the shot queue is empty, BLACS remains in 'manual mode', in which the devices' outputs may be controlled in real-time by the user. The graphical interface of BLACS (shown in Figure 4.4) presents controls for the outputs of all devices, with each device and output channel labelled with its name as specified in a 'connection table' file containing `labscript` code describing the connection hierarchy of all devices. BLACS automatically generates the graphical interface for each device based a specification of its capabilities by its driver class, resulting in a set of controls for digital, analogue, and frequency generator outputs labelled with their names and using the same unit conversions set in `labscript` code for use in manual mode. Driver code may add to this interface arbitrarily to add a graphical interface for other capabilities of the device. The BLACS queue can be paused, shots reordered or deleted, and the currently running shot can be aborted. The queue can be put in one of two repeat modes, such that either all queued shots, or only the final shot are repeated indefinitely.

4.1.5 lyse

Once a shot has been executed, BLACS optionally passes the shot file on to the analysis program `lyse`. `lyse` is essentially a scheduler for user-provided analysis routines. A list of analysis routines (in the form of Python scripts), called *single-shot routines* are executed in order whenever a new shot is received by `lyse`, with the shot file provided as input to each script. The analysis routines may read raw data from the HDF5 file, or read analysis results saved by previously-run analysis routines, and save their own results. Analysis routines may also produce plots using the `matplotlib` library [105]—`lyse` detects these and reuses the same window for subsequent plots so that repeated runs of the analysis routines result in the plot updating in-place, rather than a proliferation of plot windows. Any other plotting library can be used (for example `pyqtgraph` [106]), though in this case the auto-updating behaviour is not provided automatically by `lyse`.

The shot globals and analysis results for all shots received by `lyse` are maintained in a tabular data structure—a 'dataframe' provided by the `pandas` [107] Python package—which is browsable in the `lyse` interface. This table of data is available to a further list of analysis routines, called *multi-shot routines*. This list of analysis routines is also run in sequence, but only once single-shot analysis has completed on all shots presently loaded into `lyse`. These routines can be analyses of relations between input parameters and analysis results of the shots, in order to say, measure a trend of the number of atoms in a MOT as the magnetic field gradient was varied. Both single-shot and multi-shot routines can be run from within `lyse`, or externally by running Python manually. In this latter case, the shot file on which to run single shot analysis can be provided as a command line argument, and the dataframe analysed by multi-shot routines can be obtained from a running instance of `lyse` over the network. This is no different to what happens when multi-shot routines are run from within `lyse`: they are simply run by `lyse` with no input, and are expected to call a function `lyse.data()` to obtain the dataframe containing multi-shot data. Because of the way this is implemented, one can also open an interactive Python interpreter such as IPython [108] on any computer on the same network, type `import lyse; df = lyse.data(hostname)` (where `hostname` is the network

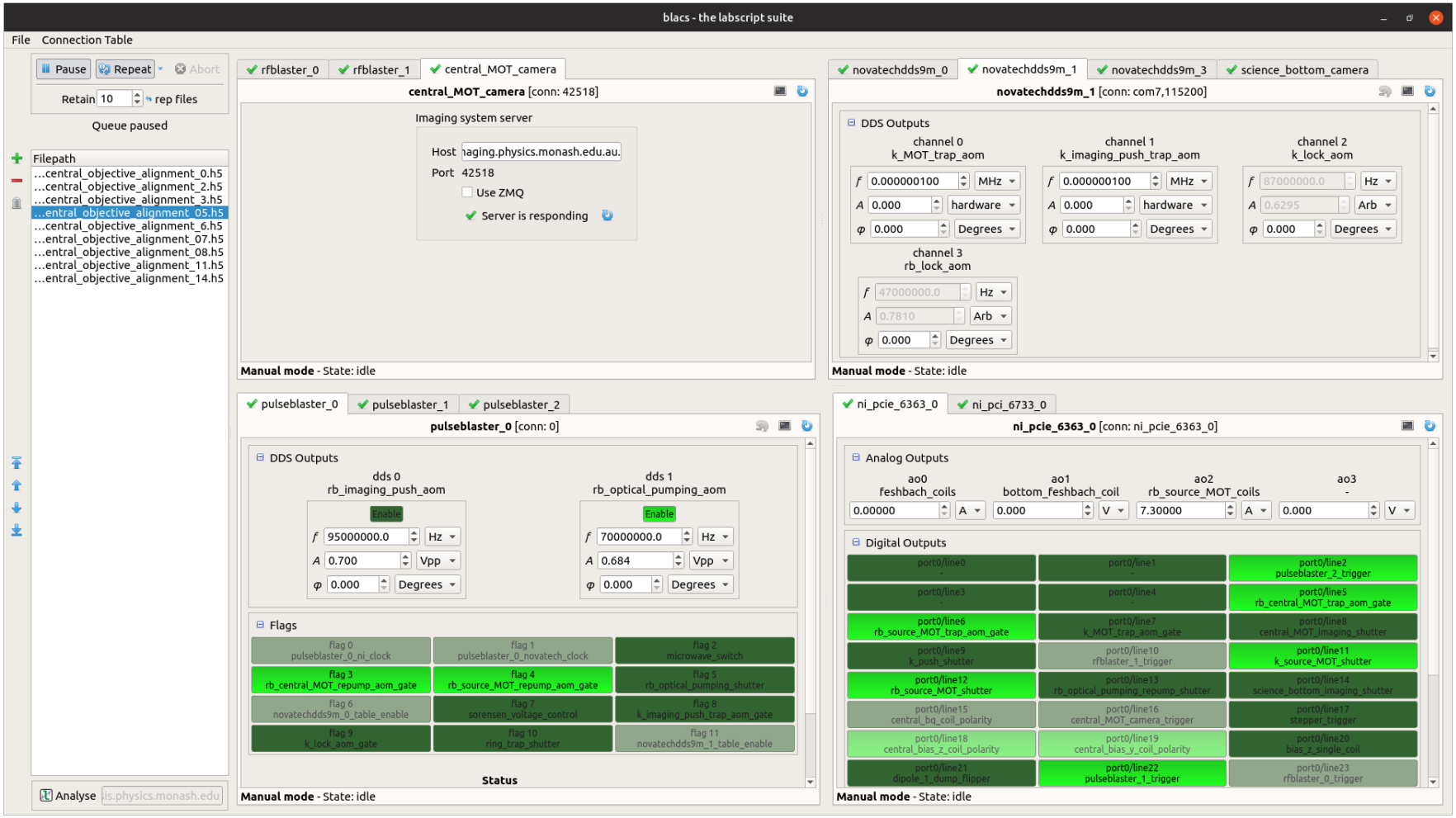

Figure 4.4: The interface of BLACS, showing the experiment queue to the left and manual controls of devices within a tabbed interface to the right. When a hardware-timed experiment is not in progress, outputs may be controlled manually using this interface. Presently the GUI for four devices can be displayed simultaneously, though we plan to allow BLACS tabs to occupy their own windows or even reside on different computers to allow a better use of screen space. Output widgets are labelled with the names of the outputs they control and can be 'locked' to temporarily disallow modifications to their state. Most of the interface for each device is automatically generated from the description of the device and its outputs given in `labscript` code, though the author of a device driver for the labscript suite is free to include any graphical elements in a device tab. This screenshot was taken using the current device configuration of the Monash KRb lab.

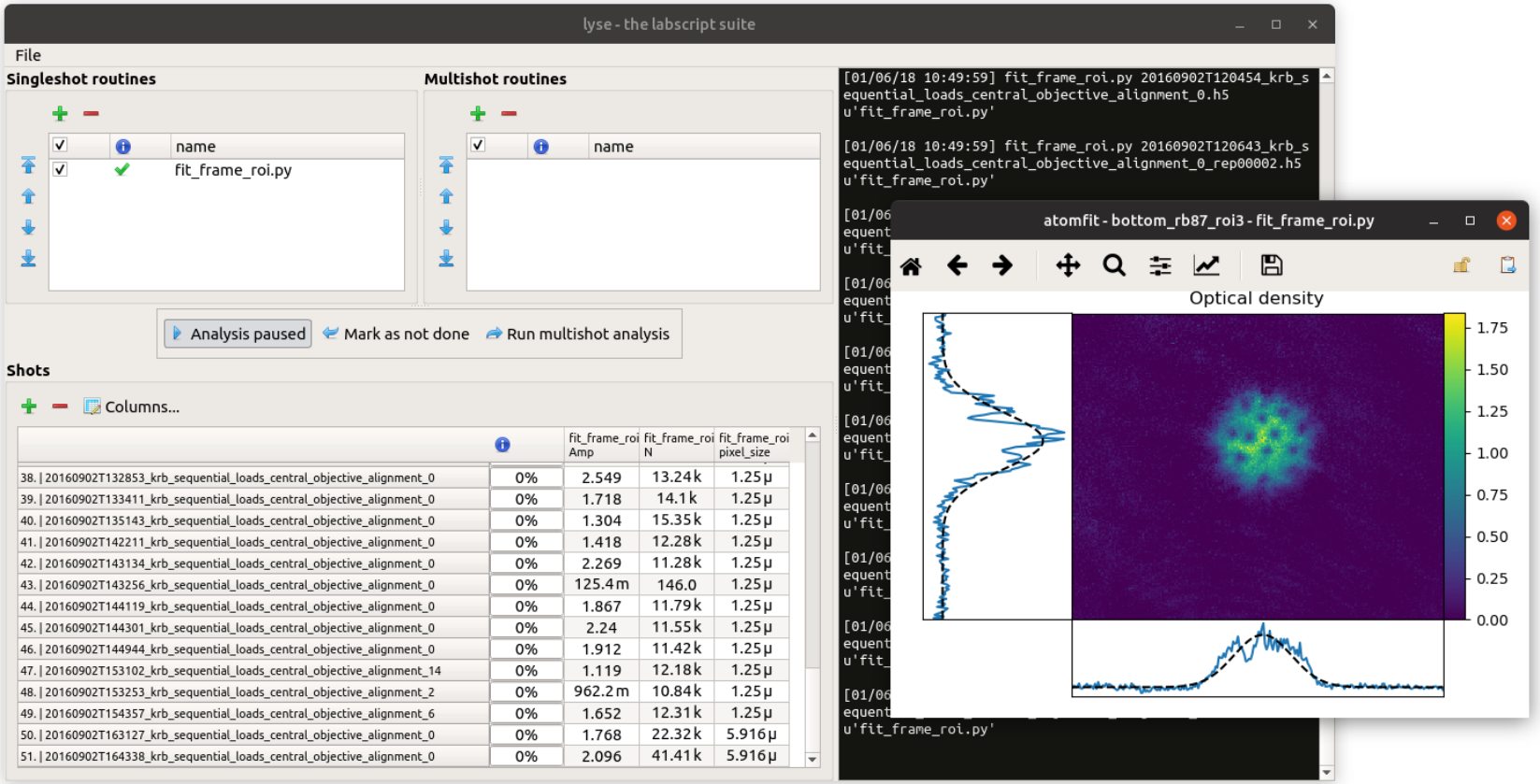

Figure 4.5: The interface of `lyse`. To the top left is the list of single-shot routines, and to the right of it the multi-shot routines presently loaded (empty in this example). The list of shots is below them, showing some of the globals and analysis results of these shots. To the right is the output box where any textual output produced by analysis routines is displayed. Overlaid is a plot produced by a single-shot analysis routine. This screenshot was taken using shot files and an analysis routine script from the Monash KRb lab's two-dimensional turbulence experiment [109]

Figure 4.6: Sage advice from the Monash Spinor BEC lab gremlin. When troubleshooting a hyper-parametrised experiment, one often asks "What's changed (since it last worked)?" This can be answered with a textual diff of the user's experiment script, or by a similar 'diff' of the experiment parameters (globals) built into the `runmanager` GUI, the latter exposing those parameters that have changed, been removed, or added, with respect to any prior shot.

name of the computer running `lyse`) and begin interactively exploring the data using the `pandas` library, one of its great strengths.

4.2 Design philosophy and advantages of approach

4.2.1 It's code

The design of the labscript suite brings the process of composing experiments closer to that of software development, making it amenable to useful development precepts such as version control, bug tracking, and textual 'diffs' highlighting changes between versions of experiment logic. Both the experiment logic and the analysis routines comprise Python code amenable to these practices. This can allow risky changes to experiment logic or analysis code to be explored with the reassurance of being able to roll back to a known working state should the changes not prove fruitful, or for changes to shared experiment or analysis code to be forked into multiple versions using distributed version control and merged back together again to combine improvements from multiple sources. One exception to this 'everything is code' philosophy is the globals themselves, which are stored in non-textual HDF5 files, though we do have a tool for 'diffing' them too, to highlight what has changed between two globals files, or between the current set of values configured in `runmanager` as compared to a particular shot file, which is crucial in the experiment debugging process (Figure 4.6).

As in software development, experimental physics re-uses the same procedures time and time again, and benefits from a way to manage the complexity of turning large parts of functionality on and off, repeating them, or otherwise conditionally modifying their behaviour. In a traditional atomic physics control system, disabling or repeating a part of the experiment logic might involve tedious clicking to remove or duplicate each instruction involved. In Python, this can be a single `if` statement or `for` loop wrapping a function call containing the complexity of the part of the experiment being disabled or repeated. This ability of high level programming languages to manage complexity via encapsulation and code re-use with functions, classes and modules carries well over to experimental physics. Using an existing programming language saves us—as the developers of the control system—from having to re-invent (likely badly) the features of a programming language within our control system. For example, the tedious clicking

required to disable part of an experiment in the aforementioned hypothetical 'traditional' control system could be avoided if the system implemented a feature for conditionals—akin to `if` statements. However, more complex control akin to *nested* `if` statements—or the equivalent of other control flow statements such as `while` loops or recursion—may be required for some applications. To support all the use cases that may arise, one would ultimately be inventing and embedding a complete programming language within such a control system. Use of an existing complete programming language obviates this need.

The use of an existing programming language for experiment control flow only aids the labscript suite in the case of 'compile-time' conditionals and other control statements—those that can be evaluated when the hardware instructions are produced by `labscript`—as opposed to those at 'run time' when the shot is run on the hardware (such as conditionally turning a laser on if a photon is detected on a photo-detector). Such run-time control statements do require special treatment in `labscript`, and `labscript` currently only has one type of functionality like this built-in. This is the ability to pause the shot until a pulse is produced that resumes the 'master' pseudoclock. This allows the common use case of synchronisation with the background 50 or 60 Hz magnetic noise from mains electricity by pausing shot execution until a fixed point in the power line cycle to ensure the magnetic field is close to identical from one shot to the next. Another example is servoing of the MOT load, by loading for a variable amount of time based on a threshold fluorescence such that variations in MOT loading efficiency from one shot to the next do not result in variations in actual atom numbers.

Further run-time control flow tools such as conditional branches would not be too difficult to incorporate into `labscript` in the future, but would require hardware capable of holding multiple alternative sets of instructions in memory and able to switch between them on a digital edge or the state of a digital signal on some input. The SpinCore PulseBlaster—the device used by most users of the labscript suite to produce clocking pulses—supports this, but most output devices labs use with the labscript suite do not. Custom field-programmable gate arrays (FPGAs) implementing this functionality however for digital or analogue output could be integrated with the labscript suite for this purpose, as they have been for other purposes [110].

4.2.2 Modularity and the Unix philosophy

An aspect of the Unix philosophy [111] is that tools should 'do one thing and do it well'. The components of the labscript suite are not as minimal as they could be, but are nonetheless discrete, encapsulating different concerns and communicating with each other over network connections.[5] Thus in principle one can remove a component and replace it with another that plays a different role. For example, `runmanager` has been re-purposed by Philip Starkey [102] to generate parameter space scans for numerical simulations rather than for experiments. `lyse` is in use by Fred Jendrzejewski's group at Universität Heidelberg for on-line analysis of experiment results in the form of HDF5 files produced by a different data acquisition system.

Each of `runmanager`, `BLACS`, and `lyse`, is implemented as a number of sub-components exchanging instructions over network sockets, even though the collection of processes semantically comprise a single application. This architecture facilitates a fairly direct extension (currently in development) in which different parts of the same program can be run on different computers. One can then imagine having `lyse` analysis routines run on a remote computer (perhaps a server with a powerful GPU or many CPU cores for computationally intensive analyses), or having `BLACS` control devices that connected to a remote computer. This latter configuration will allow `BLACS` to simultaneously control devices that cannot be connected to the same computer; for example, those which require different operating systems or have other conflicting software or hardware configurations.[6]. The ability to control devices via additional computers that are located close to

[5] This is in violation of another part of the Unix philosophy that says programs should exchange text streams: our programs mostly exchange messages over network sockets, containing filenames pointing to HDF5 files on a network drive.

[6] A pertinent example is the bandwidth of a computer's USB bus being the limiting factor in the speed at which one lab can run experiments: using two computers to program two USB devices could halve the programming time between shots. This is a limiting factor in the repetition rate of shots in the Spielman group's RbLi lab at the Joint Quantum Institute.

the devices they control will also reduce the need for long signal cables that can contribute to ground loops.

Modularity also allows the user's role in setting globals and commanding the experiment graphically via `runmanager` to be replaced with algorithmic control of globals and shot compilation. This opens the door to closed-loop optimisation in which globals in the next batch of shots are determined by the results of previous analysis routines run on previous shots as part of an optimisation algorithm. In the past, the labscript suite contained a program called `mise`, used for performing optimisation of the experiment. `mise` would use a library of functions provided by `runmanager` (but not the graphical interface) to produce shots based on a genetic algorithm and the results of analysis communicated to it by `lyse` analysis routines. This type of optimisation was powerful but inflexible, and we no longer maintain `mise`. However a similar method is used in the Spinor BEC lab at Monash to integrate the MLOOP machine-learning optimisation library [112] with the labscript suite for experiment optimisation.

The separation of components also aids in development of the labscript suite. Programs can be ported one-at-a-time to use updated versions of libraries, and tested separately, enabling a flexible model of development which has proved invaluable to the open-source development process.

4.2.3 Off-the-shelf hardware

The labscript suite is the software part of an experiment control system, and does not mandate any particular hardware. Whilst drivers for a range of off-the-shelf and in-house devices are developed and maintained by myself and the other labscript suite developers, if one wants to use different hardware, whether off-the-shelf or custom, one can write drivers for it and use it with the labscript suite. This does mean that the software cannot make hard assumptions about the hardware and has to deal with a wider range of possibilities, which is a complicating factor in maintaining the code, but I believe this approach is better compared to designing a limited range of hardware specifically for the labscript suite or vice versa. It is difficult to anticipate what hardware capabilities experimentalists will require for atomic physics experiments in the future, and science is by its nature pushing the envelope such that converging on 'standard' hardware can be at odds with making scientific progress. Therefore it is better to leave the software as agnostic as reasonably possible when it comes to hardware.

The abstraction of devices as Python objects in `labscript` code allows interchangeability of devices with compatible feature sets, minimising the code changes required if a device is replaced with one from the another manufacturer should it break or be superseded in some specification. Such a 'drop-in' replacement may only require one line of user code to change, with all subsequent references in code to the device instance remaining unchanged, and `labscript` thereafter generating instructions as required by the new device. This agnosticism to and abstraction over hardware decreases the friction of making hardware changes to an experiment, allowing the experimenter more flexibility in making such modifications.

Labscript suite developers and users have also made some hardware of their own, and there exists a low-cost pseudoclock and digital output device based on a sub-US$100 microcontroller board (called the PineBlaster and BitBlaster, respectively). However these have proved difficult to maintain in the face of changing software development kits for the microcontrollers, and I suspect the use of FPGAs instead of microcontrollers will offer the same functionality at similar cost, but with greater longevity.[7] Rory Speirs [113] has developed a low-cost FPGA-based pseudoclock (to be released as an open-source project) that may be an attractive alternative to the SpinCore PulseBlaster for use with the labscript suite.

[7] Longevity in the sense of the code continuing to work unmodified with future versions of the device and the software development kit required to program it.

4.2.4 Open-source, popular programming language and data format

The open source nature of labscript suite—as well as that of the technologies used in its implementation—has benefited the project considerably.

The labscript suite itself being open source allows others to modify it to their needs, expanding the range of experiments that it can be used with. If such modifications are applicable to a wide enough range of users, they can be contributed back to the main project. Not least importantly, bugs in the code that have been worked around or fixed by a single end-user can be contributed back to the main project for the benefit of all. This has led to a steady improvement in the usability and stability of the software as usability issues and bugs have been noticed and fixed by people inside and outside the core development team. Sometimes there is disagreement about what features belong in the labscript suite, or more often, over how they should be implemented. The open-source nature of the project allows end users to continue to use an implementation of functionality that there may not be agreement about including in the main project, either permanently or until another implementation is available. This is preferable to simply being at the mercy of the core developers as to what features they will or will not implement (whether due to differing opinions or to time constraints).[8]

[8] Although a commercial model has the potential to work here too, in which users pay for features to be implemented. There is also the 'best of both worlds' approach of bug and feature 'bounties', where users add a cash reward for implementing certain features or resolving bugs, adopted by some open-source software projects.

The use of third-party open-source technologies in the labscript suite has also been advantageous, as when the projects we rely on do not fully satisfy our needs, we can modify them accordingly. Furthermore, these modifications, if agreeable to the developers of said libraries, can be included in the 'upstream' project to remove the requirement that we maintain the patches ourselves. To this end I have had changes accepted into the `numpy` [114], `pandas` [107], `pyzmq` [115] and `h5py` [116] projects which improve their behaviour for the labscript suite's purposes.

Python is a popular programming language with a gentle learning curve, allowing new users with little programming experience to compose experiment logic quickly. Many students and other physicists already use Python for data analysis and other tasks, and knowledge of the language is increasingly widespread among physicists. This decreases the barrier to entry for modifying the labscript suite or contributing to its development.

The use of the standardised HDF5 format, and of the popular `zeromq` [117] messaging protocol for communication between components allows interoperability with a wide range of other programming languages and technologies, such that software written in other languages can interact with running labscript suite programs and read the data files produced by them. The camera interface program `BIAS` is an example of this—it is written in LabVIEW and yet reads and writes data to the same shot files as the rest of the labscript suite using the HDF5 library, and communicates with components of the suite over the network using the `zeromq` library.

4.2.5 Collateral benefits

Developing and maintaining the labscript suite has involved achieving several intermediate, instrumental goals in order to advance the ultimate goals of the project. Some of these solutions have been packaged as separate software projects, or are available from within the labscript suite, and may be used for other purposes.

For example, the modular nature of the suite (Section 4.2.2) and the multi-threaded and multi-process implementation of its components necessitates that components communicate with each other by message passing (largely in line with the 'actor model' of concurrent programming[9]). As such the codebase contains reusable pieces for launching processes and initiating message passing with them, of redirecting output of subprocesses to some visible location rather than a terminal, or of starting and stopping 'servers' and creating 'clients' to either pipe data continuously, or make discrete requests that necessitate a response. Furthermore, these multiple threads and processes often require access to

[9] Though I had no formal education in the actor model before writing code in this style—I attribute my fondness for it to the computer game *SpaceChem* [118], in which the player constructs chemical 'reactors' comprising multiple threads of execution exchanging atoms and molecules.

the same files (the HDF5 files containing the data for each experiment run), and access to these files needs to be serialised to prevent data corruption, including the case of multiple computers attempting to access the file over the network. To address these problems I developed `zprocess` [119], an open-source Python package officially separate from the labscript suite but very much developed to meet its needs for management of and communication/synchronisation between multiple processes. This project is used not just for the labscript suite, but for other laboratory automation tasks. It is also used in the software implementation of an undergraduate experiment in the School of Physics and Astronomy at Monash University, allowing students to remotely control the experiment and collect data over the internet [120].

Multi-threaded graphical programs can present development problems as the software library for the graphical interface generally must be accessed only from the one thread. This is the case for the Qt toolkit [121] as used in the labscript suite, and so interacting with the graphical interface from multiple threads necessitates message passing to request that an operation be performed in the 'main' thread. We have similarly encapsulated our solutions to this—as well as several other problems repeatedly encountered in using the Qt toolkit, such as an icon set, automatic loading of graphical layouts from external files, and others—into the `qtutils` project [122], which is used for other small graphical utilities in our group, separately from the labscript suite. This library has also been used in an undergraduate teaching context at Monash University to improve infrastructure for digital logbooks used by undergraduate students.

We have also developed a number of debugging and profiling tools that are useful for debugging numerical simulations, or other code unrelated to the labscript suite.

4.3 Recent and future developments

Since the publication of our paper in 2013 [15], the number of groups using our software has increased considerably, from the two BEC laboratories at Monash, to a modest number of groups around the world.

During my PhD I was invited to visit the Joint Quantum Institute in the group of Ian Spielman, with the aim of improving the labscript suite to make it more usable, and easier for others to install. The primary goal was to port the graphical programs from the GTK toolkit [123]—used to produce the graphical interfaces of the labscript suite—to the Qt toolkit [121], and to write a program to automate the installation process, which was previously somewhat tedious. These goals were achieved, and installing the labscript suite is now a matter of installing an appropriate Python environment, and then downloading and running our installation script.

The following subsections detail this and a number of other developments since the publication of our paper, as well as planned and in-progress improvements to the suite.

4.3.1 Port to Qt

The port to Qt [121] was a crucial development. As discussed in the publication reproduced at the end of this chapter, we initially chose the GTK toolkit for its cross-platform compatibility and good Python bindings [124]. This proved to be the wrong decision, as the GTK project has rapidly developed and dropped support for older versions. The newer versions were difficult to install or deploy on operating systems other than Linux, and the changes were significant enough to impose a considerable ongoing development cost to keeping code up to date with them. Although slated as a cross-platform toolkit, the GTK project is developed by and primarily serves the needs of the GNOME project [125], and thus its development is driven by those needs. There is little incentive for Windows- or Mac-specific bugs to be fixed, or for the installation process to be improved. Although

it's an open-source project such that people other than the core developers could contribute fixes to these issues, the Linux-centric nature of the project impedes these fixes from being accepted for inclusion. Using the (also open-source) Qt toolkit is the chosen solution for many cross-platform graphical software projects, and so we chose to adopt this toolkit in 2013, with most programs ported in 2014. The experience of both maintaining and installing the labscript suite is much improved as a result. Being the standard cross-platform GUI toolkit, Qt is already available in the standard Anaconda Python distribution [126]–the preferred Python distribution among scientists at the present time. With our `qtutils` package, an icon set is available, obviating the need to install a separate icon pack as was previously the case. Most importantly, the Qt software project exists to serve the needs of graphical programs generally, and is used by many major cross-platform projects. Combined with the fact that Qt is open source, this is insurance against future breaking changes in Qt or against the Qt Company going out of business—open source projects can be forked and maintained by the community, and the more popular they are the more likely this is to occur if they take an unpopular turn or are abandoned by their present maintainers. We therefore have confidence that the Qt project's direction is aligned with the needs of the labscript suite, and that we can continue to rely on it in a way we could not with the GTK project for our project's longevity, without introducing unnecessary technical debt.

4.3.2 Python 3

The labscript suite was initially written in version 2.7 of the Python programming language, even though version 3 of Python had been released several years prior. The Python community has in this time been in a decade-long transition from one version to the next due to some non-trivial differences between the two versions. I believe we made the right decision to initially use Python version 2.7, as many of the technologies the labscript suite relies on did not have Python 3 compatibility for some time. However, the point of inflection in the adoption curve for Python 3 has occurred in the last two years or so, and now is the right time for Python 3 adoption. With the help of third party contributors (primarily Jan Werkmann), the entirety of the labscript suite has now been ported to run on both Python 2 and Python 3.

We do not expect such an extended issue as this to occur again: the Python core developers consider this transition to be a one-off, and future porting efforts of labscript-suite components will likely be no more work than usually required to keep up with minor changes between language versions.

4.3.3 More devices, more features, general polish

There are more devices compatible with the labscript suite, and more models and features of existing devices are now supported. The programs are easier to use, and many cases where obscure errors were thrown have been replaced with friendlier error messages explaining the situation in human-readable terms. Following is an incomplete list of minor to modest usability improvements:

- `BLACS` has a plugin to delete shots that are repeated versions of previous shots. This prevents unnecessary consumption of disk space when these shots are running only to keep an experiment 'warm' (initially implemented by Ian Spielman, and re-implemented by me as a plugin for `BLACS`).
- `lyse` more gracefully handles shot files that have been deleted off disk: declining to run single-shot analysis on them, but keeping their data in the dataframe available for multi-shot analysis until they are deleted from the `lyse` interface. This aids in, for example, diagnosing a day-long drift in some performance characteristic of

the experiment even though most shot files are deleted due to the aforementioned desire to save disk space (implemented by me).

- `lyse` is more performant, only updating those values of the dataframe that have changed, and minimising the number of times it opens a HDF5 file. This improves performance for very large numbers of very short duration shots, as is common in ion trapping (implemented by Jan Werkmann with changes by me).

- `BLACS` has had some bugs resolved that unnecessarily introduced delays on the order of 0.5 s between running shots. These delays were not very noticeable for the cold atom experiments of order 15–30 s, but again are important for the ion trappers (implemented by me and Philip Starkey).

- `BLACS` tabs have a separate optional terminal output for each device subprocess, allowing simpler debugging and development of devices (implemented by me).

- More flexible camera interface. There are now a number of camera 'servers' in use by various groups playing the role that `BIAS` plays at Monash, including a fork of `BIAS` named `unBIASed` by Ian Spielman (changes made to facilitate communication with Python camera servers implemented by me)

- `labscript` can accept arbitrary function ramps using user-supplied functions, not limited to the built-in list of functional forms. This has always been possible with some effort, but now has a friendlier programming interface (implemented by me).

- There is a unified interface for saving and retrieving configuration settings of devices in `labscript` to HDF5 files, including JavaScript Object Notation serialisation for complex data types that do not coincide with a HDF5 datatype. This replaces a number of ad-hoc serialisation methods previously in use to store configuration settings of devices (specification designed by Ian Spielman and core developers, implementation by Ian Spielman and me).

- There is a unified `labscript` device driver for National Instruments DAQmx devices, removing the code duplication and complexity of maintaining multiple device classes for this range of devices. This class exists in the fork of the `labscript_devices` repository in use by the Spielman group at the Joint Quantum Institute, but will be merged into the mainline codebase soon—it is already in use by groups who are not otherwise using the Spielman fork of the code, and so has undergone some testing and bugfixes outside of the hardware in the Spielman group. In time the model-specific code will likely be removed in favour of the unified interface (implemented by Ian Spielman).

- `labscript` has functionality to mark certain points of time of the shot with a named marker, visible in `runviewer` to visibly delineate different stages of the experiment (implemented by Jan Werkmann).

- `runviewer` has a 'nonlinear time' mode, in which the time axis uses a different scale for the different stages of the experiment as described by the markers, allowing short and long timescales to be visible on the same plots in. (implemented by Jan Werkmann with changes by Shaun Johnstone).

- `labscript` has the ability to mark digital outputs as 'inverted', such that digital low represents a device being on, semantically speaking. The buttons for these outputs are represented with different colours in the `BLACS` interface to avoid confusion (implemented by Jan Werkmann).

- Gated clocks: devices with vastly different memory capabilities can receive clocking signals from the same clocking device such as a PulseBlaster, but on different digital channels such that the clock ticks intended for one device are not received by other devices. There is still a single pseudoclock, but its outputs are 'gated'—whilst the clock is ticking for one device it is not ticking for another (implemented by Philip Starkey).

- `lyse` plots can be copied to the clipboard with a button click, reducing the number of steps to include plots in a digital log book (implemented by me).

- `runmanager`, `lyse`, and `runviewer` have the ability to save and load configuration settings, such that the same sets of globals files in the case of `runmanager` or the same sets of analysis routines in the case of `lyse` and the same view settings in the case of `runviewer` can be loaded at start-up of each application (implemented by me for `runmanager` and Jan Werkmann for `lyse` and `runviewer`).

- `runmanager` allows finer control over parameter spaces, including randomising the order of a parameter space scan on a per-axis basis, control over the nesting order in which the axes are looped over, and a graphical representation of this looping (implemented by Philip Starkey).

4.3.4 Optimisation

The program `mise`, mentioned in the paper, has been deprecated. Nothing has replaced it, though due to the modularity of the labscript suite, optimisation is still possible through use of the `runmanager` library directly. This is a testament to the flexible design of the labscript suite, but could nonetheless be improved.

I plan to improve this functionality in the future, and one of the near-term development goals is to add a 'remote' application programming interface (API) for `runmanager`. This will enable a program to control a running instance of `runmanager` to set the values of globals and initiate shot compilation and submission to `BLACS`. This will be a much simpler interface as well as being compatible with just-in-time compilation, discussed below.

4.3.5 Just-in-time compilation

One feedback mechanism not previously anticipated—more accurately described as *feed-forward* in this context—is the need to modify one or more parameters in the very next shot to be run, whilst performing some parameter space scan or repetition. For example, in the Spielman group's atom chip lab, an environmental magnetic field drift on a several hour timescale is corrected for by performing an error measurement each shot, to be fed-forward to the next shot as a change in the applied field bias. Other than this, the experiment is not performing any optimisation or feedback. In the atom chip lab, this functionality is implemented by having `BLACS` use the existing `runmanager` API (not the proposed 'remote' API) to re-compile the shot files just before running them, to be sure to include the updated magnetic field bias estimate.

This is an example of functionality being implemented by users to serve an immediate need. However I would like to move this 'just-in-time' compilation into `runmanager`, so that the shot is compiled only once rather than being recompiled. Compilation by `BLACS` is unappealing since it may be on a different computer with different versions of the code being compiled, leading to the possibility of subtle errors, and more generally violates the design philosophy of separation of concerns that has served us well thus far.

For this reason—as well the applicability to optimisation mentioned above—I plan to implement both a remote API for `runmanager` as well as a 'compilation queue' containing

shots yet to be compiled, whose variables can still be changed (possibly remotely) up until the moment BLACS requests a new shot,[10] triggering compilation to occur.

[10] Either because its queue is empty, or nearly empty—by requesting a shot before the queue is fully empty one can prevent the experiment being idle during compilation, at the expense of the feed-forward changes taking effect only some number of shots in the future.

4.3.6 Fixed shot repetition interval

Since BLACS takes a usually small—but variable—amount of time to program the hardware in between shots, this contributes to a variation in the average proportion of time an atom dispenser is receiving current or ultraviolet light-induced desorption is active, leading to vapour pressure variations in experiments that operate in this manner. In the Spielman fork of the BLACS repository, there is functionality—to be merged or reimplemented in the main codebase—for setting a fixed overall duration for the shot execution plus programming time, such that after programming devices, BLACS waits some additional amount of time in order that shots run at precisely equal intervals. This can be used not only to compensate for variations in programming time, but also for variations caused by changes of parameters that affect the shot duration, or by variable-length waits while the experiment is paused, such as servoing a MOT load as mentioned in Section 4.2.1.

4.3.7 Remote device control

A work in progress is to allow devices to be connected to any computer, not just the one that BLACS is running on. Some devices—such as those communicating over Ethernet—are inherently remote controllable, but others can only be programmed from the computer to which they are directly connected. In addition to the benefits of remote control outlined in Section 4.2.2, a distributed graphical interface would also allow one to make better use of computer screen real estate, with interfaces for different device controlled by BLACS presented on separate computers, reducing the clutter in a single BLACS interface with a large number of devices. One reason for making a separate camera control program in the form of BIAS was to be able to view images immediately at all times without having to ensure the correct tab in the BLACS interface is active. Being able to display these tabs as separate windows on different screens and computers will obviate this need and allow cameras to be once again treated the same as other devices.[11]

In discussion with developers and users, we have designed a specification for how the desired layout of devices on a network will be described by users in labscript code, and we have a partial implementation. Most of the work toward this has been in the zprocess package, which I have been adapting to meet these needs, including the use of encryption to ensure that the ability to start processes on remote computers is secure. This feature in zprocess is nearly complete, after which we will modify BLACS to make use of zprocess to launch remote processes for communicating with hardware and displaying their graphical interfaces. These two features are planned to be separate, such that BLACS, the graphical interface for a specific device, and the device itself can be on the same computer, or two, or three, for maximum flexibility in the location of graphical interfaces and hardware within the lab.

[11] This was not the only reason BIAS was made as a separate system. Another reason was the availability of software libraries for interacting with certain cameras, these were available for LabVIEW at the time, but not Python. Python wrappers, pyvisa [127] and pynivision [128] for the National Instruments VISA and NI-Vision libraries have since become available, removing the need for LabVIEW to be used with cameras or other devices requiring these interfaces.

4.4 labscript version 3

The labscript compiler itself is the oldest part of our codebase, and has changed significantly since its initial incarnation. We have become more skilled programmers in the 7 years since it was first written, and some design decisions have proved to be limiting. For example, many calculations performed by labscript during compilation are *destructive*—that is, new data replaces old data as processing proceeds. Specifically, timing transformations in labscript discard the original, untransformed timing data. This makes it difficult to debug where timing problems have occurred, and makes the code fragile to the introduction of bugs in which timing transformations are accidentally taken

into account twice or not at all, as opposed to exactly once as required. Furthermore, timing calculations are performed mostly using floating point arithmetic, necessitating that all comparisons be performed with some tolerance or rounding. This is error-prone and unnatural given that the hardware devices generally have a quantised duration of each instruction. Finally, when an exception occurs in `labscript` indicative of the user requesting something not possible (such as two instructions closer together in time than the hardware is capable of), `labscript` does not identify where in the user's code the instruction originated, making it difficult to provide the user with information that helps them resolve the issue.

To address these three concerns, I have been restructuring the core instruction and timing processing of `labscript`, such that: all processing is non-destructive, the points in the user's code where instructions are created are noted for later use in error messages, and all timing calculations are performed using integer arithmetic after quantising timing details as early as possible, according to the time resolution of the pseudoclock of each device. This project, called `labscript_core`, will eventually replace the part of `labscript` responsible for timing calculations and instruction handling, improving the code from the perspective of users and developers. If possible I will keep the timing computations separate from the other higher level parts of `labscript` (globals, device properties, HDF5 files) so that it can be independently tested and verified (ideally in the context of an automated test suite) so that we can be confident in the output it produces, and that regressions have not been introduced when changes are made.

4.5 Other future developments

An immediate concern is to unify the mainline labscript suite codebase—that maintained by myself and the other core developers—with the Spielman group's fork of the code. This will involve merging (or re-implementing in the mainline codebase) the remaining features present in the Spielman fork as discussed in the previous sections, as well as others not mentioned here.

There are a number of third party contributions (mostly authored by Jan Werkmann) awaiting approval by a core maintainer such as myself or Philip Starkey. Some of these are:

- Analogue input widgets for `BLACS`, which display a numerical value for the voltage of each analogue input, or interactively updating plot of each voltage trace over time (i.e. a virtual oscilloscope).
- Plugin tabs for `BLACS` to allow plugins for `BLACS` to insert tabs into its graphical interface, allowing plugins to have rich interfaces of their own, separate from the main interface of `BLACS`.

Other changes proposed but not implemented include:

- 'Analysis globals': like the globals set in `runmanager`, but set in `lyse` instead. Presently, users are 'abusing' globals set in `runmanager` in order to configure how analysis will run. For example, there are globals being set by users that tell analysis code which pair of variables to plot against each other. As this may change after a shot has run, an interface where 'analysis globals' can be set and used by analysis routines would be preferable.

Finally, the entire project would benefit from more and better documentation. Documentation exists,[12] though it could be far more thorough. More importantly, the development process has not ensured that documentation is kept up to date with changes to the code. Instead, new information is often hidden in code comments or in commit messages.

[12] Primarily written by Philip Starkey

This as a flaw in the development process, rather than a lack of effort or consideration. The Python community has a solution to this problem, which is to use libraries that turn appropriately formatted text embedded within the code, called 'docstrings', into proper documentation. This creates incentives to write and update these docstrings to a higher standard, knowing that they will be immediately visible in official documentation. This expectation can be enforced during code reviews that occur when pull requests are made for changes to be included in the mainline code repository. Such a change of process would help maintain and improve documentation continuously over time, and is better in my opinion than occasional bursts of effort and re-writes of the documentation.

An immediately available strategy to bring about better documentation is to put the process in place to automatically render and publish the existing docstrings, despite their present inadequacy, and begin to enforce the policy of updating the documentation whenever the code changes. This way over time the documentation will improve, and being code rather than a PDF or Microsoft Word document, will be amenable to pull requests and bug reports in the same manner as the rest of the code, which—as we have experienced—lead to inexorable improvement via an open process that accepts external fixes from others.

There is also a substantial quantity of best-practices and lore known to users of the labscript suite regarding hardware as well as software, including many tips and tricks that are specific to certain setups. Whilst the labscript suite has a mailing list in which much of this information is exchanged, it would benefit the project to have a user-editable wiki, to decrease the barrier-to-entry for users to share domain-specific knowledge outside the context of an email thread, even though it may not suit the official project documentation.

4.6 Project history and attribution

Here I acknowledge the contributions of authors to the different components of the labscript suite, and provide a history of major developments. This is not comprehensive, and many improvements have been provided by others; nonetheless the majority of the design and programming effort behind the labscript suite is due to the authors responsible for the initial creation and major developments of each subproject as outlined below. A full history of the codebase is publicly available on our on-line repositories [119, 122, 129] and can be used to credit any particular change or piece of code to the appropriate author.[13] Plots of present and historical authorship by number of lines of code are shown in Figure 4.7.

[13] With the exception of generic usernames occasionally being used from lab computers such that authorship was not recorded.

The initial idea of having object-oriented Python code compiled to low-level instructions was inspired by a similar implementation by Scott Owen and David Hall [22] in which experiment logic is described in C++ and programmed into the hardware using a LabVIEW program with a state-machine architecture.

Toward this goal I developed an initial implementation of the `labscript` compiler in Python in February 2011. Throughout the following year or so, the architecture of the software suite, its components and how they would interact was developed by members of the Monash Quantum Fluids group including Lincoln Turner, Russell Anderson, myself, Philip Starkey, Shaun Johnsone, Martijn Jasperse, and others.

The initial implementation of programming `labscript`-generated instructions into hardware was written in LabVIEW by to Russell Anderson in early 2011, followed by the initial version of `runmanager` written by me in August 2011 and the initial version of `runviewer`, also by me, in September 2011. Early in development I moved `runviewer` to use the Qt toolkit in order to take advantage of the `pyqtgraph` [106] plotting library, which was necessary for acceptable performance of `runviewer`.

Philip Starkey wrote the initial version of `BLACS` in October 2011, superseding the LabVIEW control system. In November 2011 Philip Starkey and I re-architected `BLACS`

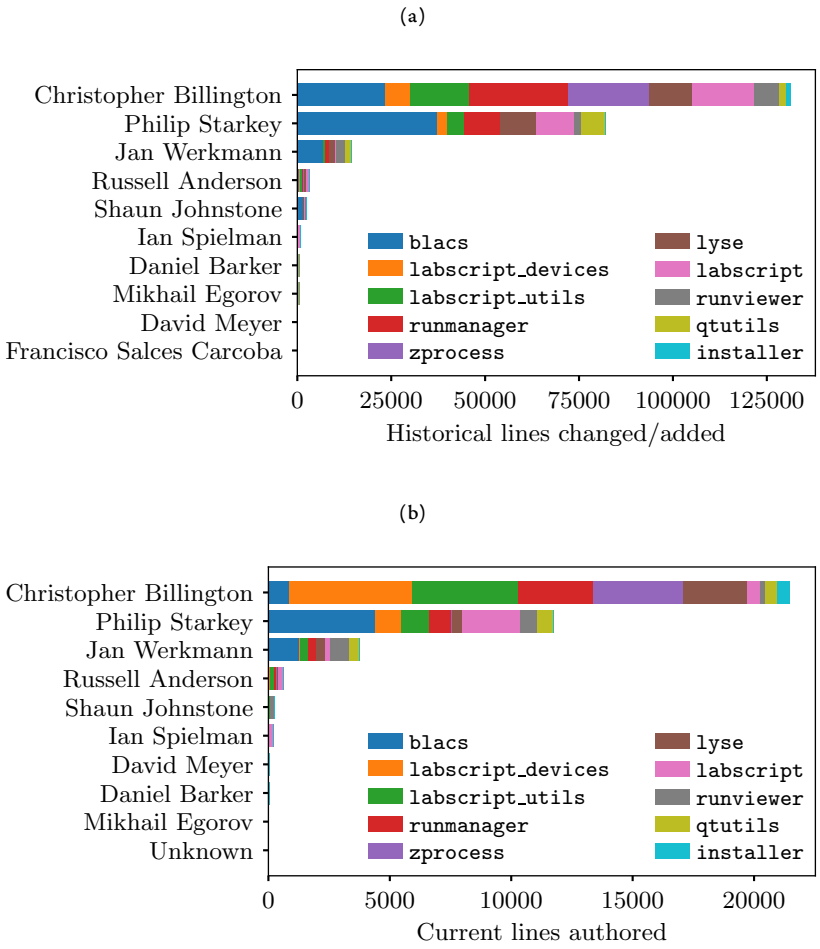

Figure 4.7: Authorship by lines of code of each component of the labscript suite, for the top ten authors. (a) Authorship of line changes over the entire project history. (b) Authorship of lines currently present in the most recent revision of each component. The line counts for these plots were obtained using the mercurial 'churn' extension and 'blame' command to analyse all Python, Cython and C source files in our code repositories. This excludes documentation (most of which is in Microsoft Word format) as well as layout files for the graphical interfaces of the programs. Although these results are broadly representative of authors' contributions to the components of the labscript suite, they should be taken with a grain of salt. Authorship is difficult to ascertain programatically as refactoring counts the same as original authorship even though usually less credit (though not zero) should be assigned to someone refactoring code than the author writing it originally. As such my apparent contribution to `labscript_devices`—the repository containing device drivers—is exaggerated due to me being the one to migrate this code from another repository. Similarly Jan Werkmann's contributions to `BLACS` are exaggerated due to recent renaming of one particular large source file. We both have made original contributions to both repositories as well, but these are not distinguishable from refactoring in these plots.

as a multi-process model and improved the state machine architecture it uses.

In February 2012 I wrote `lyse`. In April 2012, Philip Starkey re-wrote a substantial portion of `runmanager`, making its user interface suitable for larger number of globals and improving other functionality.

In January 2013, Philip Starkey ported `BLACS` to use the Qt toolkit, and in May 2014, he expanded the capabilities of the `labscript` compiler by implementing the 'gated clocks' feature, allowing more devices to be controlled by a single pseudoclock device such as a SpinCore PulseBlaster. Between August and November 2014 while visiting Ian Spielman's group, I ported `runmanager` and `lyse` to the Qt toolkit, wrote the labscript suite installer script, and adopted the practice of tracking dependencies of the different components on each other using semantic versioning [130].

Throughout this time device driver code was contributed, various features implemented, and bugs fixed by many including Russell Anderson, Shaun Johnstone, Martijn Jasperse and Ian Spielman. Documentation was written primarily by Philip Starkey. `BIAS` was written and maintained by Martijn Jasperse from late 2011 onward.

When we started writing applications in Qt, I wrote the initial code for multithreading in Qt that became the `qtutils` [122] package, with most of the later functionality in the package added by Philip Starkey.

The `zprocess` package, encapsulating the network communication and multiprocessing code common to several components within the labscript suite, was written and is maintained by me (as always with some bugfixes and features contributed by others).

4.7 Conclusion

The labscript suite is an increasingly mature software project for control of hardware-timed experiments. It has an increasing number of users and contributors, and has evolved to keep up with changing software environments. It is a living project accepting changes from non-core developers, and is free for anybody to use under a permissive license. Due to the modular design and open development process, the labscript suite has thus far avoided some of the pitfalls that befall many laboratory control systems and software, such as relegation to legacy software or hardware environments due to a lack of development process capable of keeping them up to date, or fixes to bugs remaining unresolved for most because a fix applied in one place has not been distributed to others, or because the source code is only available to few. The labscript suite is hardware-agnostic, ensuring its use is not restricted to officially sanctioned or in-house hardware. Finally, it is written in a popular programming language with an abundance of on-line resources and a vibrant community behind it, within both scientific and software engineering circles. Equally important as the implementation details of the codebase itself, these decisions have resulted in a thriving project beneficial to experimental physics research.

4.8 Reproduced publication: A scripted control system for autonomous hardware-timed experiments

See over page for a reproduction of our 2013 paper, *A scripted control system for autonomous hardware-timed experiments*,

A scripted control system for autonomous hardware-timed experiments

P. T. Starkey,[a),b)] C. J. Billington,[a)] S. P. Johnstone, M. Jasperse, K. Helmerson, L. D. Turner, and R. P. Anderson
School of Physics, Monash University, Victoria 3800, Australia

(Received 11 April 2013; accepted 17 July 2013; published online 8 August 2013)

We present the *labscript suite*, an open-source experiment control system for automating shot-based experiments and their analysis. Experiments are composed as Python code, which is used to produce low-level hardware instructions. They are queued up and executed on the hardware in real time, synchronized by a pseudoclock. Experiment parameters are manipulated graphically, and analysis routines are run as new data are acquired. With this system, we can easily automate exploration of parameter spaces, including closed-loop optimization. © *2013 AIP Publishing LLC*. [http://dx.doi.org/10.1063/1.4817213]

I. INTRODUCTION

Modern experiments in quantum science demand flexible, autonomous control of heterogeneous hardware. Many such experiments are *shot*-based: a single experiment shot comprises analog, digital, and radiofrequency (rf) outputs operating under precise timing, as well as synchronized camera exposures and voltage measurements. Bose–Einstein condensation (BEC) experiments,[1] for example, require a timing resolution down to a few hundred nanoseconds, and may last for up to a minute. Output must, therefore, be hardware timed, requiring devices be programmed with instructions in advance of an experiment shot. Most measurements of interest require numerous shots, to build up statistics, or to observe the response of the system to varying parameters. Such repetition is common to experiments employing cold quantum gases or trapped ions for precision metrology,[2] quantum computation,[3] and quantum simulation.[4]

Individual shots are typically complex, requiring the coordination of many devices. This coordination is the role of a *control system*. A good control system should automate the programming of devices based on a high-level description of the experiment logic.[5] It should handle the repetition of shots and automated variation of experiment parameters, the increasingly complex demands of which cannot be rapidly, robustly, and continuously met by human operators. It should automate analysis, leading to the prospect of closed-loop control: the results of analysis influencing subsequent experiment shots. Applications of such closed-loop control include autonomous algorithmic optimization of parameters, and automatic recalibration in response to environmental drifts.

Most existing control systems take one of the two approaches for providing a human interface to programming hardware. One is text-based, in which experiments are written using a general purpose programming language.[6] In the other, experiments are instead described graphically using a custom user interface.[7–11] The text-based approach natively offers the advantages of a programming language, particularly control-flow tools such as conditional statements, loops, and functions. Its disadvantage is that frequently varied settings and parameters may be hidden in hundreds of lines of code. Conversely, the graphical-user-interface (GUI) approach makes experiment parameters more accessible to the user, but features providing for complex experiment logic must be anticipated and implemented specifically.[10,11]

The two approaches need not be mutually exclusive: by separating experiment parameters from experiment logic, parameters can be manipulated graphically and logic textually.[12,13] We contend that by using a high-level programming language with appropriate hardware abstraction, text-based control can be more comprehensible to a newcomer than an equivalent graphical representation of hardware instructions.

We present the *labscript suite* which utilizes a hybrid text-and-GUI approach for control and builds on previous work by addressing the need for autonomous control, analysis, and optimization. Hardware control is abstracted, providing an identical software interface to devices of a common type. Graphical interfaces are dynamically generated based on the current hardware set in use. Analysis is an integral part of the control system, with user-written analysis routines run automatically on new data. Finally, analysis results can modify subsequent experiment shots, closing the feedback loop on analysis and control.

II. AN OVERVIEW OF THE LABSCRIPT SUITE

The labscript suite comprises several programs, each performing one main function; the flow of data between programs is shown in Fig. 1. Each experiment shot is associated with a single file: each program writes to and reads from this file as required before passing it on to the next program. Programs may be run on separate computers, communicating over the network using the ZeroMQ messaging library,[14] exchanging references to the experiment file.

We use the Hierarchical Data Format (HDF version 5)[15] which provides cross-platform storage of large scientific datasets. Exploiting the extensibility of HDF, each file

a)P. T. Starkey and C. J. Billington contributed equally to this work.
b)Author to whom correspondence should be addressed. Electronic mail: philip.starkey@monash.edu

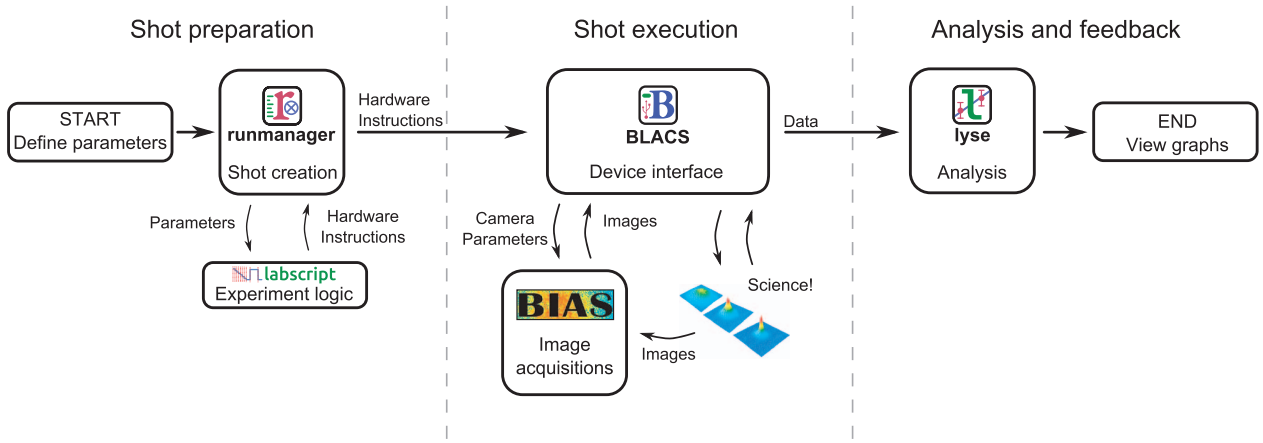

FIG. 1. Each experiment shot comprises three stages: preparation, execution, and analysis. Arrows indicate how the HDF file for an experiment shot passes between software components of the labscript suite. Only the shot execution stage is coupled to hardware timing, allowing new shots to be created and queued while others are running. Similarly, analysis can be performed on executed shots at any time.

is a complete description of the experiment shot. The HDF file begins life containing only experiment parameters. As it is passed between components of the labscript suite, the file grows to contain the hardware instructions, acquired data, and analysis results. Metadata is also stored including user-written scripts and version control information. This maintains a comprehensive record of the experiment shot for post-hoc analysis, reproducibility, and publication preparation.

Attempts to standardize laboratory device programming have largely failed, with only a minority of devices conforming to standards such as SCPI (Standard Commands for Programmable Instruments).[16] This calls for abstraction to shield the user from low-level interaction. We have created a software library for Python,[17,18] `labscript` (Sec. IV), which provides a common interface for commanding output and measurements from devices. The user writes the experiment logic in Python, and `labscript` generates the required hardware instructions, including a clocking signal for timing (Sec. III).

The labscript suite separates experiment logic (written in Python) from experiment parameters, which are manipulated in a GUI. The GUI, `runmanager` (Sec. V), creates the HDF file for the experiment shot and stores the parameters within. If a parameter is a list of values, rather than a single value, `runmanager` creates an HDF file (a prospective shot) for each value. If lists are entered for more than one parameter, `runmanager` creates a file for each point in the resulting parameter space.

For each shot, `labscript` inserts the parameters from the HDF file into the experiment logic, compiles hardware instructions for each device, and writes them to the same file. `runmanager` sends the compiled HDF files to `BLACS` (Sec. VI) which places them in a queue. `BLACS` interfaces with hardware devices either directly, or via secondary control programs such as `BIAS` (Sec. VII). `BLACS` programs the hardware and triggers the experiment shot to begin. The experiment then proceeds under hardware-timed control.

Once the experiment shot has finished, acquired data such as voltage time-series and images are added to the HDF file. `BLACS` then passes the file to a dedicated analysis system, `lyse` (Sec. VIII). `lyse` coordinates the execution of analysis routines, which are Python scripts written by the user. These scripts may analyze individual shots or a sequence of shots as a whole. This facilitates autonomous analysis of results from parameter space scans, as experiment shots are completed.

The `labscript` software library can be applied to automatically generate shots based on the results of analysis. We have used this to implement a closed-loop optimization system, `mise` (Sec. IX).

III. PSEUDOCLOCK

A typical BEC experiment requires precise timing over a large range of time scales.[1] There are periods during which magnetic fields or laser intensities, for example, may change with sub-microsecond resolution. Conversely, there are periods during which no devices change their output for several seconds, e.g., loading a magneto-optical trap (MOT). To ensure accurate output during the rapid changes, hardware devices must be preloaded with a set of instructions that can be stepped through by a clock once the experiment begins. Stepping through instructions at a constant rate requires repetitive instructions during the more inactive periods. As many devices only support a limited number of instructions, a constant-rate clock limits the maximum sample rate. A common solution[8,9,11,12] is a variable frequency master clock, or *pseudoclock*, which steps through instructions only when a clocked device needs to update an output (see Fig. 2). This removes the need for redundant instructions.

All devices sharing a pseudoclock must have an instruction when any one of their outputs changes value. This can lead to redundant instructions if only some of the devices are changing at a given time. The instruction limitations of one device may then limit another, e.g., some devices hold only a few thousand instructions in their internal memory, whereas others are limited only by the RAM of the host computer refilling their buffers. To solve this problem, we employ

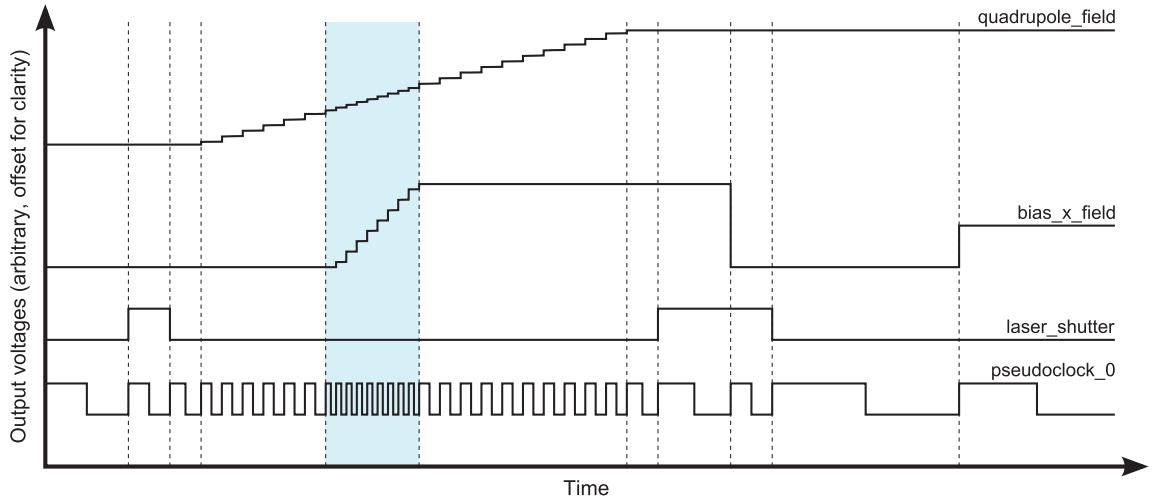

FIG. 2. An example of digital and analog voltage outputs generated by the labscript code in Fig. 3. The pseudoclock (lower trace) ticks when a digital output must change, or at the requested sample rate for time-varying analog outputs (upper two traces). Dashed vertical lines indicate a change in the pseudoclock frequency. When multiple analog outputs are varying at the same time (shaded region), the pseudoclock ticks at the highest of their sampling rates.

multiple pseudoclocks, assigning devices of similar memory limitations to the same clock. At the beginning of a shot, the software starts one pseudoclock (the *master clock*), which then triggers other clocks.

To be a useful pseudoclock, a device must be able to deterministically generate arbitrary digital signals, be hardware triggerable, and hold enough instructions for the required experiment. We currently use two pseudoclocks: the Spin-Core PulseBlaster DDS-II-300-AWG, a commercial device based on a field-programmable gate array (FPGA); and the PineBlaster, a device developed in house based on a microcontroller. Both devices are externally referenced to a stable 10 MHz source.

The PineBlaster is a low-cost device using commodity hardware, based on the Arduino-like Digilent ChipKIT Max32 microcontroller prototyping board.[19] The board is flashed with a program that accepts clock instructions over universal serial bus (USB) and executes them with deterministic timing. It is capable of clocking at up to 10 MHz (100 ns between rising edges) with a resolution of 25 ns. The PineBlaster needs one instruction for each change in clock rate (see Fig. 2) and supports up to 15 000 instructions.

`labscript` provides support for adding new pseudoclocks. It uses an intermediate format for storing timing instructions; implementing a new pseudoclock entails translating them into the required format for the hardware.

Some experiments require the time between instructions to be determined *during* a shot. This can be achieved by pausing the pseudoclocks until some condition is met. A common example[11–13] is waiting for a sufficient level of fluorescence from a loading MOT. Both the PulseBlaster and the PineBlaster support *wait instructions*, which pause output until resumed by a trigger. These instructions, when used in tandem with devices such as voltage comparators, can command the experiment to wait for events of interest.

IV. THE LABSCRIPT LIBRARY

We have created a Python software library, `labscript`, for defining experiment logic. `labscript` provides *hardware abstraction*, a common interface to control heterogeneous hardware. For example, the `DigitalOut` class provides `go_high(t)` and `go_low(t)` functions to set the state of a digital output at time `t`. The user calls these functions without regard to the underlying device, its method of programming, or the state of other digital outputs connected to the same device. Based on an experiment script containing such function calls, `labscript` automatically generates instructions for output and measurement devices as well as pseudoclocks. The automatic generation of pseudoclock instructions saves the user from dividing overlapping function ramps into segments (Fig. 2), or manually interpolating output values when a new time point is created on another channel.

An experiment script consists of two parts: a *connection table* (Fig. 3(a)), and code defining the logic of the experiment (Fig. 3(b)). The connection table provides a complete description of devices that are required for the experiment and how they are connected. `labscript` creates a set of Python objects based on the connection table, each with associated functions for commanding output or measurement from devices. The logic of the experiment is then defined by calling these functions with parameters such as time and output value.

As the experiment script is executable Python code, the user has full access to standard Python control flow tools, as well as standard and third party Python libraries. Using a high level language such as Python spares the user from low-level tasks such as memory management.[5] User-created functions can be stored in modules and imported into other experiment scripts. This allows complex experiments to be constructed from simple components, while maintaining comprehensibility, resulting in a gentler learning curve for new students. For example, one might define a `make_BEC()` function which

(a)
```
# Device definitions
PulseBlaster(name='pseudoclock_0', board_number=0)
NI_PCIe_6363(name='ni_card_0',  parent_device=pseudoclock_0, clock_type='fast clock',
             MAX_name='ni_pcie_6363_0', clock_terminal='/ni_pcie_6363_0/PFI0')

# Channel definitions
Shutter  (name='laser_shutter',    parent_device=ni_card_0, connection='port0/line13')
AnalogOut(name='quadrupole_field', parent_device=ni_card_0, connection='ao0')
AnalogOut(name='bias_x_field',     parent_device=ni_card_0, connection='ao1')
```

(b)
```
# Experiment logic
start()
t = 0

# first laser pulse at t = 1 second
t += 1;    laser_shutter.open(t)
t += 0.5;  laser_shutter.close(t)
t += 0.4;

t += quadrupole_field.ramp(t, duration=5, initial=0, final=3, samplerate=4)  # samplerate in Hz
# start ramping the bias field 3 seconds before the quadrupole ramp ends
bias_x_field.ramp(t-3, duration=1, initial=0, final=2.731, samplerate=8)
# t is now 6.9s, the end of the quadrupole field ramp

# second laser pulse
t += 0.4;  laser_shutter.open(t)
t += 1;    bias_x_field.constant(t, value=0.0)
t += 0.5;  laser_shutter.close(t)
t += 2

# hold bias field at bias_x_final_field for 2 seconds before finishing shot
bias_x_field.constant(t, value=bias_x_final_field)
t += 2
stop(t)
```

FIG. 3. An example `labscript` file. The connection table (a) defines a pseudoclock and a multifunction DAC object and configures three output channels. This is followed by the experiment logic (b) which commands output from these channels by name at times specified by the variable `t`. The experiment logic refers to the parameter `bias_x_final_field` which is set in `runmanager` (Sec. V).

contains the logic to form a Bose–Einstein condensate. While students might not fully understand the experiment logic to create a BEC, they can focus on subsequent experiment logic after a BEC is made. We have found that text based experiment scripts benefit not just from code re-use but also version control, bug tracking, and comparison of incremental changes (diffs).

When the experiment script is run and a timing sequence created, the `labscript` functions take into account hardware limitations and provide error messages if these are exceeded. If no errors are found, the hardware instruction set for all devices in the connection table is written to the HDF file.

While a text-based definition of experiment logic gives a broad overview of the timing sequence, it is not ideal for visualizing the device outputs to ensure the experiment logic is as intended. The hardware instructions generated by running experiment scripts are difficult to interpret (indeed, `labscript` was created to mitigate this very problem). Our program (`runviewer`) produces plots (similar to Fig. 2) of the hardware instructions generated by `labscript`, allowing quick diagnosis of the timing sequence before reaching for the oscilloscope.

V. SETTING PARAMETERS—RUNMANAGER

Repeating experiments while varying parameters is a fundamental part of the scientific method. Anyone who has performed a quantum science experiment will be familiar with tweaking parameters to find a resonance, calibrating a measurement, or acquiring a large amount of scientific data prior to publication. The logic of the experiment does not change every time a parameter is adjusted, and it is cumbersome to edit numbers in a text file for each modification.

To ameliorate this, `labscript` experiments can take a series of parameters as input. The names and values of these parameters are defined in the graphical interface of `runmanager` (Fig. 4). The values can be any valid Python expression (such as `0.74`, `1E-3`, `sin(pi/2)`, or `True`) and can refer to each other. We call these parameters *globals* because they are available as global variables in experiment scripts, where they are simply referred to by name. For example, these globals might be used to specify the duration of a π-pulse, the delay between releasing atoms from a trap and imaging them, or the field strengths of bias magnetic coils. This provides a clean separation between code, which defines the nature of the experiment (such as creating a BEC with a vortex or performing a matter-wave mixing experiment), and parameters that modify individual shots.

The user may enter a list of values for a global, such as `[1,2,3]`, or `linspace(0,10,100)`. In this case `runmanager` produces a corresponding list of experiment shots: one for each value. If multiple globals are entered as lists, `runmanager` performs a Cartesian product, creating one shot for each point in the resulting parameter space. Two or more lists can be *zipped*, in which case `runmanager` iterates over these lists in lock-step when producing shots.

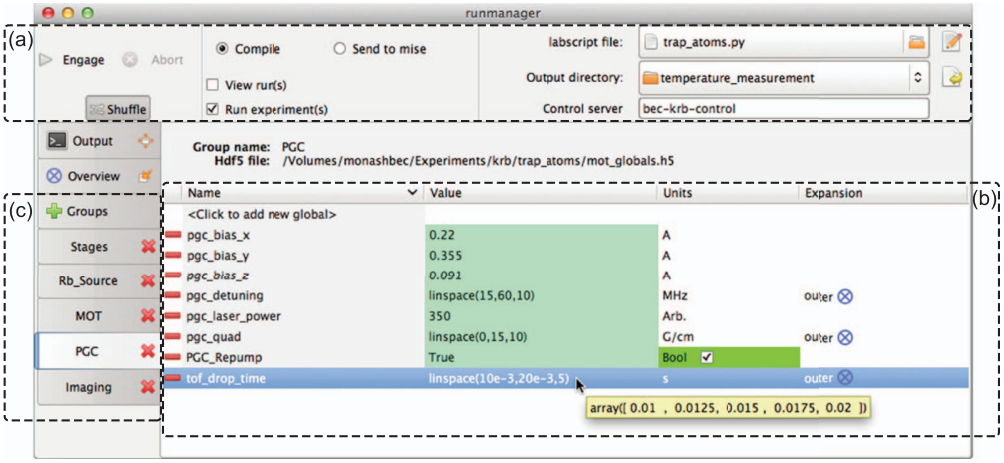

FIG. 4. The `runmanager` interface for configuring experiment parameters. (a) The experiment logic is specified by the labscript file (here trap_atoms.py). HDF files for experiment shots created by `runmanager` are saved in the output directory. (b) The value of experiment parameters ("globals") are specified by Python expressions and may have units. These can be single values (i.e., `350` or `True`), lists, or expressions creating lists (as shown for the globals `pgc_detuning`, `pgc_quad`, and `tof_drop_time`). A tooltip shows the evaluation of the global. The "Expansion" column specifies how lists of values are combined to construct a parameter space. (c) Globals can be separated into groups for convenience.

Specifying globals as lists makes it possible to explore complicated parameter spaces containing hundreds or thousands of shots. For example, one might investigate how the temperature of laser cooled atoms varies with laser detuning and magnetic field gradient. Taking the Cartesian product of ten field strengths and ten detunings results in a parameter space of one hundred points. Thermometry at each point in this parameter space commonly requires multiple shots to characterize the expansion rate of the atom cloud. A five-shot temperature measurement brings the number of shots to five hundred. Producing these shots amounts to entering three lists in `runmanager` and clicking on the "Engage" button, as shown in Fig. 4. `runmanager` then creates five hundred HDF files containing the globals for each shot. The experiment script is run for each shot, storing hardware instructions in each file.[20] The HDF files are then submitted to BLACS for execution.

VI. EXPERIMENT EXECUTION—BLACS

`BLACS` coordinates input and output through hardware devices. These devices can be local, and thus under the direct control of `BLACS`, or connected to a different computer as part of a secondary control program such as `BIAS` (Sec. VII). `BLACS` provides both manual control of devices (through a GUI) and buffered execution of experiment shots.

The GUI for manual control is dynamically generated from a *lab connection table* that describes the current configuration of all connected devices. Each device is allocated a tab in the interface, containing controls for commanding output when in manual control mode (Fig. 5).

Upon submission to `BLACS`, HDF files containing hardware instructions are checked for validity and placed in a queue. The queue can be reordered, paused, or put on repeat. The validity check compares the connection table of each shot to the lab connection table, rejecting those with incompatible hardware. This prevents unintended device output that would produce nonsensical results and possibly damage equipment.

`BLACS` takes the first experiment in the queue, coordinates hardware programming, and sends a start trigger to the master pseudoclock. The experiment then proceeds under hardware timing. At the end of a shot, `BLACS` coordinates saving data acquired by devices to the HDF file, and returns to manual control mode. Each GUI control is updated to the final values of the shot, maintaining output continuity.

Laboratories are a hostile environment for hardware interface libraries. Power cycling of devices and unplugging of cables are common occurrences. A student tripping over a USB cable (health and safety implications notwithstanding) might be expected to cause an experiment to fail, however the control system ought to recover gracefully when it is plugged back in. Similarly, bugs in closed source drivers and libraries are points of failure outside of a users control.

To make our system robust against such hardware and software failures, `BLACS` implements a multiprocess architecture similar to the sandboxed tabs of the Google Chrome web browser.[21] For each device in `BLACS`, a *worker process* is spawned, which communicates with the hardware device. This makes `BLACS` robust against crashes: if one device has a problem it will not affect others. If a hardware device becomes unresponsive, or the device driver encounters a serious error, its isolation in a separate process prevents the GUI and other devices from suffering the same fate.

Should a worker process crash, the user is presented with the option of restarting the process, which will reload any device libraries it uses. It is worth noting that systems implemented in LabVIEW cannot force libraries to reload, so errors leading to an undefined state would only be remedied by restarting the entire control system.

The initialization of hardware in preparation for a shot is an important part of an experiment, and can significantly contribute to the experiment cycle time. The multiprocess

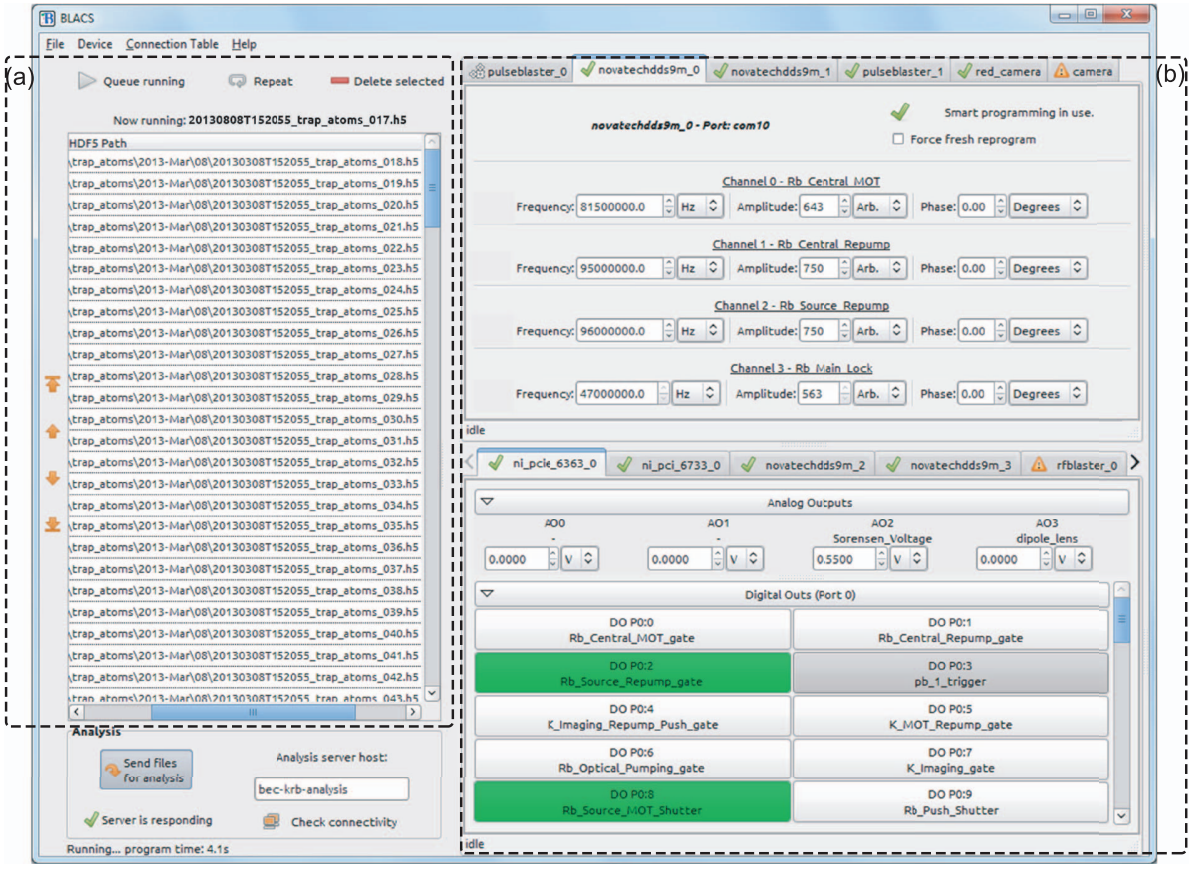

FIG. 5. The BLACS interface for controlling hardware. (a) The queue of shots submitted via runmanager. (b) The manual control interface. Each tab controls one device. Controls for all outputs are automatically generated and are named based on the BLACS connection table.

architecture naturally provides for simultaneous programming of hardware devices, resulting in an increased experiment duty cycle. We have implemented a *smart programming* feature on many of our devices, further decreasing programming time, reprogramming them only if their instructions have changed since the previous shot (on a per-instruction basis when possible). Devices with large buffers and slow communication (such as the Novatech DDS9m rf synthesizer) benefit greatly from this technique.

VII. IMAGE ACQUISITION—BIAS

Using secondary control programs to communicate with specific devices is desirable when software to do so exists and has been debugged, particularly software written in another programming language. BLACS integrates such programs into the control flow by sending them HDF files containing hardware instructions to program devices for execution upon a hardware trigger. BLACS notifies secondary control programs that the shot has completed, at which point they write any acquired data to the HDF file.

Our camera control and image acquisition system, BIAS, is one such program. BIAS is a LabVIEW application that operates scientific cameras, captures image sequences, and performs image processing tasks such as background subtraction, saturation correction, optical depth calculation, and simple 2D fitting.

Multiple instances of BIAS can be run simultaneously to control multiple cameras in one experiment. BIAS can also run as a stand-alone program for quick visualization of previously captured data or acquire images manually. Hardware communication in BIAS is abstracted through LabVIEW's object hierarchy, allowing a camera class to be written for any vendor library.

LabVIEW provides convenient components for creating graphical interfaces, and BIAS displays raw and computed images as they become available (Fig. 6). Fit results such as atom cloud shape and atom number are prominently displayed to detect and diagnose problems as they occur. The camera acquisition area and regions of interest used to inform fits can be interactively adjusted, without needing to interrupt or recreate a currently running sequence of shots. Multiple regions of interest can be selected and their coordinates saved to the HDF file, enabling further analysis.

VIII. ANALYSIS—LYSE

Analysis is a critical part of an autonomous control system. Automated analysis—performed immediately after every shot—is often restricted to routines that change infrequently and are applied uniformly once per shot. Ideally analysis should be flexible as well as autonomous; these can be conflicting goals without a unifying analysis framework.

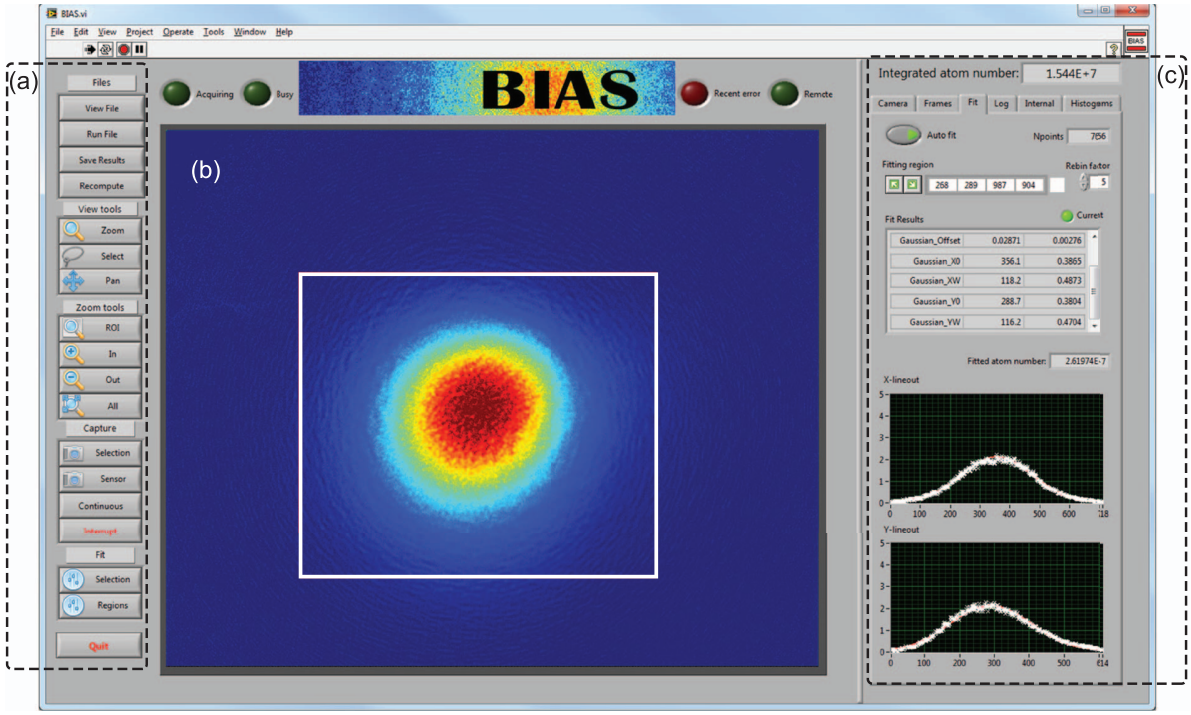

FIG. 6. The `BIAS` interface displaying a laser cooled atom cloud. (a) Manual controls for loading and capturing images, selecting regions of interest and zooming. (b) Computed optical depth (OD) image of the atoms, with a region of interest (white) selected for fitting. Multiple regions of interest may be selected for multi-component atom clouds. (c) Atom number and cloud size are displayed for immediate feedback.

Our analysis system `lyse` accommodates collective analysis of a group of shots and trivial re-analysis upon changing or adding routines.

`lyse` is a scheduler for user-written analysis routines, which are ordinary Python scripts. It provides functions for extracting the experiment data and metadata from the HDF files and saving analysis results to these files. Multiple analysis routines added to `lyse` execute one after the other when a new HDF file is received over the network, or on command through the GUI. Plots produced by the user's code are updated following every shot as new data comes in from the experiment.

There are two types of routine that `lyse` can run: single-shot, which are run on every shot, and multi-shot, which analyze a group of shots together. Analysis of the thermometry example in Sec. V is shown in Fig. 7. A single-shot routine computes the size of an atom cloud after a fixed expansion time, and a multi-shot routine uses these results to determine the expansion rate and thus the temperature. The multi-shot routine then plots this temperature as a function of laser detuning and magnetic field strength.

Splitting, sorting, plotting, and exploring large multidimensional datasets are cumbersome when directly accessing a set of files. In addition to direct access to the HDF files, `lyse` provides a tabular data structure—a `pandas`[22] DataFrame—for multi-shot routines, containing all globals as set by `runmanager`, and all single-shot analysis results. With `pandas` and the standard Python scientific stack of `numpy`,[23] `scipy`,[24] and `matplotlib`,[25] `lyse` provides a powerful environment for analysis.[26]

Analysis routines can be run independently of `lyse` if desired. This allows the same framework and analysis code to be used for publication preparation.

IX. OPTIMIZATION—MISE

Marrying powerful Python tools to shot-based analysis permits extensibility of the control system, such as closed loop optimization of measured quantities. One often performs parameter space scans for optimization, requiring many shots. This may be tuning a parameter of an apparatus to enhance its performance, finding a resonance of some transition, or some other feature of interest. The quantity being optimized is often the result of some analysis, e.g., the temperature of ultracold atoms (mentioned in Secs. V and VIII). We have created `mise`, a program that performs automatic optimization of analysis results using a genetic algorithm.[27] A user specifies one or more parameters to optimize against a predefined figure of merit. Genetic algorithms are resistant to noise, making them particularly useful for optimizing experimental results.

The data flow of the optimization process follows Fig. 8, modifying that shown in Fig. 1. The user specifies in `runmanager` one or more parameters to optimize, with upper and lower limits for each. An analysis routine in `lyse` reports optimality to `mise`, which creates shots with modified parameters and submits them to `BLACS`.

For each parameter being optimized the user also specifies a *mutation rate*. This determines how much the parameter is varied per generation of the genetic algorithm:

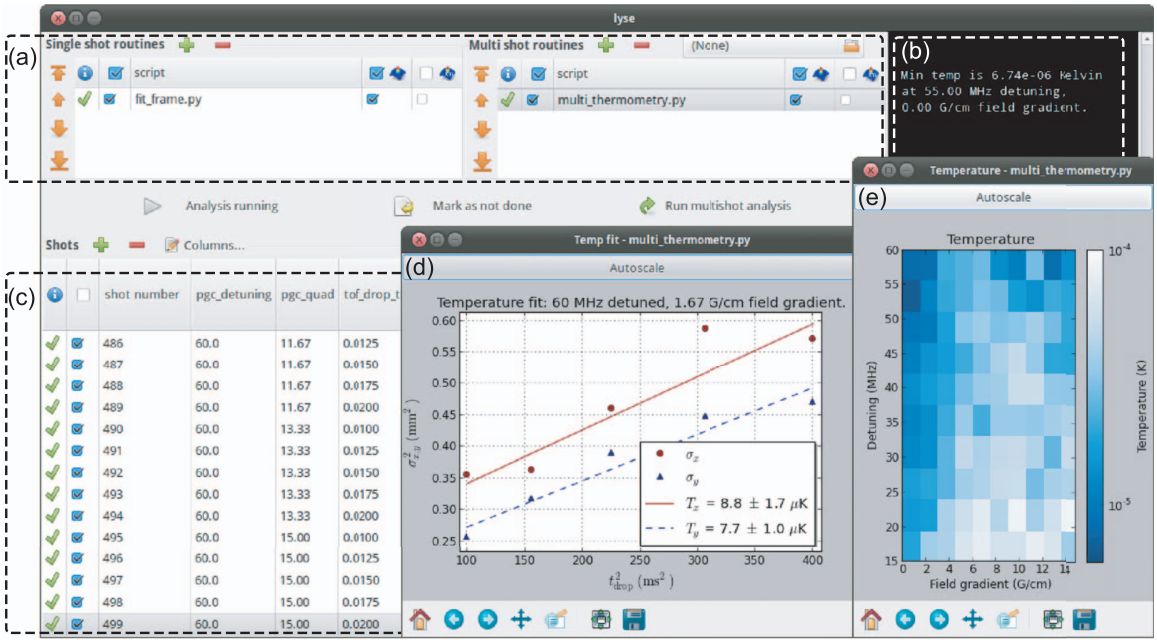

FIG. 7. The `lyse` interface. (a) Routines can be selected to analyze single or multiple shots. (b) Terminal output from the analysis routines in (a). (c) Table of shots; columns show globals and analysis results. A small subset of columns is displayed here. (d) A fit yielding the temperature of laser cooled atoms prepared at a particular field gradient and detuning. (e) The results of the analysis in (d) repeated at each point in the parameter space.

the larger the mutation rate, the faster `mise` will move towards the optimum. However, a large mutation rate limits the precision to which the optimal parameters can be determined.

With this specification of parameters, `mise` creates a population of *individuals*. Each individual comprises values from one point in the optimization parameter space, initially chosen at random. An individual may be a single experiment shot, or—when optimizing the result of a multi-shot analysis—a sequence of shots. Once the shots comprising an individual have executed, the user's analysis routine computes a *fitness*, which may be derived from any measured quantity. `mise` uses the reported fitness in the genetic algorithm to optimize the specified parameters. The genetic algorithm used by `mise`[28] is a variation on pointed directed mutation,[29] in which mutations are biased in directions previously shown to be successful.

The user can specify when to stop the optimization, either by manual intervention or by a convergence condition written into their analysis script. They may also "guide" the evolution by adding and deleting individuals from the gene pool at any time.

An example of automated optimization using `mise` is shown in Fig. 9. By preferentially exploring the more interesting regions of parameter space, autonomous optimization allows optima to be found in fewer shots.

`mise` uses the `labscript` software library to create HDF shot files and submit them to `BLACS`. Additional user-written components could similarly submit shots to `BLACS` if more complex programmatic generation of shots is required.

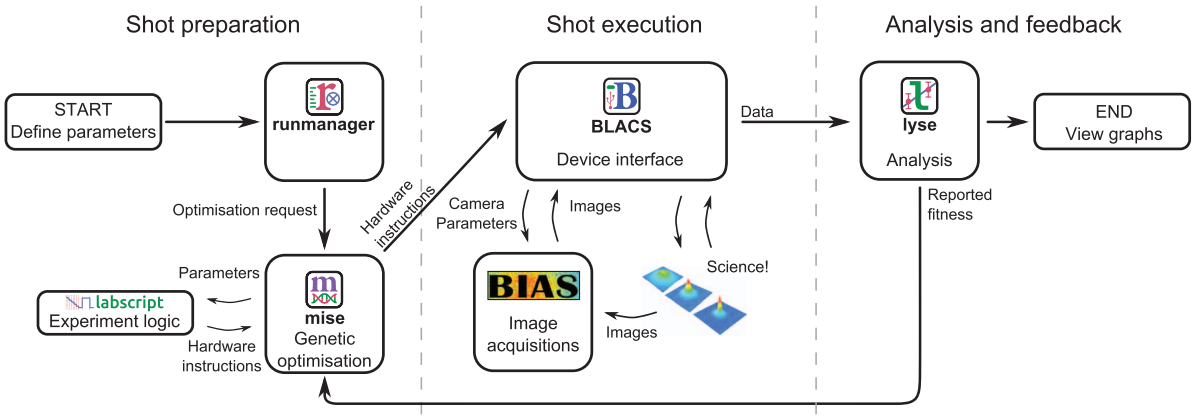

FIG. 8. The data flow for closed loop optimization. In contrast to Fig. 1, analysis results are used to determine future shots automatically. The optimizer `mise` varies parameters, directly calling `labscript` to compile new experiment shots. Parameters to be optimized are selected by the user in `runmanager`. `lyse` reports fitness to `mise` which is used to create the next generation of shots.

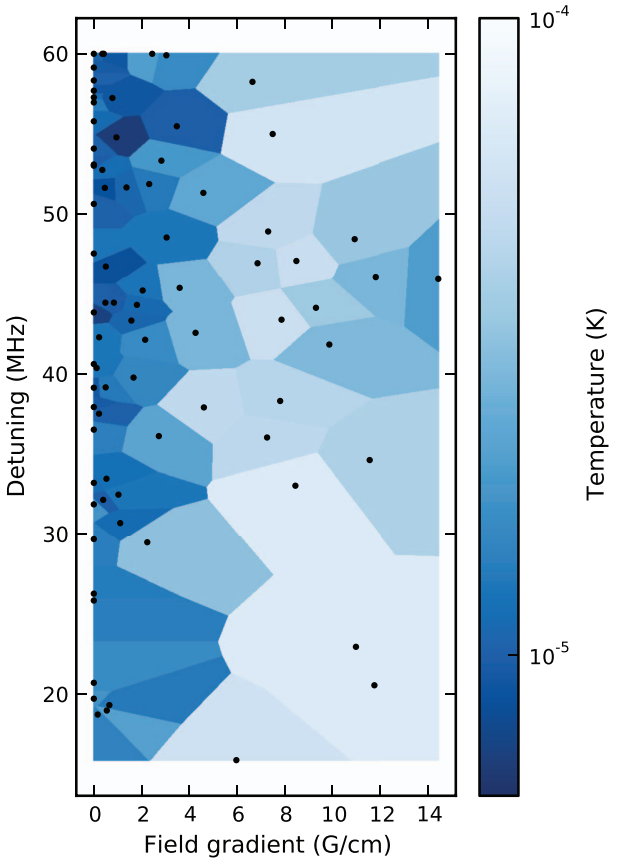

FIG. 9. A proof-of-principle optimization using `mise`. `mise` scanned the parameter space described in Sec. V, searching for the coldest point. Each black point represents a temperature measurement at a specific field gradient and detuning, with the surrounding shading indicating the temperature. Eighty points were taken, corresponding to 400 shots. The colder region of parameter space is sampled more densely than the uniformly-sampled scan, shown in Fig. 7(e), with 500 shots.

X. PORTABILITY AND EXTENSIBILITY

Our software runs on Windows, Linux, and OS X, although `BLACS` and `BIAS` compatibility is subject to the availability of appropriate hardware drivers. If particular devices must be interfaced with a specific computer, operating system, or programming language, a secondary control program (such as `BIAS`, Sec. VII) can be used. The components of the labscript suite communicate with each other via data in HDF files, and over the network with ZeroMQ sockets. The widespread support of these technologies across many platforms[30] ensures users are not bound to any one operating system or programming language. The modular nature of our system allows users to replace or supplement any of our programs in their choice of language.

The programs themselves are also written with extensibility in mind. Adding new hardware support to the labscript suite entails writing a new device class for `labscript`, and a GUI tab for `BLACS`,[31] or a camera class for `BIAS`. Adding analysis routines to `lyse` amounts to writing a Python script to process experiment data. Existing library functions and base classes assist such development. The suite has already proved useful in a setting distinct from quantum science experiments, automating the prototyping of an objective lens, in which the image of a pinhole was acquired and analyzed at 3600 points in a plane to determine the field of view.[32]

The labscript suite is open-source and freely available online.[33] We encourage readers to contact us if they are interested in implementing the suite in their laboratory.

ACKNOWLEDGMENTS

The authors would like to thank the current users of our system, who were not part of the development team, L. Bennie, M. Egorov, and A. Wood for their input into making this a better system. This work was supported by Australian Research Council Grant Nos. DP1094399 and DP1096830.

[1]See, e.g., M. Weidemüller and C. Zimmermann, *Cold Atoms and Molecules* (Wiley, 2009), and references within.
[2]See, e.g., N. Robins, P. Altin, J. Debs, and J. Close, "Atom lasers: Production, properties and prospects for precision inertial measurement," Phys. Rep. **529**, 265 (2013); A. D. Cronin, J. Schmiedmayer, and D. E. Pritchard, Rev. Mod. Phys. **81**, 1051 (2009), and references within.
[3]See, e.g., A. Negretti, P. Treutlein, and T. Calarco, Quantum Inf. Process. **10**, 721 (2011); T. D. Ladd, F. Jelezko, R. Laflamme, Y. Nakamura, C. Monroe, and J. L. O'Brien, Nature (London) **464**, 45 (2010), and references within.
[4]See, e.g., I. Bloch, J. Dalibard, and S. Nascimbène, Nat. Phys. **8**, 267 (2012); R. Blatt and C. F. Roos, *ibid.* **8**, 277 (2012), and references within.
[5]G. Varoquaux, Comput. Sci. Eng. **10**, 55 (2008).
[6]P. E. Gaskell, J. J. Thorn, S. Alba, and D. A. Steck, Rev. Sci. Instrum. **80**, 115103 (2009).
[7]R. P. Anderson, Ph.D. thesis, Swinburne University of Technology, 2010.
[8]M. Beeler, Ph.D. thesis, University of Maryland, 2011.
[9]P. A. Altin, Ph.D. thesis, Australian National University, 2012.
[10]T. Stöferle, Ph.D. thesis, Swiss Federal Institute of Technology, 2005.
[11]A. Keshet and W. Ketterle, Rev. Sci. Instrum. **84**, 015105 (2013).
[12]S. F. Owen and D. S. Hall, Rev. Sci. Instrum. **75**, 259 (2004).
[13]T. Meyrath and F. Schreck, "A laboratory control system for cold atom experiments," see http://www.strontiumbec.com/indexControl.html (2012).
[14]P. Hintjens, *Code Connected Volume 1: Learning ZeroMQ* (CreateSpace Independent Publishing Platform, 2013); see also "ØMQ: the intelligent transport layer," http://www.zeromq.org/.
[15]The HDF Group. Hierarchical data format version 5, 2000-2010. http://www.hdfgroup.org/HDF5.
[16]"IEEE Standard Codes, Formats, Protocols, and Common Commands for Use With IEEE Std 488.1-1987, IEEE Standard Digital Interface for Programmable Instrumentation," IEEE Std 488.2-1992.
[17]G. van Rossum *et al.*, "Python programming language v2.7," see http://docs.python.org/2.7/ (2010).
[18]J. M. Hughes, *Real World Instrumentation with Python: Automated Data Acquisition and Control Systems* (O'Reilly Media, Inc., 2010).
[19]The source code for turning a ChipKIT Max32 into a PineBlaster is available at http://hardware.labscriptsuite.org/.
[20]In our lab, a typical hardware set for running this experiment would be a SpinCore PulseBlaster DDS-II-300-AWG as a pseudoclock, along with a Novatech DDS9m, National Instruments PCIe6363 and PCI6733 boards and a Photonfocus MV1-D1312(I) camera.
[21]A. Barth, C. Jackson, C. Reis, and The Google Chrome Team, "The security architecture of the chromium browser," Technical Report, Stanford Security Laboratory, 2008, available at http://seclab.stanford.edu/websec/chromium. See http://youtu.be/29e0CtgXZSI for more information.
[22]W. McKinney, "pandas: a Python data analysis library," see http://pandas.pydata.org/.
[23]T. Oliphant, "NumPy: numerical Python," see http://www.numpy.org/.
[24]E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: open source scientific tools for Python," see http://www.scipy.org/ (2001).
[25]J. Hunter, Comput. Sci. Eng. **9**, 90 (2007); see also "matplotlib: Python plotting," http://matplotlib.org/.
[26]W. McKinney, *Python for Data Analysis* (O'Reilly Media, Inc., 2012).
[27]T. Bäck and H.-P. Schwefel, Evol. Comput. **1**, 1–23 (1993).

[28] See supplementary material at http://dx.doi.org/10.1063/1.4817213 for implementation details of the genetic algorithm used by `mise`.

[29] A. Berry and P. Vamplew, "PoD Can Mutate: A Simple Dynamic Directed Mutation Approach for Genetic Algorithms," in *AISAT2004: International Conference on Artificial Intelligence in Science and Technology*, 21–25 November 2004, Hobart, Tasmania, Australia.

[30] HDF bindings include C/C++, MATLAB, Python, LabVIEW and Mathematica. ZeroMQ support includes C/C++, Python, LabVIEW, Java and many more. See http://www.hdfgroup.org/products/hdf5_tools/ and http://www.zeromq.org/bindings:_start/ for more complete lists.

[31] `BLACS` communicates with hardware devices through user-written interface code. Devices communicating over standard buses (RS232, USB, Ethernet) are easily interfaced using standard Python libraries for these buses. Devices with proprietary interfaces can be programmed by calls to vendor-supplied libraries through Python's sophisticated foreign-function interface.

[32] L. M. Bennie, P. T. Starkey, M. Jasperse, C. J. Billington, R. P. Anderson, and L. D. Turner, Opt. Express **21**, 9011 (2013).

[33] "The labscript suite: an open source experiment control and analysis system," see http://labscriptsuite.org/.

Chapter 5

Particle velocimetry of vortices in Bose–Einstein condensates

This chapter investigates, via numerical simulation, an imaging method for the real time tracking of quantum vortices in a turbulent ^{41}K condensate. The method involves ultracold ^{87}Rb tracer 'particles' (atoms) that become bound to vortex lines in the condensate and are imaged continuously to track the vortex lines as they move. The resulting images—either multiple images of the vortices at different times, or a single exposure with vortex trajectories visible as traces—can be used to infer the motion of the vortex cores. The imaging of tracer particles to track vortex motion has previously proved successful in superfluid helium [131–133]. Imaging cold atoms without excessive heating necessitates a cooling mechanism—the method of laser cooling and imaging atoms in high resolution with the same laser light has also been successful in cold atom systems [134]. This chapter presents the results of numerical simulations of the method under a number of assumptions to establish its feasibility as an imaging method. I present a new sub-Doppler laser cooling scheme designed for a 34 G magnetic field. This scheme could be used to cool tracer atoms at the magnetic field strength required for a Feshbach resonance that enhances the interaction between the tracer atoms and the BEC. I also briefly discuss an additional cooling scheme, proposed by Prof. Helmerson, that involves using the vortices themselves to provide a state-selective force that could be used for cooling. The state-selective force—central to the proposed cooling effect—was not possible to model semiclassically in the same way as other laser cooling schemes, which was one of the motivations that led me to develop the hidden-variable semiclassical method discussed in Chapter 6.

This method has the potential to overcome several difficulties that imaging techniques face when used to image vortices. In ordinary absorption imaging, atoms are imaged via resonant absorption, and vortices—visible as density minima—generally can only be seen when the vortex line is parallel to the illumination. If not viewed end-on in this way, a vortex line represents only a minor decrease in column density and cannot be distinguished from the rest of the condensate (Figure 5.1). One solution to this problem is to slice the condensate into layers, and image them separately [135].

The use of tracer particles that are only present within vortex cores allows vortex lines to be visible from any viewing angle. Furthermore, since the atoms being imaged reside in the vortex cores themselves rather than the bulk of the condensate, this imaging can potentially be repeatedly or continuously performed without destroying the condensate. This may enable observation of the time evolution of Kelvin waves [136], vortex reconnections [137], and vortex rings [135].

This *in-situ* imaging of vortex dynamics may allow more types of vortex motion to

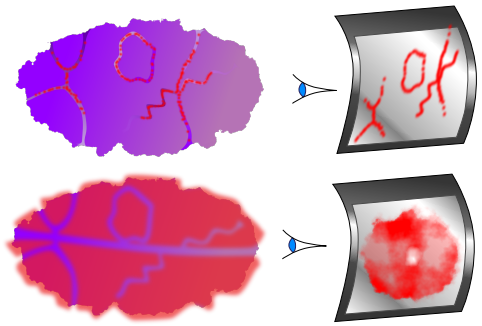

Figure 5.1: Imaging of the condensate itself, whether by fluorescence (bottom) or absorption imaging makes it difficult to resolve vortices unless they are viewed end-on. The vortex cores are usually smaller than the imaging wavelength, and are thus also difficult to resolve unless the cloud is allowed to expand. Imaging tracer particles instead (top) has the potential to resolve both problems.

be imaged. Dynamics of BECs are typically studied using a shot-by-shot method, in which repeated experiments with identical initial conditions are imaged destructively after being allowed to evolve for different amounts of time. Whilst this works for many types of dynamics, it fails for experiments that are sensitive to initial conditions and noise (quantum or otherwise), such as turbulent flow. This includes phenomena which cannot be created reliably in the same initial state, even though the evolution thereafter would be consistent from one experimental run to the next. One such phenomenon is the spontaneous generation of vortices after evaporative cooling [138].

In-situ imaging of vortex motion has been achieved previously [139], by ejecting a fraction of the atoms from the condensate periodically and imaging them. This process is limited by depletion of the condensate, and was also used only to image vortices end-on. The fraction of the condensate being imaged was also allowed to freely expand before being imaged, since vortex cores are otherwise unable to be resolved by the wavelength of light used. Our proposed method would require neither free expansion or depletion of the condensate.

5.1 Motivation: Turbulence

It is commonly said that turbulence is one of the greatest unsolved problems of classical physics. But in what sense is it an unsolved problem? It is not a problem at all if your aim is reductionism—the Navier–Stokes equation adequately describes the evolution of a Newtonian fluid within its domain of validity, including rich turbulent phenomena such as turbulent boundary layers, Rayleigh–Bénard convection [140], and energy cascades [141].

and the process of deriving it from the underlying motion of classical particles is well understood. It's turtles all the way down [142, p 1]; what more could we ask for?

A demonstrative comparison might be with the field of statistical mechanics, as precisely the same statement can be made about the energy content and exchange between systems of particles. Statistical mechanics has revealed that despite the chaotic motion of individual particles in an ensemble, definite statements can still be made about the behaviour of the system as a whole, *without having to consider the dynamics of the constituent components in detail*.

This is the kind of solution people have in mind when they speak of 'solving' the problem of turbulence. Laws describing the average properties of a fluid without reference to its precise flow field would not simply be interesting because they describe turbulence as an emergent phenomenon, but would aid practical computations, which for many problems of interest are prohibitively computationally expensive. The flow of a turbulent fluid contains detail on such a wide range of length scales that finite-element or finite-difference analyses of a system such as an aeroplane wing requires a very high resolution in order to be accurate. Following an estimate of computing power required to simulate a turbulent system down to its smallest length scales, Stanley Corrsin quipped [143]:

> The foregoing estimate is enough to suggest the use of analog instead of digital computation; in particular, how about an analog consisting of a tank of water?

The reliance of the aerospace industry on wind tunnels and practical tests shows that there is some truth to the necessity of using nature as one's computer when it comes to turbulence. Whilst nature must always have the final say, it would be of great benefit to be able to compute expected results more cheaply before setting up a wind-tunnel experiment or constructing a prototype aircraft. Gaining predictive power through an improved ability to reason about turbulence would help escape to evolutionary model of testing and modifying prototypes of systems subject to turbulent flow.

But are we asking for too much? Perhaps the statistical properties of a turbulent fluid fundamentally cannot be decoupled from the finer details. There is reason to believe that this is not the case. There are several tantalising results that hint at universal properties that all turbulent flows share, and there is the simple empirical observation that the average flow of turbulent fluids at large scales is reproducible from one experimental run to the next [144, pp 13, 86].

One of these universal results is Kolmogorov's theory of the statistics of small eddies [145, 146]. Another is the fact that the rate of energy dissipation via the action of viscosity at small scales is independent of the viscosity itself [144, p 77]. Then there is the Richardson energy cascade [147], in which energy is continually transferred from larger scales to smaller scales. With dissipation at the smallest scales and energy injection at larger scales, this allows for the existence of 'steady state' turbulence.

The above examples derive from ordinary, viscous fluids. Bose–Einstein condensates on the other hand are superfluids. There are several interesting aspects of superfluid turbulence that differ from classical turbulence. The defining difference is the absence of viscosity; another major difference is the quantisation of circulation. On length scales much larger than spacing between vortex lines, superfluid turbulence is expected to closely resemble classical turbulence [148]. At smaller scales however the energy dissipation mechanism is different, instead involving the production of sound waves via vortex interactions [148, 149].

In certain 2D geometries, an *inverse cascade* [150, 151] is predicted to take place in superfluids, whereby energy moves not from large scales to small, but from small to large, clustering quantised vortices of the same circulation direction together. This phenomenon has been studied theoretically and numerically in the Monash Quantum Fluids group [152, 153] and recently experimentally observed in the Monash Dual-Species laboratory in experiments performed by Shaun Johnstone [109], simultaneously with a group at the University of Queensland [154].

5.1.1 Characterisation of turbulence as vortex dynamics

The following definition of turbulence, taken from [144, p 53], emphasises the role of vortices in turbulence in general:

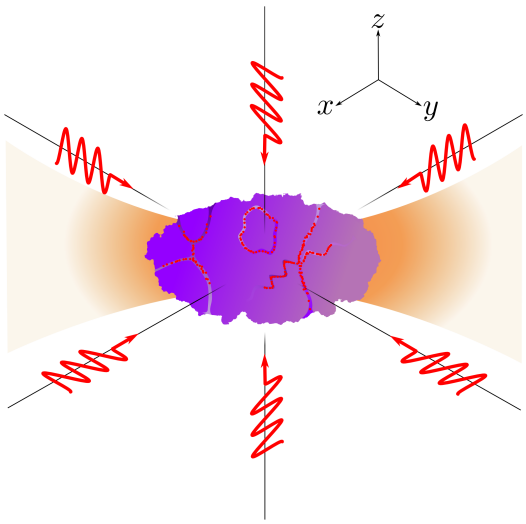

Figure 5.2: The simplest scheme for cooling and imaging the tracer particles with the same light is polarisation gradient cooling: six red-detuned beams (red), with each counter-propagating pair having opposite linear polarisations. Light scatters off the tracer atoms and cools them to sub-Doppler temperatures. If the cooling is sufficient, atoms move into the vortex cores where their energy is lower, if they aren't already there. Both the rubidium tracers and the potassium BEC are trapped with approximately the same trapping potential by a strong, far off-resonant laser (orange), via the dipole force. Magnetic trapping cannot be used, as polarisation gradient cooling does not work in the presence of a magnetic field.

> Incompressible hydrodynamic turbulence is a spatially complex distribution of vorticity which advects itself in a chaotic manner in accordance with [the vorticity equation[1]]. The vorticity field is random in both space and time, and exhibits a wide and continuous distribution of length and time scales.

[1] The vorticity equation is a transformation of the Navier–Stokes equation for an incompressible fluid into a form that describes the vorticity field directly, rather than the velocity field.

When vorticity exists only in infinitesimally narrow lines, as it does in a superfluid, the vorticity equation mentioned in the above definition reduces to a Biot–Savart-type law which can be used to compute the motion of vortices without having to compute the entire flow field. This is why we are interested in the study of the dynamics of quantised vortices. Unlike in classical fluids, the vortices in superfluids have a definite position and size; there either is a vortex line within a given spatial region or there is not. This may make it simpler to describe the motion of vortices statistically.

So far experimental studies of superfluid turbulence have been primarily in the context of liquid helium [155]. Bose–Einstein condensates offer a compelling alternative subject of study for superfluid turbulence. The high degree of control afforded over systems of cold atoms allows superfluid properties to be tweaked in several ways, such as modifying their density, temperature, trapping potential, interaction strength, or even the effective mass of fluid particles via the application of a periodic potential. This control creates a larger parameter space in which to study turbulence than that afforded by liquid helium.

Vortices are the skeletons of turbulence—to study turbulence via the behaviour of vortices, we need to track them.

5.2 Overview of velocimetry scheme

As mentioned, the core idea of our proposed imaging method is to use tracer particles to track vortex cores in a BEC. In this chapter I consider ^{87}Rb atoms as tracer particles in a BEC made of ^{41}K. This choice is due to the strong interspecies repulsion between these atomic species, which gives rise to the trapping of atoms in the vortex cores. In the limit of low densities and temperatures, such that three body collisions are suppressed and s-wave scattering dominates the interspecies interactions [156, p 120], the rubidium tracer atoms experience a potential due to the potassium:

$$V(\boldsymbol{r}) = \frac{2\pi\hbar^2 a_s}{m_r}\rho_{\mathrm{K}}(\boldsymbol{r}), \tag{5.1}$$

where $\rho_{\mathrm{K}}(\boldsymbol{r})$ is the spatially varying atom density of the potassium condensate, a_s is the interspecies s-wave scattering length, and $m_r = \frac{m_{\mathrm{K}} m_{\mathrm{Rb}}}{m_{\mathrm{K}}+m_{\mathrm{Rb}}}$ is the reduced mass of the scattering pair. Vortex cores thus create potential wells for other atoms, since they are regions of low condensate density in a background of higher density.

The scheme is shown in Figure 5.2. Cold rubidium atoms are introduced (possibly via magnetic transport from a cold–but not necessarily condensed—source) to a potassium condensate, after which both species are optically trapped at the focus of a high power 1064nm laser, using the dipole force. Various methods may be used to create vortices in the condensate. These include bluff-body flow, where a repulsive potential is dragged through the condensate, or inducing a turbulent state by applying off-resonant laser speckle. The rubidium atoms are then expected to become trapped in the low density vortex cores.

The tracer atoms are imaged with resonant or near-resonant laser light, depending on the exact scheme employed. The scattering of imaging photons heats the tracer atoms, which, if sufficient to cause them to escape the vortex cores, presents a problem for acquiring images of vortex motion. In Section 5.4 I present the results of simulating tracer atoms imaged with resonant light. This light does not provide any cooling, instead the simulation includes the cooling effect of the condensate itself. These results show that sympathetic cooling of the tracer atoms by the condensate may be sufficient to retain tracer atoms in vortices whilst scattering imaging light.

The simplest scheme which attempts to laser-cool the rubidium atoms is ordinary polarisation gradient cooling, in which the same light is used for imaging and cooling the atoms (Figure 5.2). This was considered in my Honours thesis [157], the results of which I summarise in the next section. This method precludes the use of a magnetic trap or large bias field, since either would destroy the cooling effect.

The vortex potentials may be made deeper through the use of a Feshbach resonance (Section 2.1.6), which increases the interspecies scattering length. However, since this requires a magnetic field, it precludes the use of ordinary polarisation gradient cooling. In section (see Section 5.5) I present an alternative polarisation gradient cooling scheme designed work in the presence of a magnetic field of the strength required for the Feshbach resonance of interest.

Effective imaging of vortex motion would require approximately 10^5 photons per second to scatter off each rubidium atom without it escaping its vortex core trap, and without causing so much heating as to destroy the condensate on a reasonable experimental timescale. A high resolution, low aberration lens (numerical aperture ≈ 0.5) would also be required to focus the scattered light onto a fast capture, high quantum efficiency camera to produce images of vortex motion.

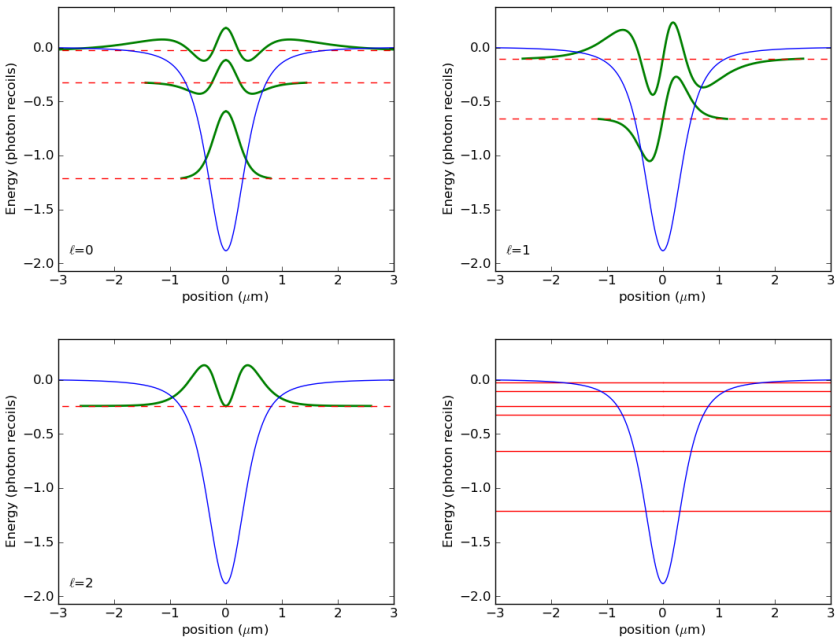

Figure 5.3: Energy eigenstates of a rubidium atom in a potassium vortex core, for a potassium BEC with background density $\approx 10^{14}$ cm^{-3}. There are a number of bound states spanning three orbital quantum numbers. Plots of the bound states are of a cross section through the centre of a vortex core, and the lower right plot shows just the energy levels (for all orbital quantum numbers ℓ). Figure reproduced from [157].

5.3 Relation to previous work

This scheme was first investigated in my Honours project [157]. In that work I investigated the ability of vortex potentials to trap atoms, including consideration of the depth of such traps when measured in units of the photon recoil energy. Considering the depth in these units was a first attempt to estimate how easily atoms may escape vortex potentials in the presence of imaging light. Figure 5.3 and Figure 5.4 show bound states of typical vortex potentials at different condensate densities.

To minimise the recoil energy, rubidium is a better choice for tracer particle than potassium due to its larger mass, enabling a rubidium atom to scatter more photons before escaping a vortex potential compared to a potassium atom. Vortex potentials are not very deep when measured in recoil energies, and their depth depends strongly on the density of the BEC. At typical condensate densities of 10^{14} cm^{-3}, the vortex potentials are expected to only be 1–2 recoil energies deep, making it unlikely that atoms could scatter many photons whilst remaining trapped in them. At larger densities of 10^{15} cm^{-3}, the vortex potentials are closer to 20 recoil energies deep, making imaging of trapped tracer atoms more plausible.

The main simulation result of my Honours project considered a potassium condensate with a peak density of 10^{15} cm^{-3} and rubidium tracer atoms being cooled using standard polarisation-gradient cooling with parameters chosen to ensure each atom scattered on the order of 10^5 photons per second. The result was that initially randomly distributed rubidium atoms were able to become—and remain—trapped in the vortex cores whilst being cooled (Figure 5.5).

However, the density assumed in this simulation was rather high for a real experiment.

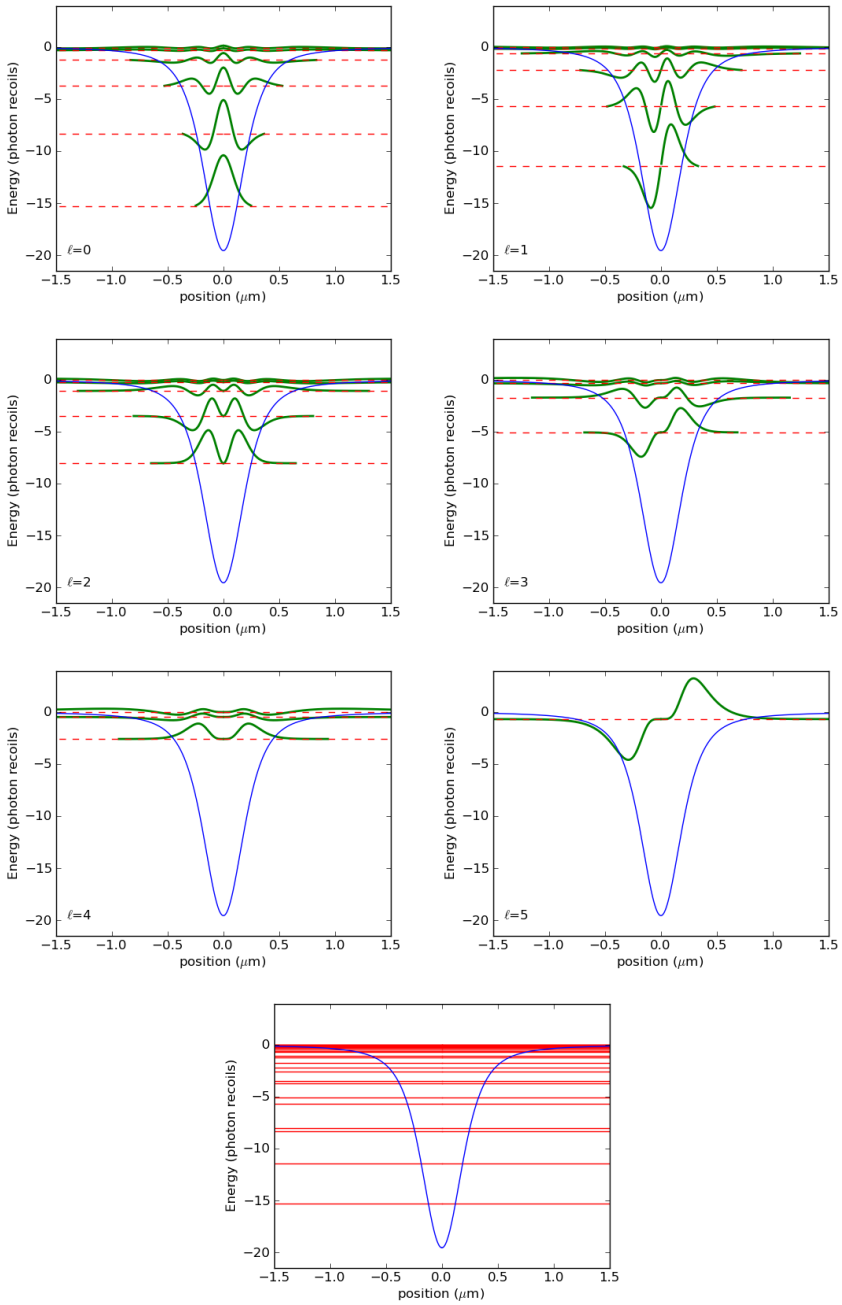

Figure 5.4: As in Figure 5.3, but for a potassium BEC with background density $\approx 10^{15}$ cm^{-3}. There are bound states over six different orbital quantum numbers. This vortex potential is much deeper than that in Figure 5.3, showing the effect of condensate density on the depth of the vortex potentials . Figure reproduced from [157].

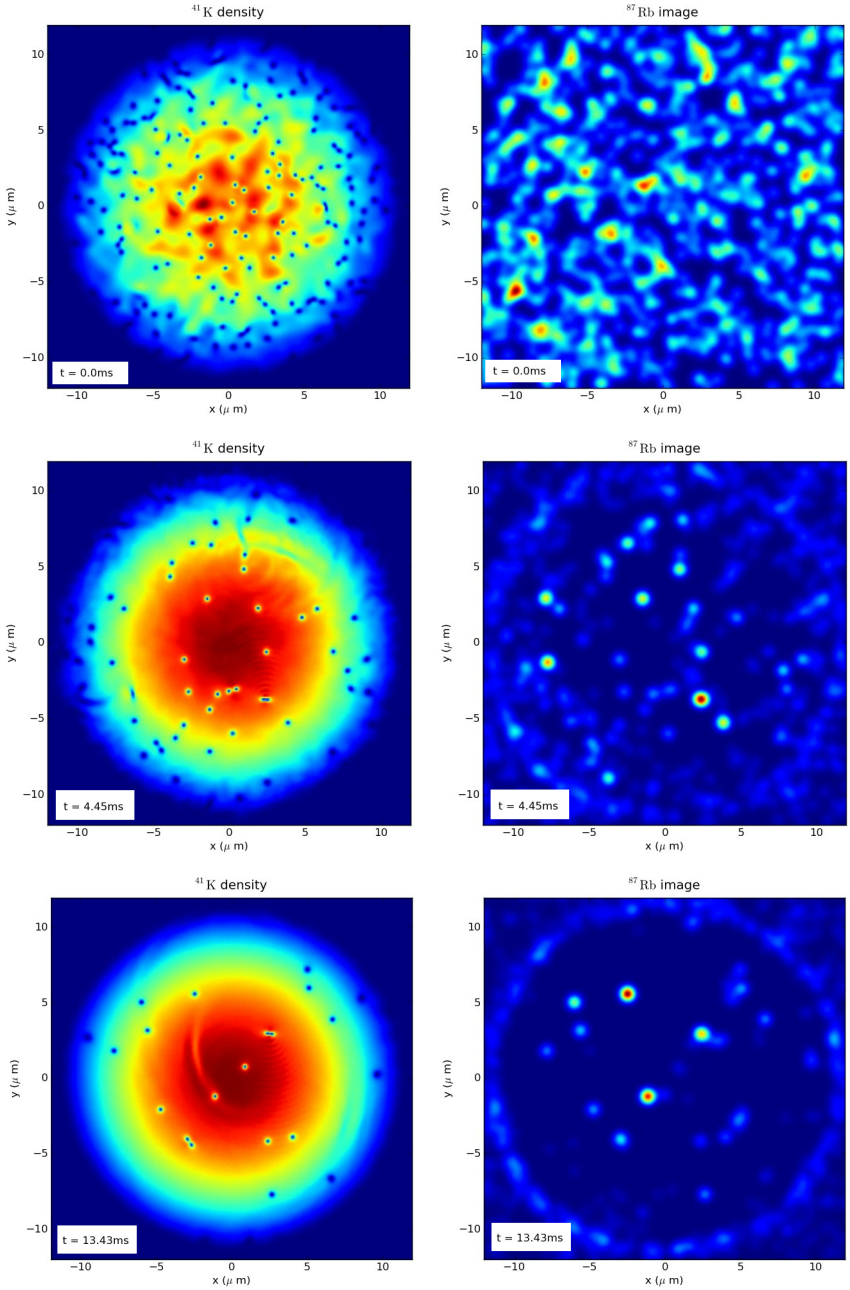

Figure 5.5: The result from [157] of a two-dimensional hybrid quantum-classical simulation for 1000 classically modelled rubidium atoms (right, depicted as fluorescence assuming diffraction through an NA = 0.5 imaging system) and a turbulent potassium BEC (left) of peak density $\approx 10^{15}$ cm^{-3}. The rubidium atoms are subject to a classical approximation of the force due to polarisation gradient cooling as described in [157]. Most rubidium atoms eventually either become bound to a vortex core or leave the condensate. Figure reproduced from [157].

Three-body losses tend to limit the lifetime of condensates at such a high density, and so the work in this chapter investigates ways to make particle velocimetry work in a less dense condensate through the use of a Feshbach resonance.

In Section 5.4 I consider a similar configuration, but with a more reasonable BEC density combined with an enhancement of the interspecies repulsion due to the Feshbach resonance. I investigate whether sympathetic cooling of the tracer atoms by the condensate may be enough to keep them trapped in the presence of imaging light. Then, in Section 5.5 I present simulation results of a new laser cooling scheme designed to work at the magnetic field strength required for the Feshbach resonance.

5.4 Sympathetic cooling

The simulation performed in my Honours thesis considered only polarisation gradient cooling counteracting the heating effect of the imaging light. In reality, collisions between tracer atoms and atoms in the condensate would also contribute to cooling of the rubidium. This sympathetic cooling of the tracer atoms—which would also lead to heating of the condensate—was disregarded in my Honours results.

Depending on the strength, sympathetic cooling may be sufficient to retain tracer atoms in vortex cores in the absence of an additional cooling mechanism. If laser cooling is not necessary to trap tracer atoms in vortices, then the Feshbach resonance may be used to enhance the interspecies scattering length, further enhancing the ability of the vortices to trap tracer atoms. In this section I consider a similar simulation to that in my Honours thesis, except with no laser cooling simulated, and with a model of sympathetic cooling taken into account, in order to examine this possibility.

5.4.1 Model

In this section I model sympathetic cooling as elastic two-body scattering between the rubidium tracer atoms and the potassium atoms in the condensate. The model is two-dimensional, appropriate for a pancake-shaped condensate. As with the simulation in my Honours thesis, I model the BEC with the damped Gross–Pitaevskii equation:

$$i\hbar\frac{\partial}{\partial t}\Psi_{\mathrm{K}}(\boldsymbol{r},t) = (1-i\gamma)\left[-\frac{\hbar^2}{2m_{\mathrm{K}}}\nabla^2 + V(\boldsymbol{r}) + g_{\mathrm{K}}|\Psi_{\mathrm{K}}(\boldsymbol{r},t)|^2\right]\Psi_{\mathrm{K}}(\boldsymbol{r},t), \tag{5.2}$$

with damping constant $\gamma = 0.01$ and all other symbols are as defined in Section 2.2, with the added subscripts K indicating quantities specific to ^{41}K. The ^{41}K s-wave scattering length is $a_{\mathrm{K}} = 121a_0$ [158], where a_0 is the Bohr radius. The damped GPE [159, 160] is a phenomenological modification of the standard GPE that includes dissipation to gradually remove higher energy excitations from the condensate wavefunction, approximating behaviour at finite-temperature. The condensate wavefunction is normalised at each timestep to compensate for the decay this model otherwise would cause. The potential $V(\boldsymbol{r})$ is a harmonic potential $V(\boldsymbol{r}) = \frac{1}{2}m_{\mathrm{K}}\omega_{\mathrm{K}}^2 r^2$.

The rubidium tracer atoms are modelled classically, evolving under the potential due to interspecies repulsion, as well as the external potential[2] $V(\boldsymbol{r})$, resulting in the equation of motion:

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2}\boldsymbol{r} = -\frac{1}{m_{\mathrm{Rb}}}\nabla\left(g_{\mathrm{Rb-K}}|\Psi_{\mathrm{K}}(\boldsymbol{r},t)|^2 + V(\boldsymbol{r})\right), \tag{5.3}$$

where $g_{\mathrm{Rb-K}}$ is the ^{87}Rb–^{41}K non-linear interaction constant $g_{\mathrm{Rb-K}} = \frac{2\pi\hbar^2 a_{\mathrm{Rb-K}}}{m_{\mathrm{r}}}$, with m_{r} the reduced mass of the ^{87}Rb–^{41}K scattering pair and $a_{\mathrm{Rb-K}}$ the s-wave interspecies scattering length. $a_{\mathrm{Rb-K}}$ is equal to $640a_0$ at zero magnetic field [38], and assumed in

[2] For simplicity I assume that both species are subject to the same external potential. Writing the potential as a harmonic trap for rubidium such that $V(\boldsymbol{r}) = \frac{1}{2}m_{\mathrm{K}}\omega_{\mathrm{K}}^2 r^2 = \frac{1}{2}m_{\mathrm{Rb}}\omega_{\mathrm{Rb}}^2 r^2$ gives $\omega_{\mathrm{Rb}} \approx 0.7\omega_{\mathrm{K}}$.

this section to be enhanced by a factor of five by means of the 34 G Feshbach resonance (see Section 2.1.6 and Figure 2.2). Since the Gross–Pitaevskii equation is solved on a grid whereas the classical motion of the atoms is modelled using continuous position variables, the condensate density is numerically differentiated and the results interpolated to the positions of the tracer atoms using cubic splines in order to evaluate (5.3) for each tracer atom.

The motion of the tracer atoms is punctuated by momentum jumps due to both the scattering of imaging photons and two-body collisions with the condensate. Photon scattering events are modelled as instantaneous momentum jumps of magnitude hc/λ where $\lambda = 780\,\mathrm{nm}$ is the wavelength of the imaging light. A random direction in 3D space is chosen and a momentum jump in this direction with the given magnitude is projected into the 2D plane of the simulation before being applied to the atom. These jumps occur at random times at an average rate given by the target photon scattering rate of 10^5 photons per second. Neglecting the exact details of the imaging light used, I assume repumping is included such that the atom spends nearly all of its time in the $|1,1\rangle$ ground state—necessary for the scattering length enhancement—and that the target scattering rate includes scattering from all lasers, repump or otherwise.

Collisions with the condensate are modelled as elastic collisions between a rubidium atom with the given classical velocity, and a potassium atom of velocity equal to the superfluid velocity $\boldsymbol{v}_\mathrm{K}$ of the condensate (discussed in Section 2.2) at the location of the tracer atom, given by

$$\boldsymbol{v}_\mathrm{K}(\boldsymbol{r},t) = \frac{\hbar}{m_\mathrm{K}}\nabla\phi_\mathrm{K}(\boldsymbol{r},t), \tag{5.4}$$

where $\phi_\mathrm{K}(\boldsymbol{r},t)$ is the complex phase of the potassium condensate wavefunction. The s-wave scattering length a is defined as [161, p 589, eq. 12.101]

$$a = -\lim_{k_\mathrm{rel}\to 0}\frac{\tan\left(\delta_0(k_\mathrm{rel})\right)}{k_\mathrm{rel}}, \tag{5.5}$$

where $k_\mathrm{rel} = m_\mathrm{r}v_\mathrm{rel}/\hbar$ is the relative wavenumber of the colliding pair of atoms with reduced mass m_r and relative velocity v_rel, and $\delta_0(k)$ the collisional phase shift. Assuming small k_rel and substituting this into the s-wave elastic scattering cross Section [161, p 584, eq. 12.66]

$$\sigma = \frac{4\pi}{k_\mathrm{rel}^2}\sin^2\left(\delta_0(k_\mathrm{rel})\right) \tag{5.6}$$

gives a low-velocity approximation to the scattering cross section for elastic collisions between the rubidium and potassium atoms:

$$\sigma_\mathrm{Rb-K} \approx 4\pi\frac{a_\mathrm{Rb-K}^2}{1+k_\mathrm{rel}^2a_\mathrm{Rb-K}^2}. \tag{5.7}$$

For rubidium atoms at 5 μK, $k_\mathrm{rel}a_\mathrm{Rb-K} < 0.1$, such that the zero-velocity scattering cross section $\sigma = 4\pi a^2$ would be accurate enough; nonetheless (5.7) is the expression used in this section.[3] From the scattering cross section we obtain the mean free path of a rubidium tracer particle within the potassium BEC:

$$\ell(\boldsymbol{r},t) = \left(\sigma_\mathrm{Rb-K}|\Psi_\mathrm{K}(\boldsymbol{r},t)|^2\right)^{-1}, \tag{5.8}$$

yielding the probability of a collision occurring in an infinitesimal time interval $\mathrm{d}t$:

$$P_\mathrm{collision}(\boldsymbol{r};t,t+\mathrm{d}t) = v_\mathrm{rel}(\boldsymbol{r},t)\sigma_\mathrm{Rb-K}|\Psi_\mathrm{K}(\boldsymbol{r},t)|^2\mathrm{d}t, \tag{5.9}$$

[3] For modestly larger Feshbach enhancements of the scattering length, the zero-velocity cross section would not be accurate and the velocity dependence of the scattering cross section would become important.

where $v_{\text{rel}} = |\boldsymbol{v}_{\text{K}}(\boldsymbol{r}, t) - \boldsymbol{v}|$ for a specific rubidium atom at position $\boldsymbol{r}$ and with velocity $\boldsymbol{v}$.

At each timestep, a random number between zero and one is generated for each atom, and if less than (5.9), a collision is taken to have occurred.[4] In the case of a collision, the 2D elastic scattering problem is solved and the rubidium atom's velocity instantaneously replaced with its post-collision value. The potassium condensate wavefunction is not modified—the sympathetic heating resulting from these collisions is ignored, which is a limitation of these results. For an ideal Bose gas of 10^6 potassium atoms at $T = T_{\text{c}}/2$, a heating rate of 10^5 rubidium recoil energies per second from each of 10^3 rubidium represents a rate of temperature increase of 25 nK ms^{-1}. This suggests that relying on sympathetic cooling alone may not allow for imaging of vortex motion for more than a few milliseconds (after which the condensate temperature will exceed the critical temperature), and that further cooling such as that presented in Section 5.5 may be necessary.

[4] During thesis writing, an error was discovered in the code that produced the results in this section, in which the computed collision probability (5.9) was too small by a factor of $\sqrt{2}$. As such, the results shown in the next subsection underestimate the sympathetic cooling effect slightly. I do not expect this error to change any of my conclusions.

5.4.2 Results

I simulated the equations described in the previous section in two dimensions, with the GPE solved on a 256×256 grid using fourth-order Runge–Kutta (Section 3.3) with $\Delta t = 500$ ns using fast Fourier transforms to evaluate the kinetic energy term (Section 3.4.1), and the tracer atom Newtonian equations of motion propagated also using fourth-order Runge–Kutta with the same timestep, for 10^3 tracer atoms. The first derivatives of the condensate wavefunction required to compute its phase gradient were evaluated using second-order finite differences, which I observed to be less susceptible to Gibbs' phenomenon in the vicinity of vortex cores, which—when using Fourier transforms—would produce a velocity field with unphysical radial motion close to a vortex core.[5]

[5] Interestingly, second derivatives—as required for the kinetic energy term of the GPE—do not appear to suffer from this problem at similar grid resolutions.

I repeated the simulation with two harmonic trapping frequencies in order to model two condensate densities. In the higher density simulations I used $\omega_{\text{K}} = 2\pi \times 130$ Hz and simulated a spatial region of 40 μm × 40 μm. In the lower density simulations I used $\omega_{\text{K}} = 2\pi \times 65$ Hz and increased the size of the simulated region to 57 μm × 57 μm.

I computed the initial conditions for the condensate wavefunction using the imaginary time evolution method (Section 3.5.1) subject to a fixed 2D normalisation constant leading to a peak density of 5.1×10^{14} cm^{-3} for the higher density simulation and 2.5×10^{14} cm^{-3} for the lower density simulation. Once the ground state condensate wavefunction was found, I then constructed a turbulent state by imposing a phase pattern on the condensate on a 16×16 grid, with the phase in each region chosen randomly from the interval $(-\pi, \pi)$. I then applied the imaginary time evolution algorithm once more for 600 μs to produce a physically realistic condensate wavefunction containing a number of vortices randomly distributed.

The initial positions of the tracer atoms were uniformly distributed over the entire spatial region, and velocities drawn from a Maxwell–Boltzmann distribution at 5 μK.

After producing the initial conditions, I then evolved the system in time for 32 ms with sympathetic cooling being modelled, but 'dark'—with zero photon scattering. The results of this are shown in Figure 5.6 for the higher density case and Figure 5.9 for the lower density. During this evolution many of the tracer atoms either moved into vortex cores or left the condensate (those that left mostly formed a ring at the Thomas–Fermi radius where the external potential and interspecies potential resulted in a potential minimum), showing that sympathetic cooling under the assumptions of this model is sufficient to cause randomly-distributed tracer atoms to become bound to vortex cores. During this evolution the vortices moved about the condensate, and due to the inclusion of damping, reduced in number over time as they left the condensate, or annihilated with each other.

I then continued this simulation, but with the inclusion of photon scattering as described in the previous section. For both the lower and higher density simulations, I used two timepoints in the dark simulations as initial conditions: 16 ms and 32 ms. This

was in order to examine whether the decay in vortex number over time improved the visibility of vortices in the resulting images, since there are fewer vortices at later times. Imaging was simulated for 10 ms, and the location of each photon emission recorded in order to accumulate a simulated image of the tracer particle locations over time. From this idealised image—which does not include sub-unity collection efficiency or diffraction—I derived a more realistic image assuming 4.2% collection efficiency, and diffraction modelled as a displacement in the location of each detected photon by a random distance drawn from a Gaussian distribution with a standard deviation of 0.32 μm. This collection efficiency and spot size are appropriate for a NA = 0.5 imaging system, assuming unit quantum efficiency of the camera sensor and a Gaussian approximation to the diffraction spot shape. No effort was made to include photons based on their emission direction.

The results are encouraging—vortex motion is clearly visible in the idealised images, and still quite visible in the more realistic images taking into account imperfect imaging. Contrary to expectations, at lower densities vortex traces are slightly more visible in the simulated images due to the slower motion of the vortices. Even though vortex potentials are only a few photon recoil energies in depth at the lower density, the simulated sympathetic cooling is effective at keeping the tracer atoms trapped within vortex cores. The modelled interaction between the tracer particles and the condensate causes the tracer particles to follow the local superfluid flow, including orbiting vortex cores prior to falling into them, and moving in circles within vortex cores once trapped.

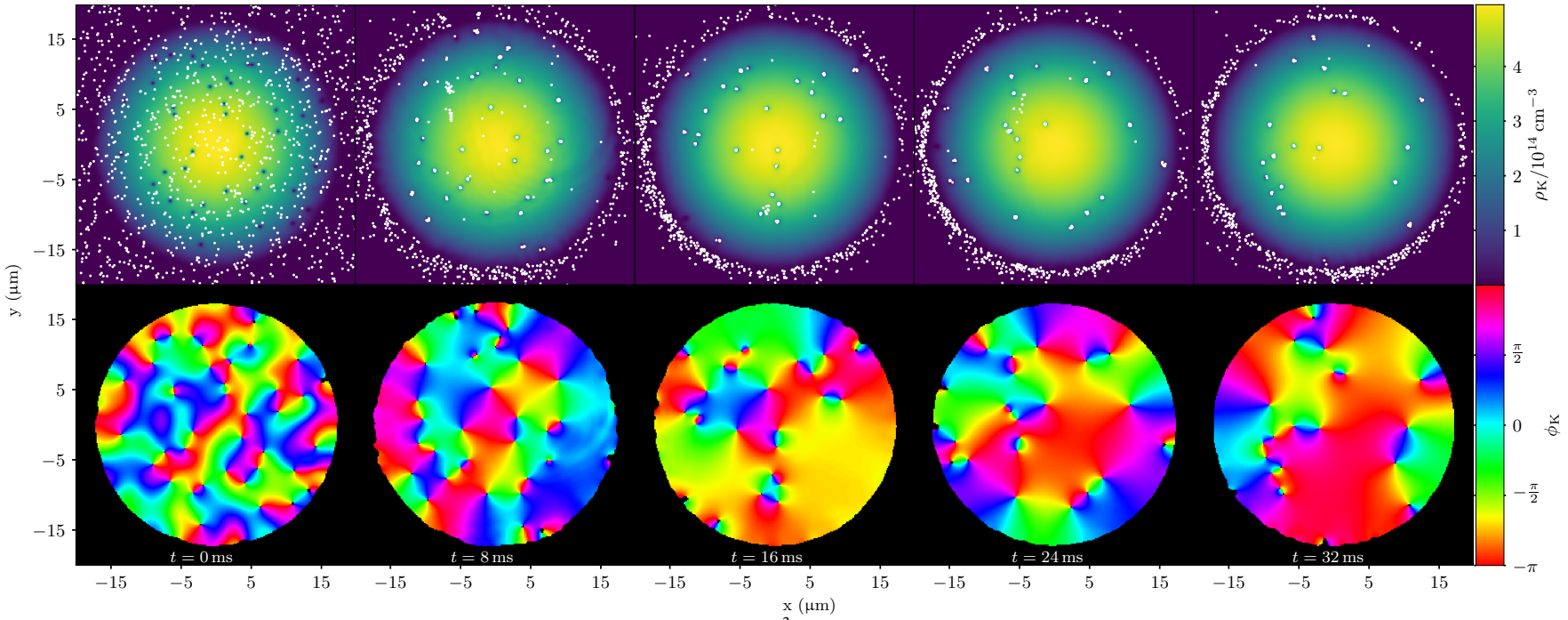

Figure 5.6: Higher density simulation of the evolution of the potassium condensate and 10^3 rubidium tracer particles interacting via sympathetic cooling, beginning from a turbulent condensate with randomly distributed tracer particles. Top row: condensate density (false colour) and tracer particle positions (white dots). Bottom row: condensate phase showing the location of vortices as points about which the phase winds by 2π. The initially randomly distributed tracer particles either leave the condensate or become trapped in vortex cores. When vortices annihilate, any tracer particles previously held remain in the condensate, this is an ongoing source of tracer particles in the bulk of the condensate that are not bound to vortex cores. Although not visible in these plots, in a video of these results the tracer particles can be seen to move in circles within vortex cores in the same direction of the superfluid flow.

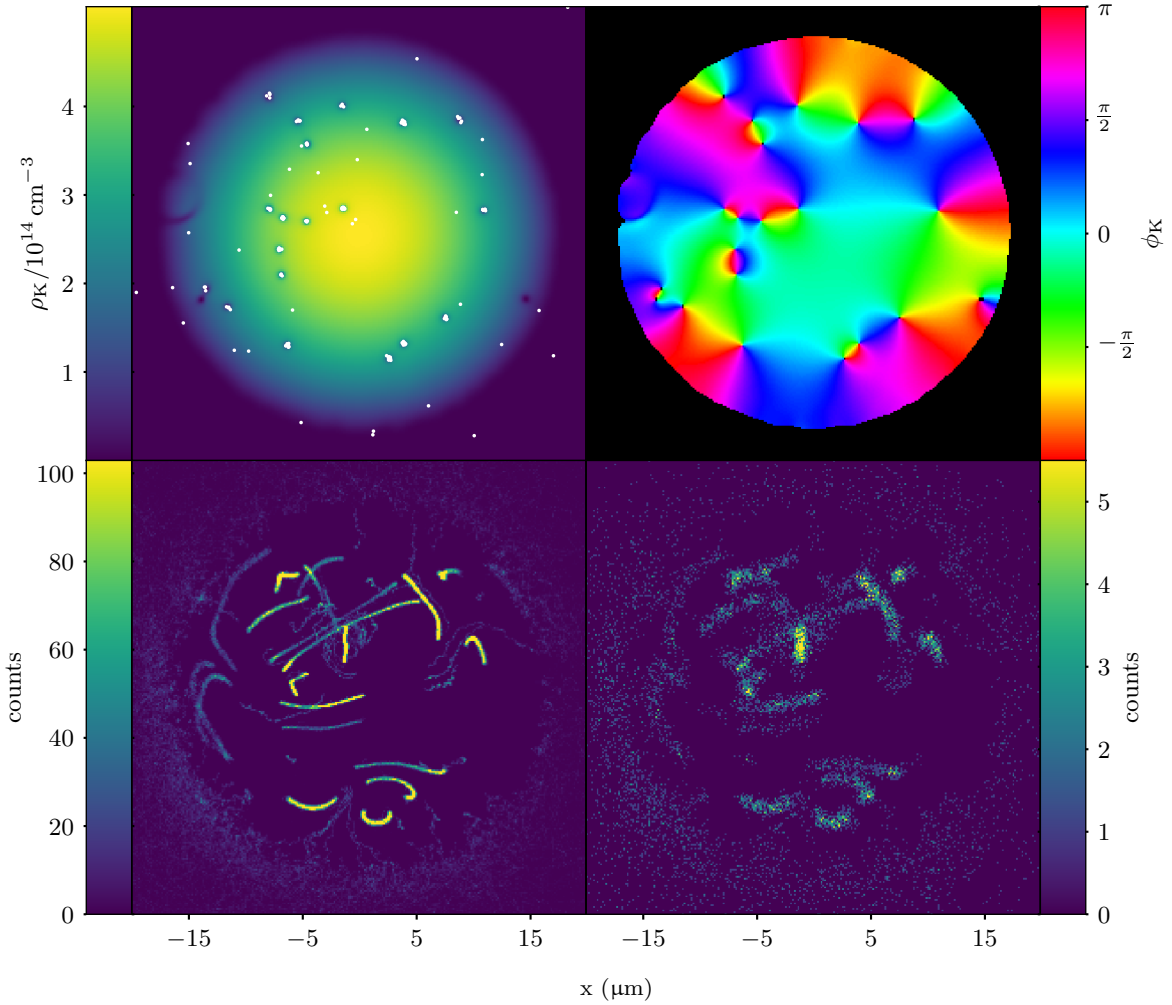

Figure 5.7: Final state of the higher density simulation of imaging of tracer particles in the BEC for 10 ms starting from the state of the 'dark' simulation at 16 ms (i.e. the central pair of Figure 5.6). Top left: The condensate density (false colour) and tracer particle positions (white dots). Tracer atoms that in the dark simulation formed a ring at the Thomas–Fermi radius are highly excited in the harmonic trap due to photon scattering, and leave the frame. Top right: condensate phase, showing the position of vortices as points about which phase winds by 2π. Bottom left: idealised image of tracer particle photon emissions over the imaging period. Photon emission locations are binned into a 256×256 grid. Complex vortex motion is clearly visible. Bottom right: modelled image taking into account non-unity collection efficiency and diffraction. Photon counts are low, but more than half the vortices are usefully tracked.

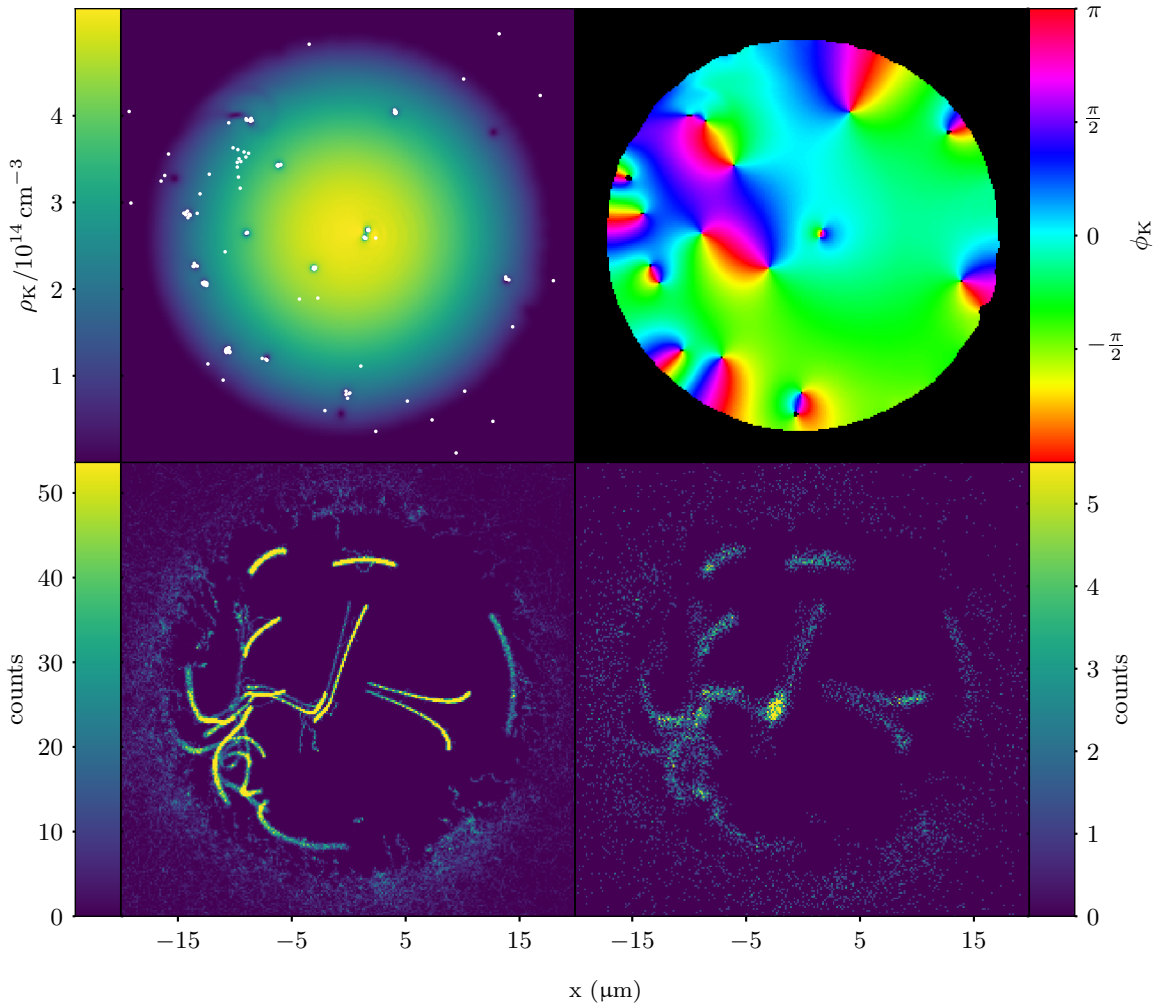

Figure 5.8: Final state of the higher density simulation of imaging of tracer particles in the BEC for 10 ms starting from the state of the 'dark' simulation at 32 ms (i.e. the rightmost pair of Figure 5.6). Top left: The condensate density (false colour) and tracer particle positions (white dots). Top right: condensate phase, showing the position of vortices as points about which phase winds by 2π. Bottom left: idealised image of tracer particle photon emissions over the imaging period. Photon emission locations are binned into a 256×256 grid. Vortex motion is visible, with slightly better contrast than in Figure 5.7, owing to fewer vortex annihilations taking place in this time interval, though the difference is minor. Bottom right: modelled image taking into account non-unity collection efficiency and diffraction. Photon counts are low, but vortex motion is still visible. Here the increased clarity of the vortex motion over Figure 5.7 is more apparent.

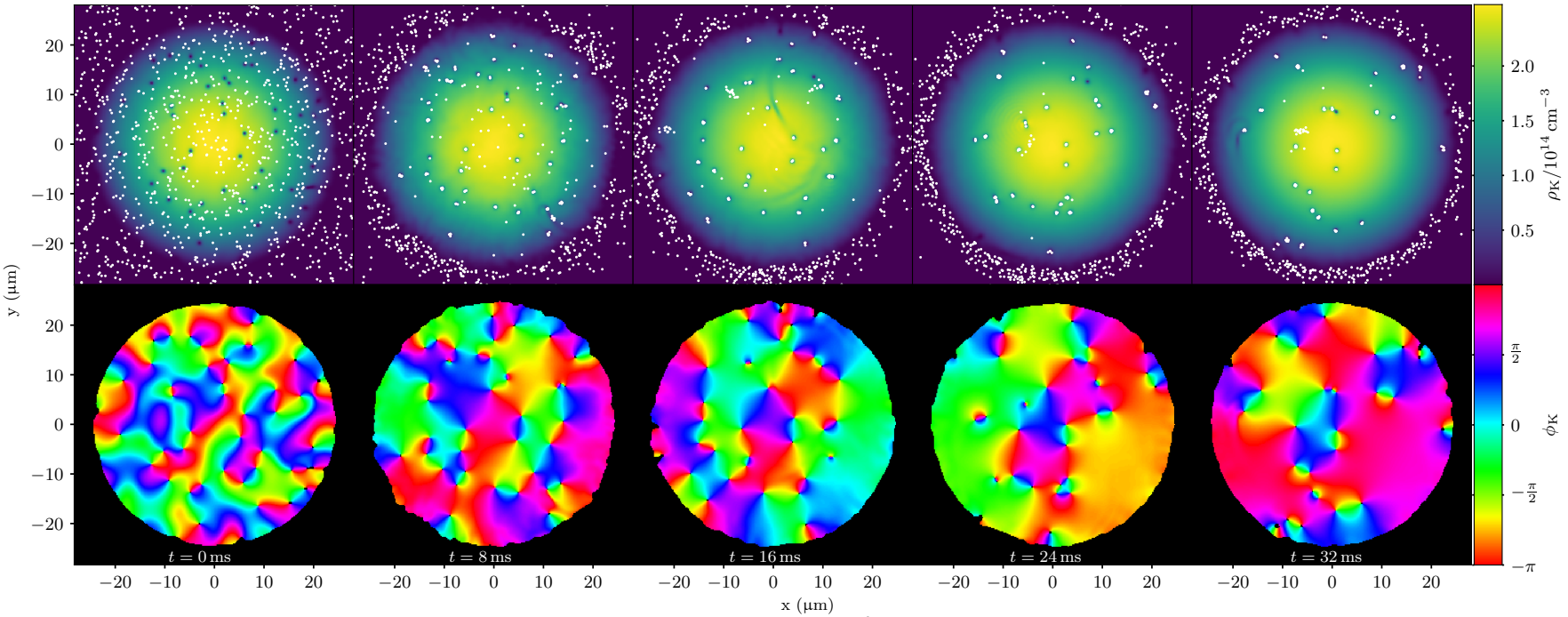

Figure 5.9: Lower density simulation of the evolution over time of the potassium condensate and 10^3 rubidium tracer particles interacting via sympathetic cooling, beginning from a turbulent condensate with randomly distributed tracer particles. Top row: condensate density (false colour) and tracer particle positions (white dots). Bottom row: condensate phase showing the location of vortices as points about which the phase winds by 2π. The initially randomly distributed tracer particles either leave the condensate or become trapped in vortex cores.

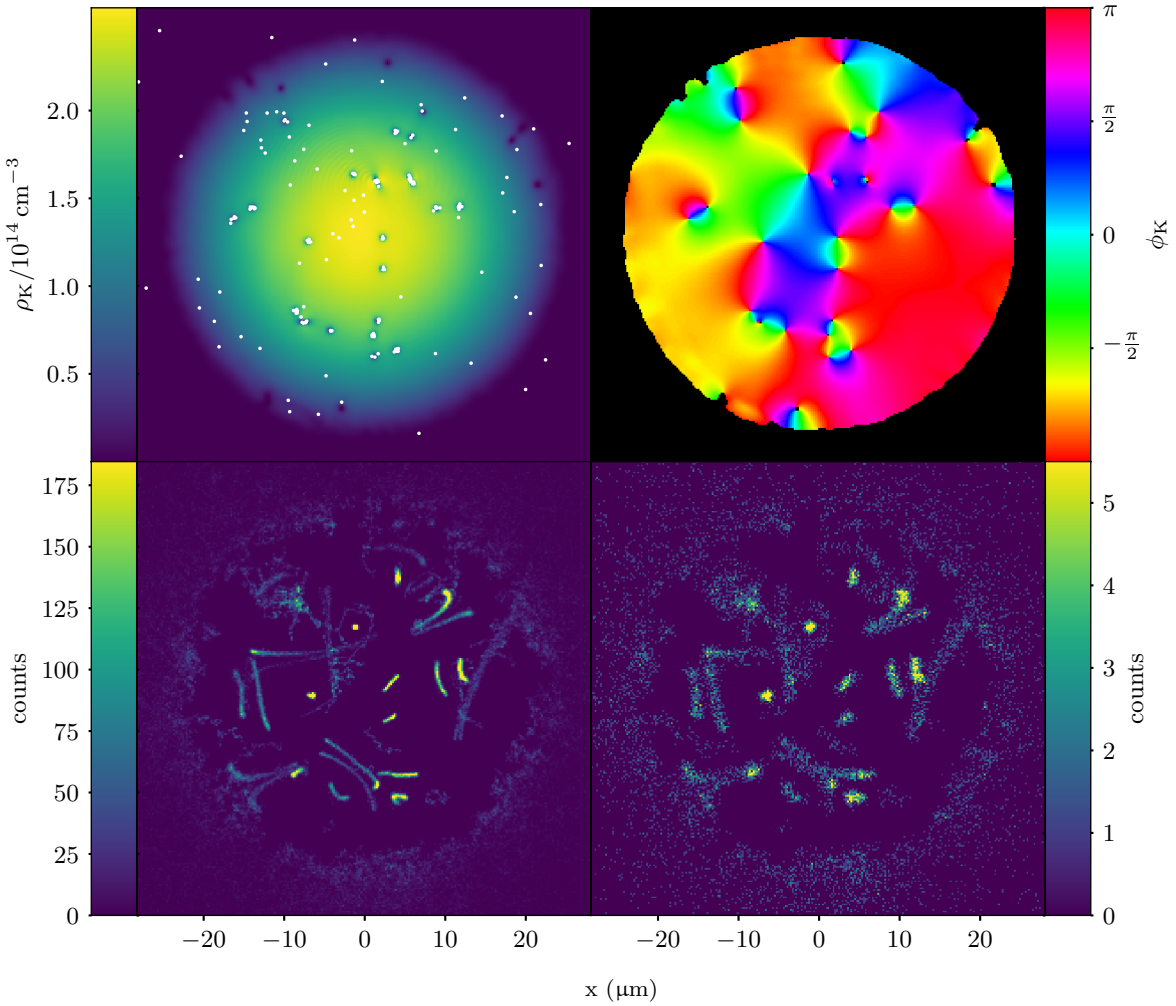

Figure 5.10: Final state of the lower density simulation of imaging of tracer particles in the BEC for 10 ms starting from the state of the 'dark' simulation at 16 ms (i.e. the central pair of Figure 5.9). Top left: The condensate density (false colour) and tracer particle positions (white dots). Top right: condensate phase, showing the position of vortices as points about which phase winds by 2π. Bottom left: idealised image of tracer particle photon emissions over the imaging period. Photon emission locations are binned into a 256×256 grid. Vortex tracks appear shorter than in the higher density simulations, owing to the slower vortex velocity at lower density. Tracer particles are clearly still able to remain trapped despite the shallower vortex potentials, and the slower vortex velocity improves contrast as more photons are emitted per unit distance of vortex motion. Bottom right: modelled image taking into account non-unity collection efficiency and diffraction. Photon counts are low, but vortex motion is still visible. The increased clarity of vortex tracks due to their slower motion is still apparent with the lower photon counts. As the lower condensate is larger, diffraction is slightly less apparent than in the higher density simulations.

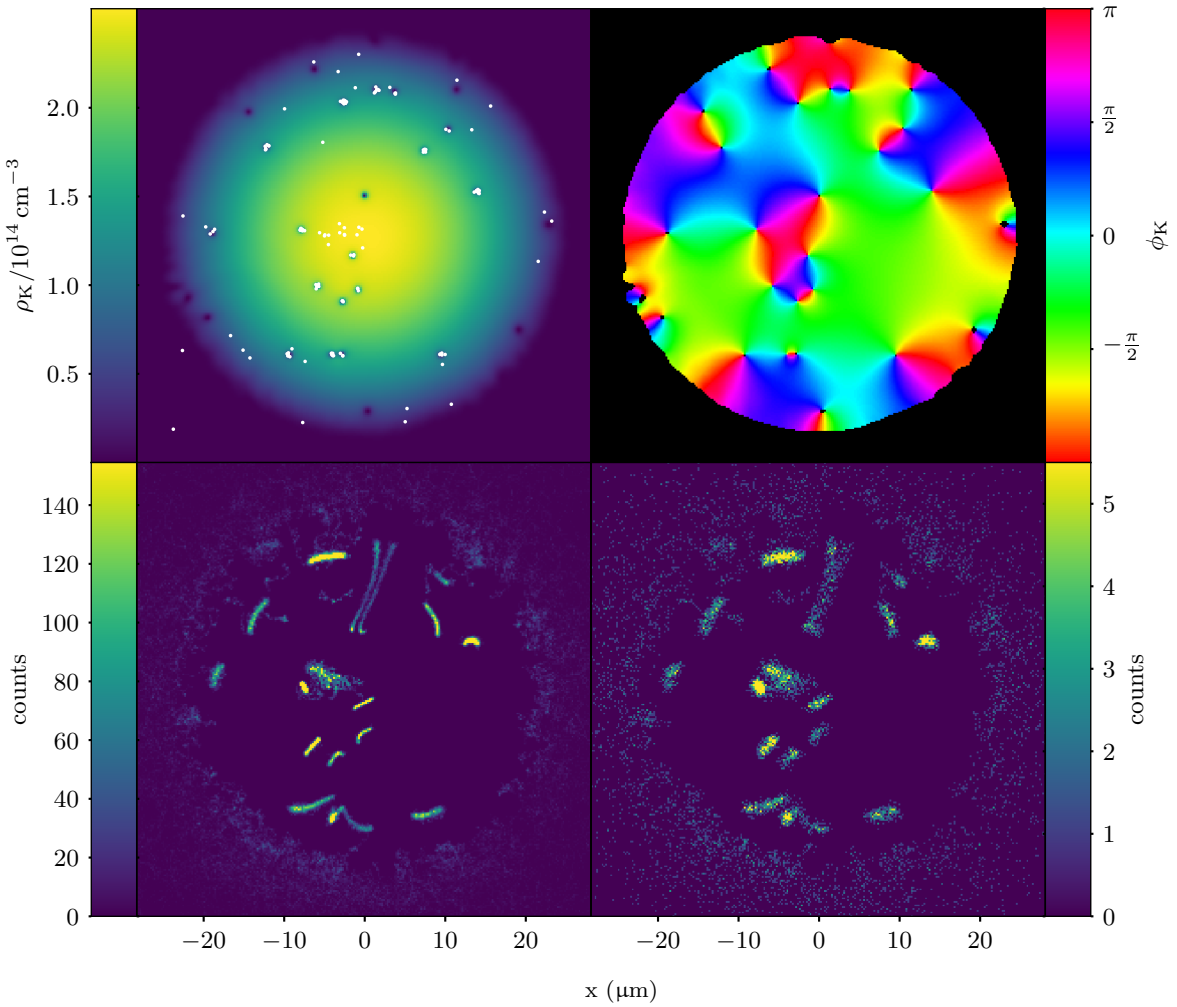

Figure 5.11: Final state of the lower density simulation of imaging of tracer particles in the BEC for 10 ms starting from the state of the 'dark' simulation at 32 ms (i.e. the rightmost pair of Figure 5.9). Top left: The condensate density and tracer particle positions (white dots). Top right: condensate phase, showing the position of vortices as points about which phase winds by 2π (phase only plotted where density is $> 5\%$ of maximum density). Bottom left: idealised image of tracer particle photon emissions over the imaging period. Photon emission locations are binned into a 256×256 grid. Vortex tracks are shorter than in the higher density simulations due to slower vortex motion, and there are fewer of them owing to more vortex annihilation occurring before the imaging light was turned on. Bottom right: modelled image taking into account non-unity collection efficiency and diffraction. This is perhaps the clearest image of vortex motion out of the four imaging simulations, with vortex trails most clearly visible and separated owing to their lower number and slower velocity.

5.5 Sisyphus cooling in a 34 G magnetic field

As mentioned, one of the limitations of the usual method of polarisation gradient cooling is that it doesn't work in a magnetic field larger than Γ/γ, where Γ is the transition line width and γ the gyromagnetic ratio. Fields larger than this cause some of the required transitions to shift out of resonance which destroys the cooling effect. Usually this is not an issue for the cooling stage used en-route to BEC; the magnetic field is simply temporarily switched off. Our imaging method would benefit from a cooling scheme that does work in a magnetic field, since the repulsive interactions between ^{87}Rb and ^{41}K can be greatly enhanced via a Feshbach resonance at 34 G [38]. This would make the potential wells that the rubidium atoms see deeper, trapping them more strongly. However if the magnetic field destroys the cooling mechanism then the atoms won't stay trapped for long. Even if sympathetic cooling is sufficient to image tracer particles trapped in vortices, the addition of a cooling scheme would increase the lifetime of the condensate on account of decreased sympathetic heating, and may allow a larger scattering rate of photons before the tracer atoms cease to be trapped.

The Feshbach resonance only occurs if both species are in their respective $|S_{1/2}, F = 1, m_F = 1\rangle$ states,[6] so a cooling mechanism in which the rubidium atoms spend a significant fraction of their time in this state is desirable.

In this section I present a sub-Doppler cooling scheme that is designed to cool ^{87}Rb in a 34 G magnetic field. The basic Sisyphus mechanism—of atoms moving alternately between spin states which see different potentials—is possible to find in many multi-level systems of sufficient complexity[7]; my cooling scheme uses a Sisyphus mechanism with four lasers to cool and repump ^{87}Rb atoms in a 34 G field, with the atoms spending approximately half their time in the $|S_{1/2}, 1, 1\rangle$ state.

In Section 5.5.4, I briefly describe another cooling scheme suggested by Prof. Helmerson, which uses the vortex cores themselves as the potential hills in a Sisyphus mechanism. I have not simulated this scheme to asses its viability; I mention it here because it is illustrative of the type of problem that is difficult to model semiclassically, and was one of the factors that led me to consider the use of hidden variables in semiclassical models, as discussed in Chapter 6.

[6] F is not a good quantum number in a non-zero magnetic field, so what we mean writing this is the state that one would get if starting in an F state and adiabatically turning on the magnetic field.

[7] And indeed, many other Sisyphus cooling mechanisms exists other than polarisation gradient cooling [24, p 116].

5.5.1 Description of cooling scheme

The scheme involves four lasers, two for cooling and two for repumping. For simplicity I will first focus on the cooling lasers only, depicted in Figure 5.12. Consider a rubidium atom at $z = 0$ and in the $|S_{1/2}, 1, 1\rangle$ hyperfine ground state. At this position the atom sees no light, as the intensity of the cooling laser labelled ω_1 is zero, and it is in the wrong state to be pumped by the ω_2 laser (which is not resonant with any transitions from the $|S_{1/2}, 1, 1\rangle$ ground state).

As the atom moves rightward however, it will have to climb the repulsive potential hill formed by the ω_1 laser. As it does so, its $|P_{1/2}, 1, 1\rangle$ excited state probability will increase, and along with it, the probability of spontaneous emission. Spontaneous emission will be most likely to occur near the top of the potential hill where the laser intensity—and hence the excited state probability—is greatest.

The most likely ground state for the atoms to decay to from the $|P_{1/2}, 1, 1\rangle$ excited state is the $|S_{1/2}, 2, 2\rangle$ ground state, and this is most likely to occur near $z = \frac{\lambda}{4}$. If this occurs, we now have an atom in the $|S_{1/2}, 2, 2\rangle$ ground state at $z = \frac{\lambda}{4}$, a situation similar to that in which it started. Again, out atom now sees no light, but which laser has zero intensity and which targets the wrong transition are swapped.

As our atom continues rightward, it now has to contend with the potential hill formed by the ω_2 laser, and is most likely to undergo spontaneous emission from the

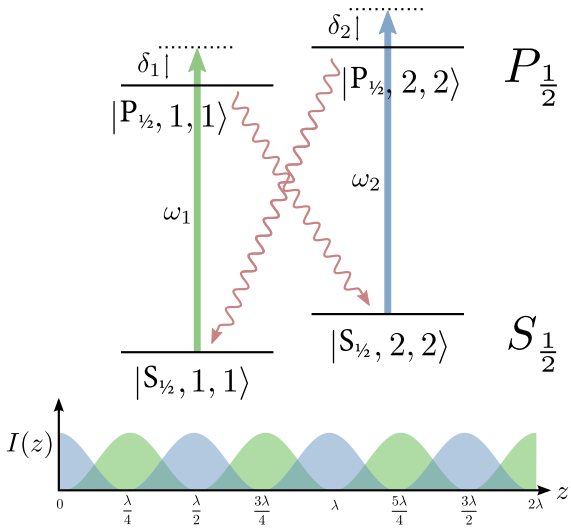

Figure 5.12: An idealised depiction of the cooling scheme, with repump lasers and undesired states not shown. Two lasers on the D_1 line are used for cooling, both linearly polarised, with the relative phases of all beams fixed so as to form two interleaved standing waves, with the nodes of one standing wave coinciding with the anti-nodes of the other. Both are blue detuned from the transitions they target, and they differ by about 6.8 GHz. This difference means that the alignment of the two standing waves can only be maintained over a distance of about a centimetre.

$|P_{1/2}, 2, 2\rangle$ excited state near the top of the potential hill. This time emission is most likely to put the atom into the $|S_{1/2}, 1, 1\rangle$ ground state.

This process repeats, with atoms repeatedly climbing potential hills and being cooled. They spend approximately half their time in the $|S_{1/2}, 1, 1\rangle$ ground state, allowing us to take advantage of the strong interspecies repulsion that this state entails for our two atomic species.

Of course, as is always the case, things aren't that simple. Whilst the two spontaneous decays mentioned above are the most likely, they are by no means the only possibilities. Some spontaneous decays will put the atoms back into the ground state from which they came, with no harm done except a little extra heating from the photon recoil. Other decays however will put our atom into states that are not involved in the cooling scheme, where they will remain with no further cooling unless we do something about it. For this we need repump lasers (Figure 5.13).

There are three states that the atom might end up in as a result of decay from the two excited states involved in the cooling process, and two repump lasers are used to excite them to three $P_{3/2}$ states,[8] from where they can spontaneously decay back to states in the cooling cycle. Two of these transitions are similar enough that they can be addressed with the same laser.

[8] I chose the $P_{3/2}$ manifold as the target of the repump transitions so that they would not interfere with the cooling cycle—repump beams that addressed the same $P_{1/2}$ states as the cooling beams would cause coherent population transfer into the undesired states.

5.5.2 Methods

The cooling scheme was simulated for the case of a single atom, with the internal state of the atom modelled with the Schrödinger equation in the eigenbasis of the hyperfine and Zeeman Hamiltonians (described in Section 2.3), comprising 32 states. The dipole transition matrix elements coupling each pair of states were each computed as the appropriate linear sum of the dipole matrix elements at zero field, using the dipole approximation and

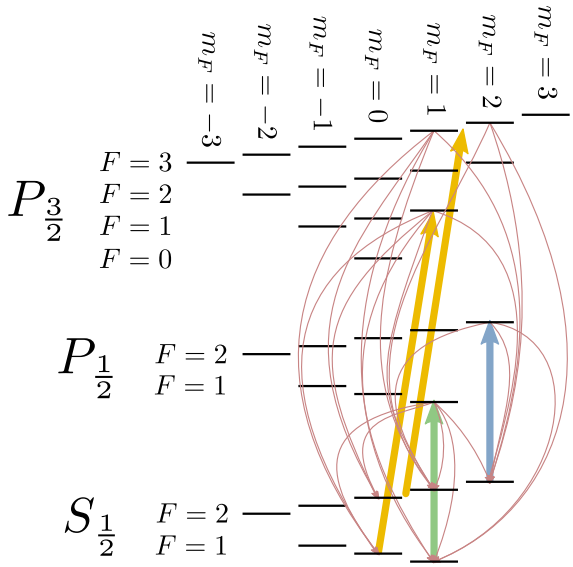

Figure 5.13: The full cooling scheme, including repump lasers (yellow), cooling lasers (blue and green), and all possible decay paths (red). The repump beam which is drawn in between two ground and excited states has a frequency equal to the average of those two transitions.

the rotating wave approximation, as described in Section 2.3.5. Since the corresponding Rabi frequencies depend on the laser intensity as well as the dipole matrix elements, they are functions of space, to be computed at each integration timestep as the atom moves through different intensities of the cooling beams. Following [24, p 4], this produced a set of 32 coupled differential equations for the amplitudes of each state, of the form :

$$i\hbar \frac{d}{dt} c_e(t) = -\frac{1}{2} e \sum_{g,\, n} E_n \langle g|q_n|e\rangle c_g(t) e^{-i\delta_{nge} t}, \tag{5.10}$$

and

$$i\hbar \frac{d}{dt} c_g(t) = -\frac{1}{2} e \sum_{e,\, n} E_n \langle g|q_n|e\rangle c_e(t) e^{i\delta_{nge} t}, \tag{5.11}$$

where each $c(t)$ is the complex amplitude of one state; the e indices are over the excited states and the g indices over the ground states; the n indices are over the lasers, with E_n being the n^{th} laser's electric field, δ_{nge} the detuning of the n^{th} laser from the $g \rightarrow e$ transition, and $\langle g|q_n|e\rangle$ the dipole moment between the g^{th} ground and e^{th} excited states for the polarisation q_n of the n^{th} laser.

The external motion of the atom was modelled classically, with the atom having a definite position and velocity in one dimension. The force on the atom was computed from the dipole forces that the two ground states involved in the cooling cycle experience due to the standing waves formed by the cooling beams. An expectation value of the dipole force was computed as:

$$\langle F \rangle = |\langle S_{1/2}, 1, 1|\Psi\rangle|^2 F_{|S_{1/2},1,1\rangle} + |\langle S_{1/2}, 2, 2|\Psi\rangle|^2 F_{|S_{1/2},2,2\rangle}, \tag{5.12}$$

where $F_{|S_{1/2},1,1\rangle}$ and $F_{|S_{1/2},2,2\rangle}$ are the dipole forces on the two ground states, calculated using [24, eqn 3.16, p 33] with one standing wave offset from the other by a quarter wavelength. The forces on other ground states are neglected since the repump beams do

Type	Transition(s)	Detuning	Intensity (per beam)	Polarisation
cooling (standing wave)	$\lvert S_{1/2}, 2, 2\rangle \rightarrow \lvert P_{1/2}, 2, 2\rangle$	+ 66.6 MHz	5.0 mW cm^{-2}	π
cooling (standing wave)	$\lvert S_{1/2}, 1, 1\rangle \rightarrow \lvert P_{1/2}, 1, 1\rangle$	+ 31.9 MHz	5.0 mW cm^{-2}	π
repump (single beam)	$\lvert S_{1/2}, 2, 1\rangle \rightarrow \lvert P_{\frac{3}{2}}, 2, 2\rangle$ $\lvert S_{1/2}, 2, 0\rangle \rightarrow \lvert P_{\frac{3}{2}}, 2, 1\rangle$	Midway between	50.0 mW cm^{-2}	σ^+
repump (single beam)	$\lvert S_{1/2}, 1, 0\rangle \rightarrow \lvert P_{\frac{3}{2}}, 1, 1\rangle$	0	10.0 mW cm^{-2}	σ^+

Table 5.1: The parameters used in the laser cooling simulations. There are four lasers, each with a specified polarisation, intensity, and detuning from the transition it targets.

not have intensity gradients, and since the cooling beams are much further away from resonance for ground states other than the two in the cooling cycle. Forces on excited states are also neglected. The forces on the two excited states in the cooling cycle are not smaller than those on the corresponding ground states, however the excited state populations are small since the cooling beams are several line widths away from resonance.

This expectation value of the dipole force was used to model the classical motion of the atom. Although the components of an atom in superposition would in reality spatially separate under the influence of a force that is different for different internal states of the atom, as in the Stern–Gerlach experiment, our atom only transitions between the two states subject to different forces via spontaneous emission, after which it is in an eigenstate. Accordingly, the expectation value calculation is always dominated by one of the two ground states and the potential for Stern–Gerlach separation does not arise.

Spontaneous emission was simulated stochastically at each integration timestep, with probability of decay per unit time equal to the sum of populations in all excited states, multiplied by their decay rates (equal to the natural line widths of each of the two fine-structure lines). Multiplying by the duration of one timestep, and comparing with a random number then determined whether a decay was to occur.

In the event of a decay, one excited state was randomly chosen with probability proportional to its population, and then one ground state, weighted by the transition strengths from the excited state. All population was then put into that ground state and the simulation continued, with one photon of momentum in a random 1D direction added to the atom's momentum to account for photon recoil.

The equations of motion were solved using fourth order Runge–Kutta integration (Section 3.3)

5.5.3 Results

The laser parameters used in the simulation are shown in Table 5.1. The magnetic field strength used was 34 G. The detunings of the cooling beams were chosen based on an approximate calculation in order to produce a scattering rate of 2×10^5 photons per second in the simulation.

The simulation was run for 7.2×10^6 integration timesteps of size $\Delta t = 20$ ps,[9] for a total of 14.3 milliseconds of simulation time. This took 14 days of computer time. In that time, the atom moved a maximum distance of 26 μm from its starting position, and its final position was 790 nm from its starting position. The atom's initial velocity

[9] Corresponding to approximately ten timesteps per oscillation of the fastest oscillating terms, which oscillate at a rate close to half the 6.8 GHz hyperfine splitting of the rubidium ground states.

was 195 mm s^{-1}, and during the simulation it reversed the direction of its velocity 2226 times; 4103 photons were emitted, for an average scattering rate of 2.87×10^5 photons per second—slightly higher than the target rate.

I computed the time-averaged kinetic energy of the atom over the whole simulation as:

$$\langle E_K \rangle = \frac{1}{2} m_{Rb} \langle v^2 \rangle. \tag{5.13}$$

Assuming the single atom ergodically sampled a sufficient fraction of the possible state space, one can compute from this the 1D temperature that a cloud of atoms subject to this cooling would have, after being allowed to come to equilibrium using

$$\frac{1}{2} k_B T_{1D} = \langle E_K \rangle . \tag{5.14}$$

This gave $T_{1D} = 16.2\,\mu K$, well below the Doppler temperature $T_D = 146\,\mu K$ [40]. A histogram showing the time the atoms spent at different velocities is shown in Figure 5.14. This one-dimensional temperature corresponds to $\approx 30E_r$ where E_r is the rubidium recoil energy. Given that the potassium vortex potentials are at about $15E_r$ deep without a Feshbach resonance at a density of 1×10^{15} cm^{-3} and only $\approx 2.0E_r$ at the more realistic density of 1×10^{14} cm^{-3}, and that the atom only spends about half its time in the state subject to the Feshbach resonance, this cooling would only be sufficient to keep rubidium atoms trapped in vortex cores if the Feshbach resonance enhances the interspecies scattering length by a factor of ≈ 2 for a 1×10^{15} cm^{-3} condensate or a factor of ≈ 30 for a 1×10^{14} cm^{-3} condensate. While a factor of 2 is easily achievable, a factor of 30 is not, indicating that the minimum density compatible with tracer particles at this temperature being trapped is between 1×10^{14} cm^{-3} and 1×10^{15} cm^{-3}.

This simulation has not, however, been optimised—the results presented here represent the only extended simulation of the scheme, and no attempt has been made to scan over parameter space to see if the temperature can be made lower. As mentioned the simulation was computationally expensive, and so optimisation would be impractical if performed using the same code. However, a significant speed up would be possible by correcting some inefficiencies. Firstly, one could exclude from the simulation the atomic states that were shown in the first run never to become occupied—the states with no arrows leading to them in Figure 5.13. This will eliminate approximately two thirds of the states, and since the simulation is quadratic in the number of states, this should provide an approximately 10× increase in simulation speed. More importantly, lasers that are detuned with respect to a given transition by approximately the hyperfine splitting of 6.8 GHz should be discarded in the coupling term for that transition. The inclusion of these fast rotating terms—which are so far detuned as to cause negligible population transfer—was the limiting factor in the simulation timestep size, which would otherwise be determined by the next fastest rotating terms on the order of tens of MHz instead of GHz. This would lead to a likely dramatic speedup as well, making repeated runs of the simulation feasible.

5.5.4 Vortex-assisted Sisyphus cooling

Another idea for a cooling scheme is to use the vortex potential itself as a spatial discriminator for transferring atoms between states. Similar to how a MOT traps atoms by bringing them into resonance with optical pumping only when they are some distance from the trap centre, one might use the shape of the vortex potential to bring an RF or microwave transition into resonance only when trapped tracer particles are some distance away from the centre of a vortex core. This method was proposed by Prof. Helmerson whilst considering different possibilities for cooling atoms in vortex cores, and I considered which states might be appropriate to implement the scheme in. I have not simulated

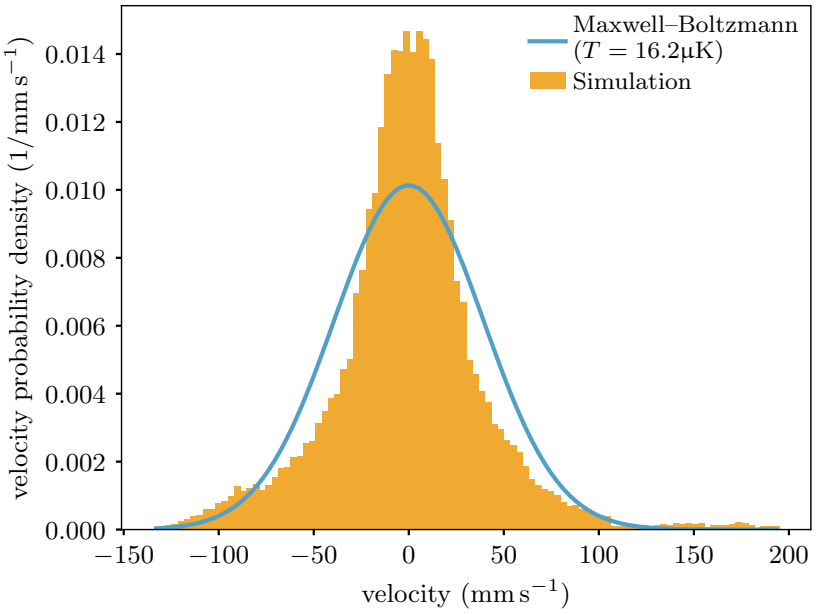

Figure 5.14: Histogram in orange of atom velocity, normalised as a probability density. The symmetry about zero velocity and low level of noise on the histogram is good evidence that the atom ergodically sampled the range of possible velocities such that the histogram can be interpreted as a probability distribution. The long tail visible to the right is the atom's initial slowdown from its starting velocity. A Maxwell–Boltzmann distribution with the same mean kinetic energy is shown as the blue line. It is no surprise that it does not agree with the histogram—there is no thermalisation happening since there is only one atom and no collisions—but this represents the distribution that a cloud of atoms with the same per-particle mean kinetic energy as our atom would thermalise to in the presence of collisions.

this scheme, but present it here because the reason it is difficult to simulate is an example of the type of problem that led me to develop the hidden variable semiclassical method presented in Chapter 6.

The basic idea of the vortex-assisted cooling scheme is outlined in Figure 5.15. In the presence of the Feshbach resonance, atoms in the $|S_{1/2}, 1, 1\rangle$ state scatter some tens of photons, using whichever transition is most likely to have them decay to the same ground state with minimal repumping (the $|P_{3/2}, 0, 0\rangle$ excited state on the D_2 line looks to be the best choice). As the atom scatters photons, it climbs the side of the vortex potential, converting the kinetic energy obtained from photon recoil into potential energy.

Due to the state-dependence of the interspecies scattering length, the vortex potentials for different states have different depths, especially when one is enhanced by a Feshbach resonance. This means that the RF or microwave frequency required to transition between the different hyperfine states and Zeeman sublevels varies as a function of space, and can be tuned so as to only be resonant with atoms which have nearly escaped the vortex core.

When the atom enters the region resonant with said RF or microwave radiation, it is then transferred into a different hyperfine or Zeeman state, for example the $|S_{1/2}, 2, 2\rangle$ ground state, and the hope is that it then lacks the kinetic energy to escape the (shallower) vortex potential it then finds itself in. Rather, it oscillates back and forth in the well until

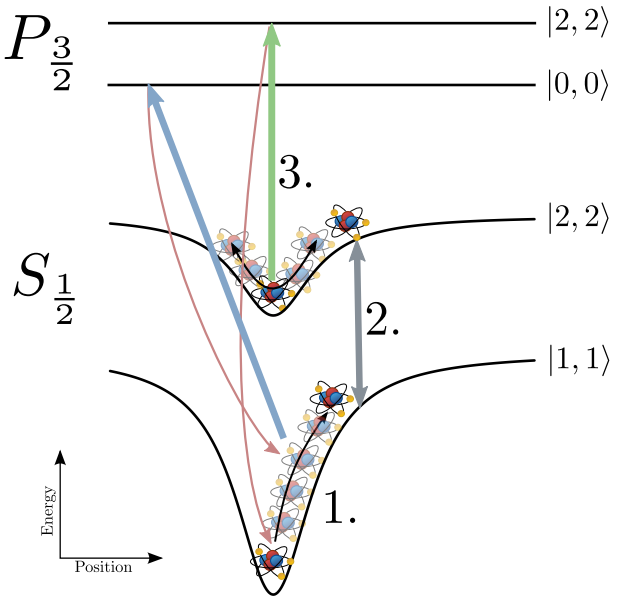

Figure 5.15: A basic description of the vortex-assisted cooling scheme. 1. The rubidium atom in its $|S_{1/2}, 1, 1\rangle$ ground state repeatedly scatters photons from the laser marked with the blue arrow, climbing the vortex potential as it does so. The optical transition's line width is large enough that the energy shift due to the vortex potential does not move it off resonance (and this allows us to ignore the vortex potentials experienced by the excited state, not shown). 2. An RF or microwave transition however, has an extremely narrow line width; its effective line width is dependent only on the RF/microwave power. A microwave transition (grey arrow) comes into resonance only when the atom moves sufficiently far from the vortex core's centre, and coherently transfers population into the $|S_{1/2}, 2, 2\rangle$ ground state. 3. The atom oscillates back and forth in the much shallower vortex potential that its $|S_{1/2}, 2, 2\rangle$ ground state experiences. It is pumped weakly by the laser marked with the green arrow, and after a random time delay (and hence at a random position) spontaneously decays back to the $|S_{1/2}, 1, 1\rangle$ ground state.

a weak laser pumps it back into the $|S_{1/2}, 1, 1\rangle$ ground state via spontaneous emission from some excited state (again chosen to maximise the decay probability to $|S_{1/2}, 1, 1\rangle$; the $|P_{3/2}, 2, 2\rangle$ excited state looks to be a good choice.)

After completing this cycle, statistically the atom will be closer to the centre of the $|S_{1/2}, 1, 1\rangle$ vortex potential than when it left the $|S_{1/2}, 1, 1\rangle$ state. Provided the corresponding drop in potential energy makes up for the photon scattering (which provides fluorescence imaging), then it comprises a cooling and imaging scheme capable of keeping the atoms trapped in the vortex cores. It is yet another Sisyphus effect, with the atom climbing steep vortex potential hills and descending shallower ones.

However, this scheme cannot be simulated in the same manner as the laser cooling scheme presented in Section 5.5. The reason is that the RF or microwave transition depicted as the grey arrow in Figure 5.15 is coherent, and as such leads to state vectors that are superpositions of the two hyperfine or Zeeman sublevels, with no one state dominating the superposition.[10] Since (crucially for the cooling scheme), the two states are subject to different potentials, an atom in such a superposition would undergo Stern–Gerlach separation. Unlike the laser cooling scheme from the previous section, since no one state dominates the superposition at a given time, the expectation value of the two forces does not accurately describe the motion of the atom. To accurately simulate this

[10] This is in contrast to the cooling scheme presented in Section 5.5, in which population was primarily on one of two ground states, and switched between them only via spontaneous emission

cooling scheme, a semiclassical method able to reproduce this Stern–Gerlach separation would likely be necessary. This realisation, along with similar difficulties in simulations of Majorana losses in evaporative cooling during collaboration with Drs. Turner and Anderson and Christopher Watkins led me to develop the hidden-variable semiclassical method, discussed in Chapter 6.

5.6 Conclusion

The sympathetic cooling results shown in Section 5.4 are promising, indicating that it is likely possible to observe vortex motion in BECs of realistic density via tracer particles using a Feshbach resonance and no additional cooling. A limitation of these results is their neglect of the corresponding sympathetic heating of the condensate, which could limit the duration over which vortices can be imaged. Furthermore, the assumption of solely elastic collisions becomes increasingly inaccurate as larger Feshbach-induced scattering enhancements are used due to an enhancement of inelastic collisions [162]. A final limitation is that in three dimensions, the halo of atoms that have left the condensate but are still trapped would also be in front of the condensate from the perspective of the imaging system, which would obscure the view of the tracer particles within the condensate somewhat.

The new laser cooling method presented in Section 5.5 appears—in simulation–to be able to provide sub-Doppler cooling at the magnetic field strength required for the ^{41}K–^{87}Rb Feshbach resonance. It is however an admittedly cumbersome method from an experimental perspective. It requires four lasers, driving transitions on both D lines from both hyperfine ground states. It requires relative phase control over the two cooling beams such that their standing waves are correctly aligned with the intensity troughs of one at the peaks of the other. These two beams are detuned from each other by the 6.8 GHz hyperfine splitting of the ground state, and so their wavelengths are different enough that this alignment can only be maintained over a distance on the order of 1 cm. Although the simulation was only in one spatial dimension, a similar arrangement of interlocking standing waves is possible in 2D to provide cooling in the two directions orthogonal to the magnetic field. However, since it is not possible for a beam with propagation direction parallel to the magnetic field to provide π–polarised light, one could not use the scheme in 3D for cooling in all three directions. Finally, although the cooling scheme did not appear to provide sufficient cooling for trapping in vortex cores, as mentioned it has not been optimised, and so may be capable of cooling to lower temperatures.

Chapter 6

Hidden variables for semiclassical models with state-dependent forces

Semiclassical models are approximate models of quantum systems, used when it is reasonable to approximate some degrees of freedom as classical, yet other degrees of freedom need to be modelled quantum mechanically. Often the internal state of an atom is modelled quantum mechanically, and its motion through space classically. Laser cooling is often modelled in this way [163–167] (and see also Section 5.5), as atoms are at high enough temperatures for their motional state to be well described by classical mechanics, and yet the evolution of the electronic state is undeniably quantum mechanical. This saves a great deal of computational cost compared to modelling a complete quantum system when a quantum model of the atomic motion adds nothing of interest not already captured by classical mechanics.

The classical part of a semiclassical model comprises Newton's second law—necessitating a known force. In some circumstances—such as the Stern–Gerlach experiment [19]—an atom is subject to a force that depends on its electronic state, which presents a problem for semiclassical models. Which force should be used? The lack of an answer to this question prevented me from simulating the proposed vortex-assisted cooling scheme discussed in Section 5.5.4, and an inadequate answer to it (namely, to use the expectation value of the force) produced unphysical results in simulations of evaporative cooling performed by Chris Watkins in the Monash Quantum Fluids group.

Hidden-variable theories [21,168] are interpretations of quantum mechanics that posit definite states underlying quantum state vectors, such that quantum indeterminacy is an illusion—an emergent phenomenon rather than a fundamental fact. Bell's theorem [169] posits that any such theory must be non-local in order to explain all the predictions of quantum mechanics, and perhaps in light of this, most physicists surveyed [170] do not believe that hidden variables underlie physical reality.

However, by framing quantum systems in classical terms, hidden-variable theories can provide an excellent computational tool for semiclassical models, and can resolve the aforementioned issue of state-dependent forces. Just as hidden-variable theories have framed the quantum world in terms that are agreeable to the classical view of the world in the minds of some interpreters of quantum mechanics, so can they bridge the gap between a simulated quantum world and a simulated classical world coexisting in the same computer simulation.

In this chapter I describe what I call the 'hidden-variable semiclassical' (HVSC) method: a method of combining quantum simulations with classical simulations, with hidden variables bridging the gap between the classical and quantum degrees of freedom. In this introduction I describe existing semiclassical methods, the manner in which their

quantum and classical parts are typically coupled based on expectation values, and in which regimes this can be inaccurate—namely the Stern–Gerlach experiment and similar situations in which considerable entanglement between motional and internal degrees of freedom of atoms can develop.

In Section 6.1 I give an overview of what a semiclassical method is, their most common implementation and in what situations this is insufficiently accurate, motivating the need for an improved method. In Section 6.2 I give the technical definition of a hidden-variable theory, and motivate the use of such a theory for coupling quantum and classical degrees of freedom in such a way that the a semiclassical model can be made to agree more closely than the Ehrenfest method with the underlying fully quantum model it is approximating, and ultimately, with experiment. In Section 6.3 I then discuss the implications of welcoming a hidden variable into a semiclassical model, including additional required assumptions and approximations, and in Sections 6.4 and 6.5 I derive the equations of motion for the model and present some algorithmic and computational details. Finally in Section 6.6, having provided all the background arguments and details, I present the complete algorithm(s), before showing simulation results in Section 6.7 that compare the model to the underlying exact Schrödinger wave equation, and concluding in Section 6.8 with further discussion of the method's benefits and limitations.

During writing this thesis, I discovered that the core idea underpinning this method is not original, and that the 'surface hopping' method enjoys widespread use and continued development in the field of computational chemical physics [20, 171–180]. An earlier version of my model [181] bears a striking resemblance to that presented in a 1991 paper by Tully [20], the pioneer of surface-hopping methods. However, this technique has not previously been applied in cold atom physics. Direct simulation of evaporative and laser cooling are becoming increasingly feasible due to increased computational power and efficient molecular dynamics techniques such as Direct Simulation Monte Carlo (DSMC) [182]. In light of this, the convergent evolution of numerical techniques is perhaps not surprising.

I did not encounter the surface-hopping literature earlier as I considered my approach to be under the same umbrella as methods such as the Monte Carlo wavefunction method [183] and quantum trajectories more generally [184, 185], which do not overlap with surface hopping in the literature. Throughout this chapter I make comparisons between my own methods and those in the existing surface-hopping literature.

Three long-standing limitations of my method may be resolved in light of the surface-hopping literature. One is that my model was previously limited to one dimension only, as I did not know in what direction atoms should gain or lose momentum upon making a transition. This is a known result in the surface-hopping literature, and discussed in Section 6.4.3, resolving this issue completely. The second limitation is that my model is limited to time-independent Hamiltonians. This is because time-dependent potentials can exchange energy with an atom, whereas part of my model relied on an energy conservation argument to compute the velocity of an atom after a transition. The surface-hopping literature uses an identical argument, and therefore suffers from the same problem—that energy conservation cannot be assumed for a time-dependent Hamiltonian. However, the hidden-variable theory primarily used for surface hopping—Tully's fewest-switches algorithm (Section 6.4.2)—can be formulated in a way that distinguishes between transitions caused by spatial variation of the Hamiltonian and those caused by its time variation. I present a way of computing these two sets of transition probabilities, and propose that the energy conservation only be imposed in the case of transitions 'caused' by the spatial variation of the Hamiltonian. However this is untested, and does not feature in the rest of the presentation of my model. Finally, the fewest-switches algorithm is computationally cheaper than the hidden-variable theory I have been using—Schrödinger theory, discussed in Section 6.2—resolving my concern over its computational expensive.

On the other hand, I have made contributions that are new. The identification of what

the surface-hopping literature calls 'hopping algorithms' with hidden-variable theories (the latter of which is usually discussed in the context of quantum foundations, philosophy and metaphysics rather than applied computational chemistry) has not previously been made. Furthermore, the computationally expensive Schrödinger theory—which is not new, though its use in surface-hopping models is—is more capable in certain circumstances: it can compute transition probabilities for arbitrarily long time intervals, given the unitary describing the corresponding quantum evolution over the time interval. Tully's fewest-switches, on the other hand, is valid for infinitesimal time-intervals only, and the probabilities need to be integrated over time to obtain transition probabilities over large intervals. It may be the case that in some scenarios Schrödinger theory is computationally cheaper than this integral. Finally, the more sophisticated of my two methods of computing decoherence, discussed in Sections 6.5.5 and 6.5.6, provides excellent agreement with the underlying Schrödinger wave equation it approximates, and is an improvement over similar methods in the surface hopping literature, which I will discuss at the end of this chapter (Section6.8).

6.1 Semiclassical models

A semiclassical model is one in which some degrees of freedom are treated quantum mechanically, and others classically. The most common combination is that of treating an atom's internal electronic state quantum mechanically and its motional degree of freedom classically. This is useful whenever the quantum effects of the atom's motion are not of interest, for example if temperatures are high and thus atomic wavelengths are short—such that quantum effects simply aren't visible in the motion of the particles and so they can accurately be modelled as classical billiard balls. The energy gaps between different electronic states of atoms are so large however that only at very high temperatures (at which atoms ionise anyway) do they start to appear as a continuum compared to thermal energy scales, and the interaction of different spin states of the atom with different optical and magnetic fields does not make them appear as classical continua either. Thus, quantum effects can be ignored for the centre of mass motion of the (relatively heavy) atom, but not for the relative motion of its (much lighter) electrons with respect to the nucleus, or for the nuclear and electronic spin degrees of freedom [20].

In this regime, atoms are often modelled semiclassically, with these internal degrees of freedom modelled using a state vector $|\chi\rangle$ evolving according to a Hamiltonian $\hat{H}$ via the Schrödinger equation, and the centre of mass motion modelled as a position $\boldsymbol{r}$ and velocity $\boldsymbol{v}$ evolving according to Newton's second law (Figure 6.1).

Once one has defined an external potential function $V(\boldsymbol{r})$ and a Hamiltonian (also possibly varying with space) $\hat{H}(\boldsymbol{r})$ for the internal state of the atom, and other possible additions,[1] ones job is done and the rest can be left to numerical differential equation solvers to evolve some concrete vector representation $\boldsymbol{\chi}$ of $|\chi\rangle$ as well as the state variables $\boldsymbol{r}$ and $\boldsymbol{v}$ for motional degree of freedom in time according to the coupled differential equations

$$\frac{\mathrm{d}}{\mathrm{d}t}|\chi\rangle = -\frac{\mathrm{i}}{\hbar}\hat{H}(\boldsymbol{r})|\chi\rangle\,, \tag{6.1}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{v} = -\frac{1}{m}\nabla V(\boldsymbol{r}), \tag{6.2}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{r} = \boldsymbol{v}. \tag{6.3}$$

This formulation has been widely successful in simulations of cooling, trapping, and manipulating cold atoms [163–167]. In the next subsection I explain why it's not always that simple.

[1] Such as using a Monte-Carlo wavefunction method [183,186] to model the effect of spontaneous emission on $|\chi\rangle$, and modifying $\boldsymbol{v}$ instantaneously by a random-direction recoil velocity upon each photon emission.

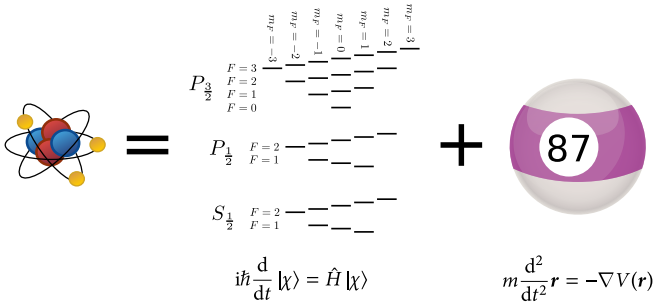

Figure 6.1: Artist's depiction of a semiclassical atom. Semiclassical models partition the atomic degrees of freedom into those to be modelled quantum mechanically with the Schrödinger equation (the electronic degrees of freedom), and those to be modelled with Newtonian mechanics (the motional degrees of freedom). This splits the system into subsystems, which may or may not interact. Some interactions, such as the state-dependent forces of the Stern–Gerlach experiment, are not obvious how to incorporate into such a model given the fundamentally different nature of the two subsystems. Hidden variables can bridge this gap.

6.1.1 Stern–Gerlach separation and evaporative cooling

In the Stern–Gerlach experiment [19], atoms—with quantum spin and a magnetic moment—were fired as a beam through a region of space with a magnetic field gradient. The well-known result was that two clusters of positions were observed once the beam emerges, rather than a continuous smear of positions, indicating that angular momentum projection—like many quantities in quantum mechanics—is quantised.

This is the case even if one spin-polarises the particles before they are passed through the magnetic field gradient, say putting them in an eigenstate of the $\hat{F}_x$ operator. Then, if the magnetic field is along the z direction, and the gradient is also in the z direction, two clusters of positions are also observed, even though all particles were in the same state when they entered the region in which there was a magnetic field gradient. This is a display of the indeterminacy of quantum mechanics: even though all particles had the same initial state, there were nonetheless different outcomes for each particle.

The outcome of the Stern–Gerlach experiment is a consequence of quantum mechanics, to be sure, but it has little to do with the wave nature of the atoms themselves. If we introduced some double slits for the atoms to pass through in addition to the magnetic field gradient, then we would be seeing the wave nature of the atoms as interference patterns at the detection screen at the end of the experiment. But if we do not, and if the particles have short de Broglie wavelengths, then quantum mechanics is not apparent in the motion of the particles through space—except via the influence of spin on seemingly choosing one trajectory or the other. The effect is well understood quantum mechanically, but is difficult to model semiclassically because even if we are happy to approximate wavepackets as small, the wavepackets do not take a single trajectory. Rather they split into two wavepackets, with the part of the superposition corresponding to one spin projection state (along the direction of the local magnetic field) moving one way, and the part of the superposition with the other spin projection going the other way. The trajectories can still be quite classical, it's just that there are two of them.

A similar situation exists in RF evaporative cooling (Section 2.1.5) of cold atoms en-route to BEC. A common configuration is a magnetic quadrupole trap, with atoms spin-polarised so as to be fully spin-down (for ^{87}Rb this is the trapped state) with respect to the local magnetic field at the position of each atom. The magnetic field direction—and magnitude—vary in space, and so the spin vectors of different atoms point in different

directions in space, but they are all spin-down with respect to the quantisation axis of their local magnetic field. As the atoms move through space, they move in orbits—punctuated by collisions—about the magnetic field zero at the centre of the trap, since they feel a force $F \propto -\nabla|\boldsymbol{B}|$ due to the gradient of the Zeeman potential. Provided they are moving slowly (specifically, provided their Larmor frequency is large compared to the rate of rotation of the magnetic field vector as seen by the atom), the atoms' spins adiabatically follow the local field and remain spin-down, even as the field as seen by each atom fully reverses its direction every half orbital period.

Near the centre of the trap where the atoms are moving faster, the fields are small and therefore have large fractional derivatives and lead to large Larmor periods, adiabaticity no longer holds and the atoms may make spin transitions with respect to their local magnetic field. Once an atom passing close to the field zero has evolved into a superposition of spin-projection states with respect to the local field, it is in a situation identical to the initial condition of the Stern–Gerlach experiment, causing the spin-projection components to spatially separate in the magnetic field gradient. The spin-up component is anti-trapped and repelled from the centre of the trap, and the zero spin-projection component (since the ground state of ^{87}Rb is spin-1) feels no force and moves in a straight line. The spin-down component continues on an orbit about the field zero that is just as tight as before, unaffected by the close approach to the field zero other than being reduced in amplitude. Eventually a collision occurs, either with other atoms or with the walls of the vacuum system and the wavefunction collapses to choose one of these options, leading to atoms probabilistically leaving the trap (called Majorana losses [187, 188]) or remaining trapped. Again, the trajectories can still be quite classical, it's just that there are three of them, and which trajectory is taken is probabilistic.

How can we model these effects semiclassically? Equations (6.1) to (6.3) are not sufficient, because exists no single classical potential $V(\boldsymbol{r})$ that can describe the motion of the atoms. Rather, the atoms feel a different force depending on which spin state they are in. Just as the Hamiltonian can be a function of space, so can the potential be a function of the internal state of the atom: $V = V(\boldsymbol{r}, |\chi\rangle)$. Ehrenfest's theorem [189] states that

$$m\frac{\mathrm{d}^2}{\mathrm{d}t^2}\langle\hat{\boldsymbol{r}}\rangle = -\langle\nabla\hat{V}\rangle\,, \tag{6.4}$$

where the expectation values are over all degrees of freedom, not just motional. If we approximate a small wavepacket centred at the position $\boldsymbol{r}$ such that $\langle\hat{\boldsymbol{r}}\rangle = \boldsymbol{r}$ in order to ignore the wave nature of the atoms, this becomes:

$$m\frac{\mathrm{d}^2}{\mathrm{d}t^2}\boldsymbol{r} = -\nabla\langle\chi|\hat{V}(\boldsymbol{r})|\chi\rangle\,, \tag{6.5}$$

where the operator $\hat{V}(\boldsymbol{r})$ now only acts on the subspace of the internal state of the atom, since we have already taken an expectation value over (a small region of) space. Provided all potentials the atom is subjected to are included in the Hamiltonian for its internal state (including any energy offsets that do not depend explicitly on the internal state), this is nothing but

$$m\frac{\mathrm{d}^2}{\mathrm{d}t^2}\boldsymbol{r} = -\nabla\langle\chi|\hat{H}(\boldsymbol{r})|\chi\rangle\,, \tag{6.6}$$

where $\hat{H}$ is the Hamiltonian describing the evolution of the atom's internal state. We now can construct the *Ehrenfest semiclassical method* describing how the *expectation value* of a well localised atom's position evolves with time:

$$\frac{\mathrm{d}}{\mathrm{d}t}|\chi\rangle = -\frac{\mathrm{i}}{\hbar}\hat{H}(\boldsymbol{r})|\chi\rangle\,, \tag{6.7}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{v} = -\frac{1}{m}\nabla\langle\chi|\hat{H}(\boldsymbol{r})|\chi\rangle\,, \tag{6.8}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{r} = \boldsymbol{v}. \tag{6.9}$$

The Ehrenfest semiclassical method is the same as the simple semiclassical method (6.1) to (6.3), except that it has an answer to the question "What should we use for $V(\boldsymbol{r})$ when the atom is in a superposition of states that feel different potentials?", which is "use the expectation value".

This is all well and good if the expectation value of position is a good approximation to the situation being modelled. But in the Stern–Gerlach experiment or a Majorana spin flip in a magnetic trap, the expectation value of position is a poor match to reality. In the Stern–Gerlach experiment beginning with spin-polarised atoms, a trajectory subject to the mean force or potential (which are both zero for a 50 : 50 superposition) would land in a single blob in the middle of the screen, rather than two blobs displaced from the centre. In the case of an atom approaching the field zero in a magnetic trap, use of the mean force would result in the atom broadening its orbit somewhat, rather than splitting into multiple possible trajectories (Figure 6.2). Semiclassical simulations of evaporative cooling performed by Christopher Watkins (unpublished) displayed an unphysical heating of the atom cloud that I believe is due to the Ehrenfest method's inability to model Stern–Gerlach separation. In a real magnetic trap during evaporative cooling to Bose–Einstein condensation, the mean free path is large enough that the part of the wavepackets that are no longer trapped will usually leave the trap without colliding with any other atoms. This means that the energy the untrapped and anti-trapped components have gained (relative to the trapped component) moving away from the field zero is not usually shared with other atoms upon collision—the extra energy leaves with the atoms. However, if a close approach to the magnetic field zero merely means a broadening of the atoms orbit, then the extra energy does not leave as fast, if at all, and can be shared with other atoms via collisions, turning what would have been an atom loss effect into an overall gradual heating of the cloud.[2]

[2] This is distinct from the real heating that occurs due to relatively lower energy atoms being more susceptible to Majorana spin flips and thus more likely to be lost, increasing the average energy of atoms remaining—a form of 'evaporative heating'.

So how can we modify a semiclassical method to choose only one trajectory? Firstly, since each trajectory corresponds to one of the internal states (though some might be degenerate), our model must choose an internal state and use the classical trajectory corresponding to that internal state only. Secondly, since all atoms begin in identical states and yet some take one trajectory and some another, this choice must be probabilistic. Finally, the probabilities must be consistent with those from quantum mechanics, i.e. the Born rule [190]: the probability of an atom taking each trajectory must be proportional to the squared amplitude of the internal state of the atom, projected onto the eigenstate corresponding to that trajectory.

There exists a category of theories dealing with precisely this question of how to choose a specific state of a quantum system in a stochastic way, such that the probability of having chosen a state is equal to that given by quantum mechanics. Such theories are called *hidden-variable* theories, and any parameter, variable or label specifying which state has been chosen is a *hidden variable*.

6.2 Hidden-variable theories

In his paper *Quantum computing and dynamical quantum models*, Aaronson defines a *dynamical quantum model* [191]:

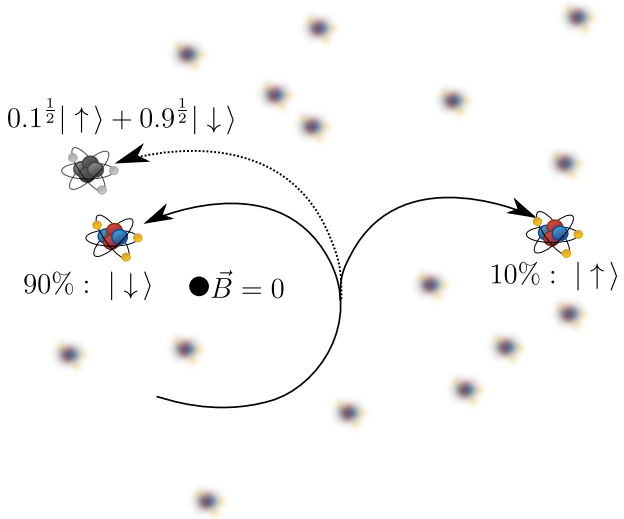

Figure 6.2: When atoms pass near to the field zero in a magnetic trap, their wavepackets diverge in space as multiple trajectories, each corresponding to one local spin projection state. For example, if a spin-$\frac{1}{2}$ atom undergoes a partial spin-flip such that it has a ten percent probability of being spin-up, the end result will be two approximate trajectories. One trajectory describes the motion of a wavepacket that is fully spin-up, and one fully spin-down, with the spin-down wavepacket's squared amplitude equal to 0.1. The Ehrenfest semiclassical method however can only model a single trajectory, and the end result when using this method is a single trajectory being approximately the mean of the two actual trajectories, and retaining a 90 : 10 ratio of spin-down:spin-up state populations.

> A *dynamical quantum model* assigns an eigenstate to a specified observable even when no measurement is made, and gives a stochastic evolution rule for that eigenstate. Such a model yields a distribution of classical histories of a quantum state.

In a later paper, *Quantum computing and hidden variables*, superseding the first, Aaronson renames dynamical quantum models to *hidden-variable theories* and refines the definition [21]:

> For us, a hidden-variable theory is simply a way to convert a unitary matrix that maps one quantum state to another, into a stochastic matrix that maps the initial probability distribution to the final one in some fixed basis.

I adopt this definition of a hidden-variable theory for the purposes of this thesis.

The language of the first definition is more tangible for us—we wish to *assign an eigenstate* to our atoms (choose one of the internal states in order to decide which trajectory to follow), and have a *stochastic evolution rule* for which eigenstate is chosen at any one time (allow the atom to begin taking a different trajectory if it makes transitions between states), probabilistically depending on the change in quantum populations of the states. But the second definition is more specific: the stochastic evolution rule is in the form of a stochastic matrix, with elements equal to transition probabilities for some time interval. And the rule should be in some way based on the unitary evolution that the quantum system evolves according to in the same interval of time, such that the initial

and final probabilities of the stochastic matrix and unitary evolution agree. That is, if a quantum state $|\chi\rangle$ evolves in a certain basis $\{|\chi_i\rangle\}$according to a unitary $\hat{U}(t',t)$:

$$\begin{bmatrix} c_1(t') \\ c_2(t') \\ c_3(t') \\ \vdots \end{bmatrix} = \begin{bmatrix} U_{11}(t',t)\ U_{12}(t',t)\ U_{13}(t',t)\ \ldots \\ U_{21}(t',t)\ U_{22}(t',t)\ U_{23}(t',t)\ \ldots \\ U_{31}(t',t)\ U_{32}(t',t)\ U_{33}(t',t)\ \ldots \\ \vdots \qquad\quad \vdots \qquad\quad \vdots \qquad\quad \ddots \end{bmatrix} \begin{bmatrix} c_1(t) \\ c_2(t) \\ c_3(t) \\ \vdots \end{bmatrix}, \tag{6.10}$$

where $c_i = \langle\chi_i|\chi\rangle$ and $U_{ij}(t',t) = \langle\chi_i|\hat{U}(t',t)|\chi_j\rangle$, then a hidden-variable theory is a matrix-valued function $S(U(t',t),\boldsymbol{\chi}(t)))$, where $\boldsymbol{\chi}(t)$ and $U(t',t)$ are the vector and matrix representations of the state vector $|\chi(t)\rangle$ and unitary $\hat{U}(t',t)$ in the $\{|\chi_i\rangle\}$ basis, that satisfies[3]

$$\begin{bmatrix} |c_1(t')|^2 \\ |c_2(t')|^2 \\ |c_3(t')|^2 \\ \vdots \end{bmatrix} = \begin{bmatrix} S_{11}(t',t)\ S_{12}(t',t)\ S_{13}(t',t)\ \ldots \\ S_{21}(t',t)\ S_{22}(t',t)\ S_{23}(t',t)\ \ldots \\ S_{31}(t',t)\ S_{32}(t',t)\ S_{33}(t',t)\ \ldots \\ \vdots \qquad\quad \vdots \qquad\quad \vdots \qquad\quad \ddots \end{bmatrix} \begin{bmatrix} |c_1(t)|^2 \\ |c_2(t)|^2 \\ |c_2(t)|^2 \\ \vdots \end{bmatrix}. \tag{6.11}$$

[3] In order to more clearly compare to quantum evolution, we are using *left stochastic* matrices, which multiply column vectors of probabilities from the left, contrary to the most common convention for stochastic matrices, which is to multiply row matrices from the right. Thus S has unit column sums, whereas the corresponding right stochastic matrix (its transpose) would have unit row sums.

In essence, if quantum unitary evolution maps complex state amplitudes to complex state amplitudes, a hidden-variable theory maps probabilities to probabilities. Thus the elements of S can be used as conditional probabilities—or transition probabilities—for a hidden variable $\eta(t)$: a piecewise-constant function of time with values equal to a state index, label, or quantum number that uniquely identifies one eigenstate at each moment in time. $\eta(t)$ evolves stochastically, with the elements of S giving the chance that the state $|\chi_\eta\rangle \in \{|\chi_i\rangle\}$ assigned by the hidden variable will change from one to another in a given time interval:

$$\Pr(\eta(t')=i|\eta(t)=j) = S_{ij}(U(t',t),\boldsymbol{\chi}(t)). \tag{6.12}$$

We are interested in such hidden variable theories because they can wrap a layer of classical interpretation around quantum evolution. In a semiclassical model, the internal state of an atom is modelled quantum mechanically, such that $\boldsymbol{\chi}(t)$ and $U(t',t)$ are known at every timestep. A stochastic hidden-variable theory can take these quantities—a description of the quantum evolution at each timestep—and give us back transition probabilities that allow us to evolve a hidden variable $\eta(t)$. This hidden variable's value at any moment in time will be equal to the label, index or quantum number of one state in the chosen basis, with probability equal to that state's population. Since each eigenstate of the Hamiltonian is subject to a well-defined adiabatic potential, the hidden variable can be used to connect the quantum evolution of the atom's internal state to the requirement that the classical motion use a single, well-defined force at each moment. And it can do so in a way consistent with the quantum probabilities of the internal state.

There are many ways to define functions S that satisfy the condition (6.12). The simplest is to ignore the unitary completely and set $S_{ij} = |c_i(t')|^2$, which yields:

$$\Pr(\eta(t')=i|\eta(t)=j) = |c_i(t')|^2, \tag{6.13}$$

that is that the hidden variable η is equally likely to transition to a given value regardless of its previous value, and regardless of the unitary, and that the conditional transition probability from all input states is just the squared amplitude of the final state. This theory represents a hidden variable that jumps between states randomly based on their population, with no regard for its history or whether there were actually amplitude flows between the states between which it is transitioning. Nonetheless, this theory, called *product theory* [21], matches the definition of a hidden-variable theory—S is a stochastic matrix, and the hidden variable on average will spend an amount of time in each state consistent with the Born rule.

In order to further classify hidden-variable theories with respect to whether they have this or other kinds of strange behaviour, Aaronson outlines some additional axioms [21] that hidden variables ought to satisfy—in addition to reproducing the Born rule—including reasonable statements about symmetries, and insensitivity to small perturbations. He goes on to prove that the axioms cannot all be satisfied simultaneously—all hidden-variable theories have some undesirable property or another—but he shows some theories satisfy more axioms than others.

It is convenient to introduce the matrix P of absolute (unconditional) transition probabilities.[4] Summing (6.12) over all possible initial values of the hidden variable yields the probability of it having a particular final value after the given time interval:

$$\Pr(\eta(t')=i) = \sum_j \Pr(\eta(t')=i|\eta(t)=j)\Pr(\eta(t)=j), \tag{6.14}$$

which, recognising that the final and initial probabilities must be the final and initial squared amplitudes of the state vector, can be rewritten:

$$|c_i(t')|^2 = \sum_j S_{ij}(U(t',t),\boldsymbol{\chi}(t))|c_j(t)|^2. \tag{6.15}$$

We now define the matrix of absolute transition probabilities P:

$$P_{ij} = S_{ij}(U(t',t),\boldsymbol{\chi}(t))|c_j(t)|^2, \tag{6.16}$$

such that:

$$|c_i(t')|^2 = \sum_j P_{ij}. \tag{6.17}$$

So we have that P has row sums equal to the final squared amplitudes. And, because $P_{ij} = S_{ij}|c_j(t)|^2$ and S is a stochastic matrix with column sums equal to one, we have that P must have row sums equal to the initial squared amplitudes.

Now that we have introduced P and shown what its row and column sums must be, we come to a particularly simply defined hidden-variable theory called Schrödinger theory,[5] discussed in Aaronson's hidden variables paper [21]. The idea is to form P by starting with the matrix of absolute values of U, and simply scaling its rows and columns to have the correct values:[6]

$$P = \begin{bmatrix} a_1b_1|U_{11}(t',t)| & a_1b_2|U_{12}(t',t)| & a_1b_3|U_{13}(t',t)| & \dots \\ a_2b_1|U_{21}(t',t)| & a_2b_2|U_{22}(t',t)| & a_2b_3|U_{23}(t',t)| & \dots \\ a_3b_1|U_{31}(t',t)| & a_3b_2|U_{32}(t',t)| & a_3b_3|U_{33}(t',t)| & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}, \tag{6.18}$$

that is,

$$P_{ij} = a_ib_j|U_{ij}(t',t)|, \tag{6.19}$$

where the row scalings $\{a_i\}$ and column scalings $\{b_j\}$ satisfy:

$$\sum_j a_ib_j|U_{ij}(t',t)| = |c_i(t')|^2, \tag{6.20}$$

$$\sum_i a_ib_j|U_{ij}(t',t)| = |c_j(t)|^2, \tag{6.21}$$

which can be solved numerically, and then the Schrödinger theory stochastic matrix then able to be extracted by inverting (6.16):

$$S_{ij}(U(t',t),\boldsymbol{\chi}(t)) = a_ib_j\frac{|U_{ij}(t',t)|}{|c_j(t)|^2}. \tag{6.22}$$

[4] By unconditional probabilities, I mean the probabilities of the hidden variable transitioning between pairs of states, given no knowledge of which state was selected prior to transitioning.

[5] Not to be confused with the wave mechanics governed by the Schrödinger wave equation, the arbiter of success for the models presented in this chapter.

[6] One might expect that the absolute value squared might be a better choice, since this matches Fermi's golden rule. Aaronson discusses in his paper, when presenting his own hidden-variable theory *Flow theory*, how absolute values of elements of the unitary have some appealing properties.

Schrödinger theory is the hidden-variable theory I have used in the simulations in the results section of this chapter (Section 6.7). In Section 6.4.1 I detail my efforts to most efficiently find the row and column scalings in order to actually compute the Schrödinger theory S matrix.

The choice to use Schrödinger theory for the simulations was made prior to my learning of Tully's fewest-switches algorithm in the surface hopping literature [20, 172, 177] (also detailed in Section 6.4.2), and fewest-switches has a number of properties that make it more appealing than Schrödinger theory for use in a hidden-variable semiclassical/surface hopping simulation. Tully's fewest-switches has three main advantages. The first is its low computational cost, whereas (as detailed in Section 6.4.1) Schrödinger theory is considerably more computationally expensive to evaluate (except in the case of a two-state system). The second, which is only relevant for time-dependent Hamiltonians, is that fewest-switches can be formulated in a way that allows a distinction to be drawn between motion of a particle through an inhomogeneous field as the cause of non-adiabatic transitions, versus non-adiabatic transitions being caused by explicitly time-dependent fields. I conjecture on the usefulness of this distinction in Sections 6.4.2 and 6.4.3. The final advantage is in the name: fewest-switches makes the *fewest number of switches* consistent with the quantum probabilities in each infinitesimal time interval, whereas Schrödinger theory makes more switches. That is, the transition probabilities in fewest-switches are smaller than those in Schrödinger theory, causing the hidden variable to make fewer transitions in the former than the latter. More switches means larger statistical variation in the outcomes: even though the expectation value of the number of atoms in each state are consistent with the Born rule, the variance is higher, and a larger ensemble will be required to see agreement with the Born rule within some tolerance. More switches also implies larger statistical variation in the classical trajectories, as it increases the chance that atoms have followed multiple classical trajectories in a region of non-adiabatic coupling than followed just one. As mentioned in the introduction to this chapter, Schrödinger theory has one advantage over fewest-switches: it allows one to compute transition probabilities corresponding to quantum evolution over an arbitrary time interval, so long as one has the unitary that describes that evolution. Fewest-switches, on the other hand, produces differential probabilities that must be integrated over time to obtain transition probabilities over intervals that are long compared to the dynamical timescales of the problem. This is usually not a problem since one typically evolves the state vector using timesteps shorter than dynamical timescales of the problem. However Schrödinger theory could still be useful for problems where the quantum evolution can be performed analytically, and where computing transition probabilities at repeated intervals is the limiting factor in the size of simulation timesteps.

6.3 Overview of method

To motivate the method and highlight some necessary properties of a sensible semiclassical model capable of simulating Stern–Gerlach separation, consider, as an example, a spin-$\frac{1}{2}$ particle undergoing separation in a magnetic field gradient. The two spin components accelerate away from each other, with the relative acceleration vector pointing in the same direction as the gradient in field strength. We can ask the question: "What spin-state populations would I see, if I followed the spin-down wavepacket only?" The meaning of 'follow' will be made more precise in Section 6.5, but for now we will simply look at the spin populations in the vicinity of the region of space occupied by the spin-down wavepacket.

As shown in Figure 6.3, one sees the spin-up population decreasing over time until only spin-down population remains. The rate at which spin-up population 'leaves' the region of space we are watching depends on how quickly the wavepackets are accelerating

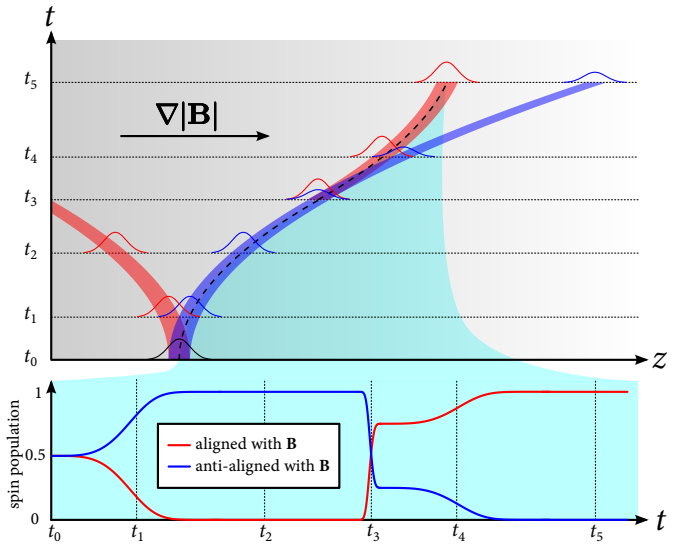

Figure 6.3: Schematic of the method. At t_0 an atom is in a 50:50 superposition of spin-up:spin-down population, and the two spin components accelerate apart in the magnetic field gradient. If the hidden variable dictates that we follow the spin-down component (the initial trajectory given by the dotted line), then we see a reduced spin-up population at times t_1 and t_2. At t_3 the field changes direction suddenly, and partly flips the spins (in the local basis with quantisation axis given by the direction of the magnetic field). The stochastic hidden variable transitions to instead select the spin-up component, which we follow thereafter. Following the spin-up component we then see the a reduced spin-down population at t_5 due to the wavepackets separating once more.

away from each other, as well as the shape of the wavepackets. Likewise, if one follows the spin-up component instead, one sees the spin-down population decrease to zero.

Motivated by this observation, the hidden-variable semiclassical method is a phenomenological model comprised of the following:

- One internal state of the atom is chosen at any moment in time, stored along with the state vector as a stochastic hidden variable that can make a transition at each timestep, according to a hidden-variable theory. The hidden-variable theory takes as input the state vector, and the Hamiltonian or unitary evolution matrix describing the evolution of the internal state of the atom at each timestep. By evolving according to a hidden-variable theory, the probability of the hidden variable selecting a particular internal state is equal to that state's population at each moment in time.

- The internal state of the atom evolves according to the Schrödinger equation, but with the addition of back-action caused by the continuous projection of the atomic wavefunction onto a specific 'classical' motional state under assumptions about the form and evolution of all motional states. This is a vexed matter to which I devote several sections later in this chapter.

- Upon a change in the state selected by the hidden variable, the velocity of the atom is modified instantaneously as required by energy conservation, or, if this would result in a negative kinetic energy, the transition is disallowed.

Written in the same way as in Section 6.1, the model amounts to the following coupled

differential equations and stochastic transition rule:

$$\frac{\mathrm{d}}{\mathrm{d}t}|\tilde{\chi}\rangle = -\frac{\mathrm{i}}{\hbar}\hat{H}(\boldsymbol{r},t)|\tilde{\chi}\rangle - \hat{\Gamma}_\eta(\boldsymbol{r},t)|\tilde{\chi}\rangle\,, \tag{6.23}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{v} = -\frac{1}{m}\nabla V_\eta(\boldsymbol{r},t), \tag{6.24}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{r} = \boldsymbol{v} \tag{6.25}$$

$$\Pr(\eta(t')=i|\eta(t)=j) = S_{ij}(U_{\text{eff}}(t',t),\boldsymbol{\chi}(t)), \tag{6.26}$$

where $|\tilde{\chi}\rangle$ is the internal state vector including the effect of back-action (neglecting normalisation), $\hat{\Gamma}_\eta(\boldsymbol{r},t)$ is a non-Hermitian operator that implements this back-action by decaying the amplitude of states not being followed (there are many ways to do this, all approximate, discussed in Section 6.5), η is the hidden variable, V_η is the adiabatic potential experienced by the eigenstate selected by the hidden variable, $\boldsymbol{\chi}$ is the vector representation of the (normalised) state vector in the local eigenbasis, and $U_{\text{eff}}(t',t)$ is the unitary matrix describing the evolution of the state vector from time t to t' in the local eigenbasis under the action of the effective Hamiltonian $\hat{H}_{\text{eff}}$, defined in Section 6.4.2. In addition to these evolution rules, the state vector must be normalised at each timestep of simulation,[7] and the velocity of the atom modified instantaneously whenever a transition is made in order to conserve energy. This latter requirement is detailed in Section 6.4.3.

[7] It would be relatively simple to include a term in the differential equation to preserve normalisation, but the non-normalisation-preserving differential equation is simpler to write, compute, and understand.

6.4 Hidden variables: implementation details

In this section I go into the gritty details of numerically evaluating hidden-variable theories, conserving energy when a transition occurs, and some other concerns.

6.4.1 Numerically evaluating Schrödinger theory

Despite the high computational cost of Schrödinger theory relative to Tully's fewest-switches, here I present the results of my investigation into how to minimise said cost. There are many ways to numerically solve for the row and column scalings $\{a_i\}$ and $\{b_i\}$ required to compute the S matrix of Schrödinger theory (6.22), but there is one unique solution for the resulting scaled matrix S.[8] The simplest method is to simply alternate between scaling the rows to get the right row sums, then scaling the columns to get the right column sums, and repeating, that is, alternating between solving (6.20) for all a_i, and solving (6.21) for all b_i, until the result converges. This is called the Sinkhorn–Knopp method of r-c (row-column) scaling [192], but is computationally intensive, with slow convergence [193]. An alternative is the method by Linial *et al.* [193] which converges much faster. Both are iterative methods, and so in practice one can save the resulting row and column scalings at each integration step of a simulation and use them as the initial guesses for the same computation at the next integration step,[9] providing a considerable speedup.

For the case of a two-state system, the row and column scalings can be found analyti-

[8] The values of $\{a_i\}$ and $\{b_j\}$ are only determined up to an overall multiplication of each $\{a_i\}$ by a constant and division of each $\{b_j\}$ by the same constant, since only products $a_i b_j$ appear in the resulting scaled matrix.

[9] With the caveat that since the row and column scalings are only determined up to an overall multiplication/division, occasional multiplication of all $\{a_i\}$ and division of all $\{b_i\}$ by a constant may be necessary to prevent the values numerically overflowing or underflowing in the middle of a simulation.

cally, with the result

$$a_1 = 1 \tag{6.27}$$

$$a_2 : a_2^2 + \left(\frac{1}{AB} - \frac{A}{B}|c_1(t)|^2 - \frac{B}{A}|c_2(t)|^2\right)a_2 + \frac{|c_2(t')|^2}{|c_1(t')|^2} = 0;\ a_2 > 0 \tag{6.28}$$

$$b_1 = \frac{|c_1(t')|^2}{A + Ba_2} \tag{6.29}$$

$$b_2 = \frac{|c_2(t')|^2}{B + Aa_2}, \tag{6.30}$$

where $A = |U_{11}| = |U_{22}|$ and $B = |U_{12}| = |U_{21}|$. This result only holds in the case of Schrödinger theory for a state vector with two components subject to unitary evolution—not for row-column scaling in general—since it makes use of symmetries of unitary matrices and the fact that the initial and final probabilities given by the squared amplitudes of the state vector components must sum to unity.

Some further notes on numerics: the above expressions for Schrödinger theory and its analytic expression for a two-state system involve dividing by elements of the unitary, and by state populations, both of which may be zero or very close to zero. Whilst the relevant limits may exist, we cannot easily compute them numerically, and so I have taken to simply replacing small values of $|U_{ij}|$, $|c_i(t')|^2$ and $|c_j(t)|^2$ with a tiny non-zero constant (I use $\varepsilon = 10^{-100}$), ensuring that the convergence criterion (which represents a tolerance for the sum squared error in the column sums) I pass to Linial's method is larger than the square of this by some margin, so as to allow convergence even though modifying the matrix elements may make the matrix no longer row-column scalable to higher precisions. I use a convergence criterion of 10^{-16} for Linial's algorithm, implying the root sum squared error in column sums will be at most 10^{-8}, which is small compared to unity—the sum of all column sums for a perfectly scaled matrix given that the column sums are probabilities that must add to unity. Smaller tolerances imply more iterations before convergence, so tolerances should be as large as is acceptable for the problem if computational cost is a limiting factor.

Potentially faster algorithms exist for row-column scaling of matrices,[10] for example, approaches that treat the problem one of root-finding or optimisation aimed at solving the simultaneous equations (6.20) and (6.21) or minimising their sum squared error [194]. When prototyping with small (3×3) random unitary matrices and random state vectors, I found Newton's method to be effective at quickly solving this set of equations (after fixing $a_1 = 1$ to make them fully determined), requiring considerably fewer iterations that Linial's method. However, the unitaries and state vectors in quantum mechanics are not random, and the fact that most elements of the unitary and state vector are zero when there is no evolution and the atom is in an eigenstate resulted in numerical difficulties with Newton's method that Linial's method does not seem to encounter. Similar to many of the methods in reference [194], one could construct a hybrid method that takes a Newton step, and then checks the row sum and column sum residuals, and if they increased compared to the previous step, ignores that step and takes a step of Linial's method instead. I have not attempted this, and for the moment use Linial's method.

A final note is that my hidden-variable semiclassical method is not only agnostic to which matrix scaling algorithm is used, but that it is also not married to any particular hidden-variable theory. An early version of the method [181] was limited to two-component systems, and the probability of transition was computed as:

$$\Pr(\eta(t')=2|\eta(t)=1) = \max\left(0, |c_2(t')|^2 - |c_2(t)|^2\right), \tag{6.31}$$

$$\Pr(\eta(t')=1|\eta(t)=2) = \max\left(0, |c_1(t')|^2 - |c_1(t)|^2\right), \tag{6.32}$$

[10] In my simulations for realistic 3×3 unitaries corresponding to evolution over small time intervals, Linial's method converges to the aforementioned tolerance in about $100 - 200$ iterations, taking about $20 - 40$ µs per matrix on a 2.9 GHz 7^{th} generation Intel core i7 CPU. This is when transitions are actually occurring; before and after periods of non-adiabatic evolution the algorithm converges in zero or one step when the unitary is the identity and the state vector is in a single eigenstate.

that is, I simply inspected the populations each step and declared any positive change in population of a state as a probability of transition from the other state. Since there were only two states, the originating state of the transition was unambiguous, but the method did not generalise to systems with three or more states.[11] However, it resulted in simulated final populations that on average agreed with the underlying Schrödinger wave equation, leading me to suspect that the exact hidden-variable theory used is not crucial, so long as it satisfies the most obvious of Aaronson's axioms so as not to behave like the product theory mentioned in Section 6.2. I chose Schrödinger theory fairly arbitrarily, it being the one that seemed most easily computable out of the two presented in Aaronson's paper [21], but one might try using Aaronson's flow theory, or inventing another altogether. The surface hopping literature uses Tully's fewest-switches algorithm with much success, as well as approximations to it [196] that are even cheaper, computationally speaking.

[11] This was before I coincidentally came across the definition of a stochastic hidden-variable theory in Aaronson's book *Quantum computing since Democritus* [195] and realised that what I had made was a hidden-variable theory, allowing me to choose a more general one from his paper (and before later still, finding Tully's fewest-switches algorithm).

Interestingly, the main conclusion of Aaronson's paper [21] is that if we could know the entire history of a hidden variable, we could use it to make a computer more powerful than a quantum computer. It is therefore perhaps not surprising that hidden-variable theories ought to be computationally expensive to simulate on a classical computer. Prior to discovering Tully's fewest-switches algorithm, I was therefore somewhat resigned to the fact that any hidden-variable theory would likely be computationally expensive to compute. In light of this it was pleasantly surprising to discover that Tully's fewest-switches (discussed in the next subsection) is computationally cheap. The seeming conflict could be reconciled however if the computational power of hidden variables is crucially dependent on some feature we are not interested in and which fewest-switches does not capture, such as agreement with arbitrary unitaries rather than only those due to evolution over small time intervals, as is assumed by fewest-switches.

6.4.2 Time-dependent formulation of Tully's fewest-switches algorithm

In this subsection I derive Tully's fewest-switches algorithm [20, 172] with the extension that the Hamiltonian may have arbitrary time-dependence. As such, the non-adiabatic transitions that can occur can be due to either the spatial variation of the Hamiltonian, or its time variation. It is important identify which of the two non-adiabatic effects is responsible for a given transition of the hidden variable, as energy conservation only applies to transitions due to spatial variation, whereby the atom is paying/receiving the energy cost of a transition using its kinetic energy. However for a transition due to temporal variation of the Hamiltonian, energy can be added and removed from the atom by the driving field without conserving its total energy. I propose that velocity corrections therefore ought to only be performed following transition of the hidden variable if that transition was due to spatial variation in the Hamiltonian, and not temporal variation.[12]

[12] One cannot simply compute the (partial) time derivative of the Hamiltonian's eigenvalues and multiply by Δt to infer the energy change over one timestep, as the actual transition takes place over many timesteps, despite the hidden variable transitioning during a single timestep. This disconnect is the fundamental source of difficulty in correctly conserving energy in these models.

This proposed extension to fewest-switches is speculative and untested—in Section 6.7 I present results of the model for the case of time-independent fields only. It is clear that the extension would yield the correct behaviour in both the limit of a time-independent inhomogeneous field (energy is always conserved via velocity jumps) and a time-dependent homogeneous field (energy is never required to be conserved and there are no velocity jumps), though its suitability in the intermediate regime is less clear without comparing simulation results to the Schrödinger wave equation.

This subsection also serves to present Tully's fewest-switches algorithm and the concepts and notation related to it that I refer to in later sections.

We begin with the time-dependent Schrödinger equation for the internal state $|\chi(t)\rangle$ of an atom at position $\boldsymbol{r}$:

$$i\hbar \frac{\mathrm{d}}{\mathrm{d}t} |\chi(t)\rangle = \hat{H}(\boldsymbol{r}, t) |\chi(t)\rangle \tag{6.33}$$

Now we take the unitary $\hat{U}_H(\boldsymbol{r},t)$ that transforms state vectors into the eigenbasis of $\hat{H}$ such that

$$\hat{H}(\boldsymbol{r},t) = \hat{U}_H^\dagger(\boldsymbol{r},t)\hat{V}(\boldsymbol{r},t)\hat{U}_H(\boldsymbol{r},t), \tag{6.34}$$

where $\hat{V}(\boldsymbol{r},t)$ is a diagonal operator with diagonals equal to the adiabatic potentials that each eigenstate of $\hat{H}$ is subject to in the adiabatic approximation. We can therefore define the state vector in the adiabatic picture for $\hat{H}$ as[13]

$$|\chi_H(t)\rangle = \hat{U}_H(\boldsymbol{r},t)\,|\chi(t)\rangle \tag{6.35}$$

$$\Rightarrow\ |\chi(t)\rangle = \hat{U}_H^\dagger(\boldsymbol{r},t)\,|\chi_H(t)\rangle\,. \tag{6.36}$$

[13] This is very similar to an interaction picture state vector (Section 3.2.2), but as I have previously used the definition of an interaction picture as the transformation that diagonalises a *time-independent* Hamiltonian, this potentially time-dependent Hamiltonian does not satisfy the definition.

Substituting (6.35) into (6.33) and premultiplying by $\hat{U}_H(\boldsymbol{r},t)$ yields

$$\mathrm{i}\hbar\hat{U}_H(\boldsymbol{r},t)\frac{\mathrm{d}}{\mathrm{d}t}\hat{U}_H^\dagger(\boldsymbol{r},t)\,|\chi_H(t)\rangle = \hat{U}_H(\boldsymbol{r},t)\hat{H}(\boldsymbol{r},t)\hat{U}_H^\dagger(\boldsymbol{r},t)\,|\chi_H(t)\rangle\,, \tag{6.37}$$

which via the product rule and our definition of $\hat{V}(\boldsymbol{r},t)$ simplifies to the differential equation obeyed by the transformed state vector $|\chi_H(t)\rangle$:

$$\mathrm{i}\hbar\frac{\mathrm{d}}{\mathrm{d}t}|\chi_H(t)\rangle = \left[\hat{V}(\boldsymbol{r},t) - \mathrm{i}\hbar\hat{U}_H(\boldsymbol{r},t)\frac{\mathrm{d}}{\mathrm{d}t}U_H^\dagger(\boldsymbol{r},t)\right]|\chi_H(t)\rangle \tag{6.38}$$

$$\equiv \hat{H}_{\text{eff}}\,|\chi_H(t)\rangle\,. \tag{6.39}$$

This equation has the same form as the Schrödinger equation, with the contents of the brackets comprising an effective Hamiltonian dictating the dynamics of the state vector in the *adiabatic basis* (the basis in which $\hat{H}$ is diagonal). Like a non-inertial reference frame in classical mechanics, use of this transformed basis has resulted in the appearance of an extra term in the Hamiltonian, the non-adiabatic coupling term depending on the time derivative of the transformation $\hat{U}_H^\dagger$.

Here we differ from Tully by proceeding without assuming that $\hat{U}_H$ has no explicit time dependence. The total time derivative of $\hat{U}_H^\dagger$ includes both its direct time dependence and the effect of motion through space; the latter obtainable via the chain rule:

$$\frac{\mathrm{d}}{\mathrm{d}t}\hat{U}_H^\dagger(\boldsymbol{r},t) = \frac{\partial}{\partial t}\hat{U}_H^\dagger(\boldsymbol{r},t) + \boldsymbol{v}\cdot\nabla\hat{U}_H^\dagger(\boldsymbol{r},t), \tag{6.40}$$

where $\boldsymbol{v} = \frac{\mathrm{d}\boldsymbol{r}}{\mathrm{d}t}$. Thus (6.38) becomes:

$$\mathrm{i}\hbar\frac{\mathrm{d}}{\mathrm{d}t}|\chi_H(t)\rangle = \left[\hat{V}(\boldsymbol{r},t) - \mathrm{i}\hbar\hat{U}_H(\boldsymbol{r},t)\frac{\partial}{\partial t}\hat{U}_H^\dagger(\boldsymbol{r},t) - \mathrm{i}\hbar\boldsymbol{v}\cdot\hat{U}_H(\boldsymbol{r},t)\nabla\hat{U}_H^\dagger(\boldsymbol{r},t)\right]|\chi_H(t)\rangle\,. \tag{6.41}$$

The final term is identical to the non-adiabatic coupling term in the equation of motion as usually written in the surface-hopping literature [172] being a matrix with elements (in the eigenbasis):

$$\left(-\mathrm{i}\hbar\boldsymbol{v}\cdot U_H(\boldsymbol{r},t)\nabla U_H^\dagger(\boldsymbol{r},t)\right)_{ij} = -\mathrm{i}\hbar\boldsymbol{v}\cdot\langle\chi_i(\boldsymbol{r},t)|\nabla|\chi_j(\boldsymbol{r},t)\rangle\,, \tag{6.42}$$

where $U_H(\boldsymbol{r},t)$ is the matrix representation of $\hat{U}_H$ in any basis that does not vary spatially (i.e. not the eigenbasis), $|\chi_i(\boldsymbol{r},t)\rangle$ is the i^{th} eigenvector of $\hat{H}(r,t)$, and $\langle\chi_i(\boldsymbol{r},t)|\nabla|\chi_j(\boldsymbol{r},t)\rangle$ is the *non-adiabatic coupling vector* between the i^{th} and j^{th} states referred to in the literature [172]. The second to last term in brackets in (6.41) is the additional contribution

due to the time-dependence of the Hamiltonian (more specifically, the time-dependence of its eigenbasis).

We now proceed identically to Tully, computing the rate of change of an eigenstate's population $|c_i(t)|^2$ as:

$$\frac{\mathrm{d}}{\mathrm{d}t}|c_i(t)|^2 = c_i(t)\frac{\mathrm{d}}{\mathrm{d}t}c_i^*(t) + c_i^*(t)\frac{\mathrm{d}}{\mathrm{d}t}c_i(t), \tag{6.43}$$

where via (6.41) we have:

$$\frac{\mathrm{d}}{\mathrm{d}t}c_i(t) = -\frac{\mathrm{i}}{\hbar}\sum_j \left(H_{\text{eff}}(\boldsymbol{r},t)\right)_{ij} c_j(t) \tag{6.44}$$

$$\Rightarrow \frac{\mathrm{d}}{\mathrm{d}t}c_i(t) = -\frac{\mathrm{i}}{\hbar}\sum_j \left[V_{ij}(\boldsymbol{r},t) - \mathrm{i}\hbar\left\langle\chi_i(\boldsymbol{r},t)\right|(\partial_t + \boldsymbol{v}\cdot\nabla)\left|\chi_j(\boldsymbol{r},t)\right\rangle\right] c_j(t). \tag{6.45}$$

This yields the time rate of change of the population $|c_i(t)|^2$:

$$\frac{\mathrm{d}}{\mathrm{d}t}|c_i(t)|^2 = \left[-\frac{\mathrm{i}}{\hbar}\sum_j c_i^*(t)\left(H_{\text{eff}}(\boldsymbol{r},t)\right)_{ij} c_j(t)\right] + \text{c.c.} \tag{6.46}$$

$$= -\frac{2}{\hbar}\sum_j \mathrm{Im}\left(c_i^*(t)\left(H_{\text{eff}}(\boldsymbol{r},t)\right)_{ij} c_j(t)\right) \tag{6.47}$$

$$= -\frac{2}{\hbar}\sum_j \mathrm{Im}\left(c_i^*(t)\left[V_{ij}(\boldsymbol{r},t) - \mathrm{i}\hbar\left\langle\chi_i(\boldsymbol{r},t)\right|(\partial_t + \boldsymbol{v}\cdot\nabla)\left|\chi_j(\boldsymbol{r},t)\right\rangle\right] c_j(t)\right). \tag{6.48}$$

[14] This is not always assumed in the surface hopping literature, since additional couplings are sometimes included in V which have not been removed by diagonalisation.

Since $V(\boldsymbol{r},t)$ is diagonal[14] and real, $c_i^*(t)V_{ij}(\boldsymbol{r},t)c_j(t)$ is zero when $i \neq j$, and has no imaginary part when $i = j$, leaving us with

$$\frac{\mathrm{d}}{\mathrm{d}t}|c_i(t)|^2 = 2\sum_j \mathrm{Re}\left(c_i^*(t)\left\langle\chi_i(\boldsymbol{r},t)\right|(\partial_t + \boldsymbol{v}\cdot\nabla)\left|\chi_j(\boldsymbol{r},t)\right\rangle c_j(t)\right). \tag{6.49}$$

The change in $|c_i(t)|^2$ in a small interval $\mathrm{d}t$ is then:

$$|c_i(t+\mathrm{d}t)|^2 - |c_i(t)|^2 = 2\mathrm{d}t\sum_j \mathrm{Re}\left(c_i^*(t)\left\langle\chi_i(\boldsymbol{r},t)\right|(\partial_t + \boldsymbol{v}\cdot\nabla)\left|\chi_j(\boldsymbol{r},t)\right\rangle c_j(t)\right). \tag{6.50}$$

This is the change in the probability of the atom being in the i^{th} state during that time interval. Tully identifies each term in the sum as a probability flow between a pair of states, and if non-negative, equates each term with the (unconditional) probability of a transition from the j^{th} state to the i^{th} state. We do the same, except that we identify two transition probabilities for each originating state, one due to the spatial variation in the eigenbasis, and one due to the temporal variation. To ensure we don't violate the criterion that on a two-state basis only the minimum number of hops consistent with the total probability flow occur, we clip each probability from above to the probability of any transition occurring at all. This gives us transition probability matrix elements

$$P_{ij}^{\text{space}} = \min\left\{q_{ij}^{\text{total}}, q_{ij}^{\text{space}}\right\}, \tag{6.51}$$

$$P_{ij}^{\text{time}} = \min\left\{q_{ij}^{\text{total}}, q_{ij}^{\text{time}}\right\}, \tag{6.52}$$

where

$$q_{ij}^{\text{space}} = 2\mathrm{d}t\ \mathrm{Re}\left(c_i^*(t)\left\langle\chi_i(\boldsymbol{r},t)\right|\boldsymbol{v}\cdot\nabla\left|\chi_j(\boldsymbol{r},t)\right\rangle c_j(t)\right), \tag{6.53}$$

$$q_{ij}^{\text{time}} = 2\mathrm{d}t\ \mathrm{Re}\left(c_i^*(t)\left\langle\chi_i(\boldsymbol{r},t)\right|\partial_t\left|\chi_j(\boldsymbol{r},t)\right\rangle, c_j(t)\right), \tag{6.54}$$

$$q_{ij}^{\text{total}} = \max\left\{0, (q_{ij}^{\text{space}} + q_{ij}^{\text{time}})\right\}, \tag{6.55}$$

for transitions of the hidden variable from the j^{th} to the i^{th} eigenstate of $\hat{H}(\boldsymbol{r}, t)$ during the time interval $\mathrm{d}t$ due to non-adiabatic spatial and temporal variations in $\hat{H}(\boldsymbol{r}, t)$ respectively. These expressions can also be numerically integrated with respect to time to obtain transition probabilities over finite time intervals—numerically integrating over a short time interval Δt using a midpoint or higher order method can produce transition probabilities more accurate than $\mathcal{O}(\Delta t)$, which would be the accuracy if Δt were simply used in place of $\mathrm{d}t$ in the above expressions.

These matrices have zeros along their diagonals, since the above derivation takes into account only probability changes, and does not count probability remaining in the same state as a transition.[15] To be able to construct properly stochastic matrices, we can take into account the probability mass that remains in the same state simply by imposing conservation of overall probability, defining a diagonal matrix P^{stay} for the unconditional probabilities of remaining in a state:

$$P_{ii}^{\text{stay}} = |c_i(t)|^2 - \sum_{j \neq i} \left(P_{ij}^{\text{space}} + P_{ij}^{\text{time}} \right). \tag{6.56}$$

[15] One can see that the $i = j$ term in (6.50) is zero since $|\chi_i\rangle$ is a unit vector, implying its temporal and spatial derivatives must be orthogonal to $|\chi_i\rangle$ itself, resulting in a zero inner product.

The sum of all three of these matrices now satisfy the row sum and column sum requirements in order to be the unconditional transition probabilities for a hidden-variable theory in the eigenbasis of $\hat{H}$:

$$P = P^{\text{space}} + P^{\text{time}} + P^{\text{stay}}, \tag{6.57}$$

$$\Rightarrow \sum_j P_{ij} = |c_i(t)|^2, \tag{6.58}$$

$$\sum_i P_{ij} = |c_j(t + \mathrm{d}t)|^2. \tag{6.59}$$

The corresponding conditional probabilities of a transition to the i^{th} state occurring—given that the hidden variable was already in the j^{th} state—can be obtained via (6.16) as:

$$S_{ij}^{\text{space}} = \frac{1}{|c_j(t)|^2} P_{ij}^{\text{space}}, \tag{6.60}$$

$$S_{ij}^{\text{time}} = \frac{1}{|c_j(t)|^2} P_{ij}^{\text{time}} \tag{6.61}$$

$$S_{ij}^{\text{stay}} = \frac{1}{|c_j(t)|^2} P_{ij}^{\text{stay}}, \tag{6.62}$$

and the sum of these three matrices of conditional probabilities is the overall (left) stochastic matrix for the fewest-switches hidden-variable theory:

$$S = S^{\text{space}} + S^{\text{time}} + S^{\text{stay}}. \tag{6.63}$$

When using a time-dependent Hamiltonian, under my proposal one would not use this stochastic matrix to make transitions.[16] Rather one would use the individual matrices S^{space}, S^{time}, and S^{stay} in order to distinguish between the different types of transitions, and make the energy conservation part of the surface hopping algorithm conditional on the transition being attributed in this way to the spatial variation of the Hamiltonian rather than its temporal variation.

[16] Though it is instructive to know that the sum of the other conditional probability matrices is indeed a stochastic matrix, such that Tully's fewest-switches does satisfy this requirement of being a hidden-variable theory as defined by Aaronson.

During a simulation, to choose whether a transition occurs due to spatial or temporal variations in the Hamiltonian, one should not make independent random choices based on the three above matrices of conditional probabilities. Rather one should assemble the probabilities of possible events—transitions from the current state to all others via both

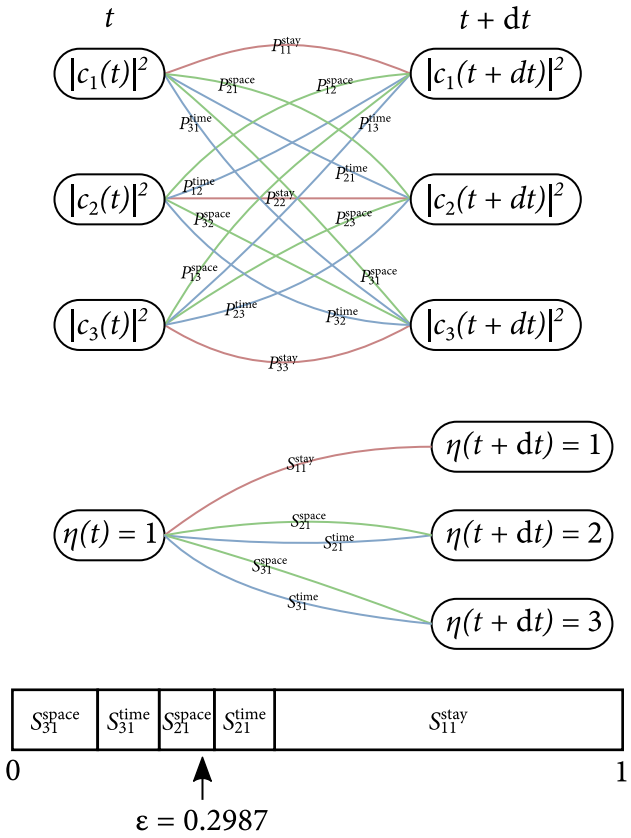

Figure 6.4: Probabilistically choosing a transition. Top: Probability flows between states in a time interval according to the three matrices of unconditional probabilities P^{stay}, P^{space}, and P^{time} in such a way that one total unit of probability is routed from states at the initial time to states at the final time consistent with the populations resulting from quantum mechanical evolution in that timestep. Centre: The hidden variable transitions according to the corresponding conditional probabilities, which are elements of the matrices S^{stay}, S^{space}, and S^{time}. Here the hidden variable is in state 1 and we are choosing whether it will remain in that state or if it will transition to state 2 or 3, and whether it transitions via a spatial or temporal non-adiabatic transition. Bottom: All the probabilities for what may happen to the hidden variable in a timestep sum to one, so an array of the cumulative probabilities can be constructed, and a random number drawn from the range $[0, 1]$. The index of the smallest element of the array of cumulative sums that the random number is smaller than corresponds to the event to occur. In the above diagrammatic example, the result of the random draw is that the hidden variable is to transition to state 2 via a spatially induced non-adiabatic transition.

spatial and temporal non-adiabatic transitions—into a single list of probabilities, and then take the cumulative sum, resulting in a list of numbers between zero and one. A randomly generated number between zero and one can then be used to determine which event occurs, with the correct probability (Figure 6.4).

Framing fewest-switches as a hidden-variable theory

Tully's fewest-switches algorithm allows one to compute transition probabilities for a hidden variable given the Hamiltonian, the state amplitudes, and a small interval of time. Does this satisfy Aaronson's definition of a hidden-variable theory (Section 6.2)? As written, not quite, since it requires the Hamiltonian rather than the unitary that describes state vector evolution over a particular time interval. However it is simple to reconcile the two sets of requirements. Given that the interval of time is small, the unitary describing evolution in the local basis can be linked to the effective Hamiltonian in (6.39) via

$$\hat{U}(t + \mathrm{d}t, t) = \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}\hat{H}_{\text{eff}}(\boldsymbol{r},t)\,\mathrm{d}t}, \tag{6.64}$$

where

$$\hat{H}_{\text{eff}} = \hat{V}(\boldsymbol{r}, t) - \mathrm{i}\hbar\hat{U}_H(\boldsymbol{r}, t)\frac{\mathrm{d}}{\mathrm{d}t}U_H^\dagger(\boldsymbol{r}, t) \tag{6.65}$$

is the effective Hamiltonian in the adiabatic picture for $\hat{H}$, the matrix representation of which can be extracted from the matrix representation of $\hat{U}(t + \mathrm{d}t, t)$ as:

$$H_{\text{eff}}(\boldsymbol{r}, t)\,\mathrm{d}t = \mathrm{i}\hbar\,\mathrm{Log}\,U(t + \mathrm{d}t, t) \tag{6.66}$$

$$\approx \mathrm{i}\hbar\,(\mathbb{I} - U(t + \mathrm{d}t, t)) \tag{6.67}$$

where Log is the principal value of the complex matrix logarithm.

Since only the matrix $H_{\text{eff}}(\boldsymbol{r}, t)\,\mathrm{d}t$ is required to compute transition probabilities according to (6.47), and not its component terms,[17] specifying the initial state vector and unitary for an interval of time evolution (both in the local basis) is a sufficient input to be able to compute transition probabilities using Tully's fewest-switches algorithm. Writing the resulting matrix of probabilities in terms of U gives:

$$P_{ij} = \begin{cases} \max\left\{0, 2\,\mathrm{Re}\left(c_i^*(t)U_{ij}(t + \mathrm{d}t, t)c_j(t)\right)\right\} & i \neq j \\ |c_i(t)|^2 - \sum_{j\neq i} P_{ij} & i = j \end{cases}. \tag{6.68}$$

The corresponding stochastic matrix S can then be obtained by scaling the columns of P by the state populations as in (6.16). Tully's fewest-switches algorithm thus satisfies Aaronson's definition of a hidden-variable theory provided that the time interval is small. The low computational complexity of computing probabilities via fewest-switches is somewhat remarkable. There is no matrix scaling, no matrix permanents, or any other large computational expense.

Expressed in Python, Tully's fewest-switches can be computed very simply as shown below, though this calculation is only first-order accurate in the time interval corresponding to the unitaries.

[17] Though we do need its component terms if we wish to distinguish between spatially vs. temporally induced transitions as in Section 6.4.2.

```
import numpy as np

def S_matrix_fewest_switches(psi, U):
    """Compute the fewest-switches S matrix for an array psi, shape (M, N),
    containing N state vectors for an system with M states, and the array of
    corresponding unitaries unitaries U, shape (M, M, N), describing the evolution
    of the state vectors over a short time interval."""
    P = 2 * np.einsum('in,ijn,jn->ijn', psi.conj(), U, psi).real
    P[P < 0] = 0
    S = P / np.abs(psi**2)
    S_diags = np.einsum('iin->in', S)
    S_diags[...] = 0
    S_diags[...] = 1 - np.sum(S, axis=0)
    return S
```

It is not clear how many of Aaronson's axioms are satisfied by fewest-switches, but the dependence of its probabilities on the actual coupling strengths between states via the non-adiabatic Hamiltonian (and vanishing probabilities when those coupling strengths are zero) is encouraging, and ensures that its behaviour is free of the pathology of the product theory. Furthermore, the form of fewest-switches when framed in terms of the unitary for the given time interval is quite similar to that of Schrödinger theory. In Schrödinger theory one takes the absolute value of elements of U and then applies row and column scalings. In fewest-switches one applies row and column scalings (explicitly given as $c_i^*(t)$ and $c_j(t)$, along with a factor of 2), then takes the real part of the result and clips it to zero from below. The two methods are not identical, but the similarity is striking.

6.4.3 Velocity correction and classically disallowed transitions

When a transition of the hidden variable occurs due to a spatial non-adiabatic transition, the kinetic energy of the atom must be adjusted to conserve overall energy. The force on an atom during a transition from the j^{th} state to the i^{th} state due to the spatial non-adiabatic coupling term in the effective Hamiltonian is in the direction of the non-adiabatic coupling vector [172]

$$\boldsymbol{d}_{ij}(\boldsymbol{r},t) = \langle \chi_i(\boldsymbol{r},t)|\nabla|\chi_j(\boldsymbol{r},t)\rangle \tag{6.69}$$

$$= \left(U_H(\boldsymbol{r},t)\nabla U_H^\dagger(\boldsymbol{r},t)\right)_{ij} \tag{6.70}$$

where $U_H(\boldsymbol{r},t)$ is the matrix representation, in any (spatially and temporally fixed) basis, of the unitary that takes state vectors into the basis in which $\hat{H}(\boldsymbol{r},t)$ is diagonal, defined by (6.34). To conserve energy, the squared component of an atom's velocity in this direction must change by an amount:

$$\Delta v_{ij}^2 = \frac{2}{m}\left(V_j(\boldsymbol{r},t) - V_i(\boldsymbol{r},t)\right), \tag{6.71}$$

where $V_i(\boldsymbol{r},t)$ is the adiabatic potential comprising the i^{th} spatially and/or temporally varying eigenvalue of $\hat{H}(\boldsymbol{r},t)$. However, it is possible that a change of this size can leave an atom with a negative kinetic energy. Whilst this is quantum-mechanically permissible, it is forbidden classically, and so we simply disallow such transitions, defining a modified matrix of unconditional transition probabilities $\tilde{P}^{\text{space}}$ that sets the probability of the disallowed transitions to zero:

$$\tilde{P}_{ij}^{\text{space}} = \begin{cases} P_{ij}^{\text{space}} & \Delta v_{ij}^2 + |\boldsymbol{v}\cdot\hat{\boldsymbol{d}}_{ij}|^2 \geq 0 \\ 0 & \Delta v_{ij}^2 + |\boldsymbol{v}\cdot\hat{\boldsymbol{d}}_{ij}|^2 < 0 \end{cases} \tag{6.72}$$

where $\hat{\boldsymbol{d}}_{ij} = \hat{\boldsymbol{d}}_{ij}(\boldsymbol{r},t)$ is the unit vector in the direction of $\boldsymbol{d}_{ij}(\boldsymbol{r},t)$.[18]

The diagonal matrix for the probabilities of remaining in each state must be adjusted similarly to absorb the probability discarded this way:

$$\tilde{P}_{ii}^{\text{stay}} = |c_i(t)|^2 - \sum_{j\neq i}\left(\tilde{P}_{ij}^{\text{space}} + P_{ij}^{\text{time}}\right) \tag{6.73}$$

The corresponding matrices of conditional probabilities for the hidden variable transitioning, given that it is already in the j^{th} state, are

$$\tilde{S}_{ij}^{\text{space}} = \frac{1}{|c_j(t)|^2}\tilde{P}_{ij}^{\text{space}}, \tag{6.74}$$

$$\tilde{S}_{ij}^{\text{stay}} = \frac{1}{|c_j(t)|^2}\tilde{P}_{ij}^{\text{stay}}, \tag{6.75}$$

[18] The surface hopping literature calls these classically disallowed transitions *frustrated* transitions. Prior to encountering the surface-hopping literature, my method of conserving energy was identical to this (for the case of time-independent potentials), with the exception that I did not know what direction the velocity kick ought to be in, limiting my method to one spatial dimension only. There is much interest in alternate methods of energy conservation in chemical physics, however so far this has been outside of my interest in these methods for simulating cold atoms.

Of course, when making probabilistic transitions of the hidden variable, the originating state is known and so only one column of any of the above matrices is used at a time, so the full matrices $\tilde{P}^{\text{space}}$, $\tilde{P}^{\text{time}}$, $\tilde{S}^{\text{space}}$, and $\tilde{S}^{\text{time}}$ do not need to be computed at each timestep.

In the case that a transition of the hidden variable does occur to the i^{th} from the j^{th} state due to a spatial non-adiabatic coupling, the velocity kick to be applied to the classical velocity vector in order to conserve energy is:

$$\Delta \boldsymbol{v}_{ij} = \left[\text{sgn}(\boldsymbol{v}\cdot\hat{\boldsymbol{d}}_{ij})\sqrt{(\boldsymbol{v}\cdot\hat{\boldsymbol{d}}_{ij})^2 + \frac{2}{m}\Big(V_i(\boldsymbol{r},t) - V_j(\boldsymbol{r},t)\Big)} - (\boldsymbol{v}\cdot\hat{\boldsymbol{d}}_{ij}). \right] \hat{\boldsymbol{d}}_{ij} \tag{6.76}$$

6.5 Decoherence

In the context of the hidden-variable semiclassical/surface-hopping method, *decoherence* refers to the fact that, due to positional separation of different internal states of the atom, those internal states transition from being a coherent superposition into a statistical mixture. For example, in the Stern–Gerlach experiment, a spin may be initially a coherent superposition of spin-up and spin-down, but by the time the two components separate, the superposition is no longer a coherent one. The spin is either up (and the atom off to one side of the screen) or down (and the atom over the other side of the screen). At this point, no interference between the two internal states is observable, as they are not co-located in space. The motional degree of freedom has played the role of an environment, and, having become entangled with the spin of the atom, decohered the spin state. Recoherence can occur if the two wavepackets are brought back together again to have well overlapping positions and velocities, but this is difficult to achieve even intentionally,[19] let alone by accident,

Decoherence was not part of Tully's original surface hopping model [20], such that if used to simulate the Stern–Gerlach experiment, two spots would result on the screen, but both spots would comprise a coherent 50 : 50 superposition of spin-up and spin-down, rather than one being up and one being down. Some decoherence is necessary to include in a hidden-variable semiclassical/surface hopping model in order to avoid this unphysical outcome, and furthermore, to obtain correct transition probabilities in the case that the atom undergoes multiple periods of local spin transitions (as being in a 50 : 50 superposition represents entirely different initial conditions for a Majorana spin flip than being in a single spin state).

Our approach to modelling decoherence requires some assumptions about the motional states of the atom and how they evolve in time. We therefore construct wavepackets that are as classical as we can make them, being as localised in position and velocity space as they can be. For this we use Gaussian wavepackets of width equal to the thermal de Broglie wavelength $\sigma = \lambda_{\text{th}} = h/\sqrt{2\pi m k_{\text{B}} T}$, evolving without dispersion, with their centre-of-mass position and velocity evolving classically according to the adiabatic potential experienced by the eigenstate to which each motional state corresponds.

In Section 6.5.1 I show the general mechanism by which the divergence of trajectories decoheres internal states of an atom, and specifically how this manifests as a decrease in the amplitude of every other state when one imposes the adiabatic trajectory corresponding to a specific internal state. I then show in Sections 6.5.3 and 6.5.2 why naïvely projecting the wavefunction onto a single trajectory at all times yields nonsensical results, and then in Subsections 6.5.4 and 6.5.5 I present two remedies for how to approximately include decoherence in spite of this.

The development of the second of these remedies brought to light a possibility I hadn't considered before, regarding the form of the 'classical' state that the total wavefunction is projected onto at each timestep. Namely, that it makes some sense for this to be a

[19] The problem of getting the wavepackets back together again has been coined the 'Humpty-Dumpty problem' [20, 197] and has made magnetic separation impractical for large-momentum atom interferometry [198], and methods such as the use of Bragg pulses [199, 200] are needed to create large momentum differences between wavepackets without having them more slowly traverse the intermediate regions of momentum space where they might accumulate coherence-destroying phase noise.

Dirac delta instead of a Gaussian wavepacket. In light of this, in Section 6.5.6 I present an alternative view of what it means to 'follow' a wavepacket, and expressions for the decoherence rates of the two methods under this alternate assumption. I argue why this is appealing and why it resolves some issues that are present when projecting onto a Gaussian wavepacket.

6.5.1 Back-action of position measurement on internal state

Consider an atom in a state $|\Psi(t)\rangle$, which is an arbitrary superposition of internal basis states $|\chi_i\rangle$ and motional basis states $|\boldsymbol{r}\rangle$:

$$|\Psi(t)\rangle = \int \sum_i \psi_i(\boldsymbol{r},t)\,|\chi_i\rangle \otimes |\boldsymbol{r}\rangle\,\mathrm{d}\boldsymbol{r}, \tag{6.77}$$

where normalisation requires that

$$\int \sum_i |\psi_i(\boldsymbol{r},t)|^2 \mathrm{d}\boldsymbol{r} = 1. \tag{6.78}$$

Recognising $\psi_i(\boldsymbol{r},t)$ as the i^{th} component of a multi-component wavefunction, we define (up to an arbitrary phase factor) a normalised wavefunction $\phi_i(\boldsymbol{r},t)$ and corresponding motional state vector $|\phi_i(t)\rangle$ and its coefficient $c_i(t)$ for each internal state:

$$c_i(t)\phi_i(\boldsymbol{r},t) \equiv \psi_i(\boldsymbol{r},t) \tag{6.79}$$

$$|\phi_i(t)\rangle \equiv \int \phi_i(\boldsymbol{r},t)\,|\boldsymbol{r}\rangle\,\mathrm{d}\boldsymbol{r} \tag{6.80}$$

such that $\int |\phi_i(\boldsymbol{r},t)|^2 \mathrm{d}\boldsymbol{r} = 1$. This allows us to write our arbitrary state vector as a sum over the internal basis only:

$$|\Psi(t)\rangle = \sum_i c_i(t)\,|\chi_i\rangle \otimes |\phi_i(t)\rangle\,, \tag{6.81}$$

where the spatial state vectors $\{|\phi_i(t)\rangle\}$ are not necessarily orthogonal. To see that spatial separation of the different components leads to decoherence of the internal states, consider the pure density operator corresponding to $|\Psi(t)\rangle$:

$$\hat{\rho}(t) = |\Psi(t)\rangle\langle\Psi(t)| = \sum_{ij} c_i(t)c_j^*(t)\,|\chi_i(t)\rangle\langle\chi_j(t)| \otimes |\phi_i(t)\rangle\langle\phi_j(t)|\,, \tag{6.82}$$

which we can write in the $\{|\chi_i\rangle \otimes |\boldsymbol{r}\rangle\} \equiv \{|\chi_i\,\boldsymbol{r}\rangle\}$ basis, resulting in matrix elements

$$\rho_{ij}(t,\boldsymbol{r},\boldsymbol{r}') = \langle\chi_i\,\boldsymbol{r}|\rho(t)|\chi_j\,\boldsymbol{r}'\rangle\,, \tag{6.83}$$

$$= \psi_i(\boldsymbol{r},t)\psi_j^*(\boldsymbol{r}',t), \tag{6.84}$$

$$= c_i(t)c_j^*(t)\phi_i(\boldsymbol{r},t)\phi_j^*(\boldsymbol{r}',t). \tag{6.85}$$

A partial trace [201] over the motional degree of freedom results in a reduced density operator describing the measurement statistics of the internal degree of freedom only:

$$\hat{\rho}^{\text{red}}(t) = \int \langle\boldsymbol{r}|\hat{\rho}(t)\hat{\boldsymbol{r}}|\boldsymbol{r}\rangle\,\mathrm{d}\boldsymbol{r} \tag{6.86}$$

$$\Rightarrow \rho_{ij}^{\text{red}}(t) = \int \rho_{ij}(t,\boldsymbol{r},\boldsymbol{r}')\delta(\boldsymbol{r}-\boldsymbol{r}')\mathrm{d}\boldsymbol{r} \tag{6.87}$$

$$= c_i(t)c_j^*(t)\int \phi_i(\boldsymbol{r},t)\phi_j^*(\boldsymbol{r},t)\mathrm{d}\boldsymbol{r} \tag{6.88}$$

$$= c_i(t)c_j^*(t)\,\langle\phi_j(t)|\phi_i(t)\rangle\,. \tag{6.89}$$

The off diagonals of the density matrix $\rho_{ij}^{\text{red}}(t)$, representing the coherences of the internal states, are reduced by a factor $\langle\phi_j(t)|\phi_i(t)\rangle$, which will be unity only for pairs of motional state vectors that are identical. We therefore see that spatial separation of the wavefunctions corresponding to different internal states leads to decoherence of the internal states, and we define the decoherence factor

$$R_{ij}(t) = \langle\phi_i(t)|\phi_j(t)\rangle\,, \tag{6.90}$$

such that (6.89) reads

$$\begin{aligned}\rho_{ij}^{\text{red}}(t) &= c_i(t)c_j^*(t)R_{ij}^*(t)\\ &= c_i(t)c_j^*(t)R_{ji}(t).\end{aligned} \tag{6.91}$$

This same decoherence factor appears when—instead of integrating over all positions—we project the total state vector onto a single motional state corresponding to a classical trajectory, which is how we impose classicality on the motional degree of freedom in our model. The effect of decoherence when 'following' a specific motional state through space in this way is to reduce the amplitudes of all other states not being followed, by the decoherence factor between that state and the one being followed, as shown schematically in Figure 6.3.

To obtain an explicit form for the decoherence factor, we need to impose an ansatz for the motional states $\{|\phi_i(t)\rangle\}$. We take each motional state $|\phi_i(t)\rangle$ to be a Gaussian wavepacket propagating—without dispersion—with centre-of-mass motion evolving classically according to the adiabatic potential $V_i(\boldsymbol{r}, t)$ experienced by the local eigenstate $|\chi_i\rangle$. Thus the wavefunctions of these motional states are

$$\langle\boldsymbol{r}|\phi_i(t)\rangle = \phi_i(\boldsymbol{r}, t) = A\exp\left[-\frac{|\boldsymbol{r}-\boldsymbol{r}_i(t)|^2}{4\sigma^2} + \mathrm{i}\frac{m}{\hbar}\boldsymbol{v}_i(t)\cdot(\boldsymbol{r}-\boldsymbol{r}_i(t))\right], \tag{6.92}$$

Where $\boldsymbol{r}_{ij}(t) = \boldsymbol{r}_i(t) - \boldsymbol{r}_j(t)$ and $\boldsymbol{k}_{ij}(t) = \frac{m}{\hbar}\left(\boldsymbol{v}_i(t) - \boldsymbol{v}_j(t)\right)$ are the mean displacement and relative wavevector of the pair of wavepackets, A is a real normalisation constant,[20] and where the centre-of-mass position and velocity of each wavepacket evolve classically according to

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{r}_i(t) = \boldsymbol{v}_i(t) \tag{6.93}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{v}_i(t) = -\frac{1}{m}\nabla V_i(\boldsymbol{r}_i, t). \tag{6.94}$$

This yields the decoherence factor:

$$R_{ij}(t) = \exp\left[-\left(\frac{1}{8\sigma^2}|\boldsymbol{r}_{ij}|^2 + \frac{\mathrm{i}}{2}\boldsymbol{r}_{ij}(t)\cdot\boldsymbol{k}_{ij}(t) + \frac{\sigma^2}{2}|\boldsymbol{k}_{ij}(t)|^2\right)\right]. \tag{6.95}$$

As expected, $R_{ij}(t)$ is equal to unity when the two wavepackets have identical positions and velocities, and decays to zero for increasing relative position and velocity.

If the two motional states being considered are identical at $t = 0$ and have constant relative acceleration $\boldsymbol{a}_{ij}$, then we have

$$\boldsymbol{r}_{ij}(t) = \frac{1}{2}\boldsymbol{a}_{ij}t^2; \tag{6.96}$$

$$\boldsymbol{k}_{ij}(t) = \frac{m}{\hbar}\boldsymbol{a}_{ij}t, \tag{6.97}$$

[20] The overall phase is determined by the offset $\boldsymbol{r} - \boldsymbol{r}_i(t)$ in the second term in the exponent, and is chosen such that $\phi_i(\boldsymbol{r}, t)$ is real at $\boldsymbol{r} = \boldsymbol{r}_i$, which ensures that $\langle\phi_i(t)|\phi_j(t)\rangle$ has a phase depending only on the mean displacement of the two wavepackets rather than their distance from some arbitrary origin. This prevents an additional arbitrary phase at each timestep of the model, which would not affect the dynamics but is unappealing.

and (6.95) reduces to

$$R_{ij}(t) = \exp\left[-\frac{|\boldsymbol{a}_{ij}|^2}{2}\left(\frac{1}{16\sigma^2}t^4 + \mathrm{i}\frac{m}{2\hbar}t^3 + \frac{m^2\sigma^2}{\hbar^2}t^2\right)\right]. \tag{6.98}$$

We will use this expression in Sections 6.5.4 in order to derive approximate decoherence rates under the assumption of uniform relative acceleration.

We are now placed to precisely define what we mean by 'following' a trajectory. Returning to our arbitrary state vector (6.79), we define (ignoring normalisation) the projected state vector

$$|\tilde{\Psi}(t)\rangle = \hat{R}(t)\,|\Psi(t)\rangle\,, \tag{6.99}$$

where $\hat{R}(t) = \hat{1} \otimes |\phi_\eta(t)\rangle\langle\phi_\eta(t)|$, that results from the projection of $|\Psi(t)\rangle$ onto the specific motional state $|\phi_\eta(t)\rangle$ corresponding to the hidden variable $\eta(t)$. This gives (neglecting normalisation) the state vector one would observe conditional on the particle being in that specific motional state at that specific time:

$$\begin{aligned}
|\tilde{\Psi}(t)\rangle &= \hat{R}(t)\,|\Psi(t)\rangle && (6.100)\\
&= \sum_i \langle\phi_\eta(t)|\phi_i(t)\rangle\, c_i(t)\,|\chi_i\rangle \otimes |\phi_\eta(t)\rangle && (6.101)\\
&= \sum_i \tilde{c}_i(t)\,|\chi_i\rangle \otimes |\phi_\eta(t)\rangle && (6.102)\\
&= |\tilde{\chi}(t)\rangle \otimes |\phi_\eta(t)\rangle\,, && (6.103)
\end{aligned}$$

where we have defined $\tilde{c}_i(t) = \langle\phi_\eta(t)|\phi_i(t)\rangle\, c_i(t)$ and $|\tilde{\chi}(t)\rangle = \sum_i \tilde{c}_i(t)\,|\chi_i\rangle$. We recognise $\langle\phi_\eta(t)|\phi_i(t)\rangle$ as the decoherence factor $R_{\eta i}(t)$ defined in (6.90), and can immediately see that the effect of such projection is to reduce the amplitude of all other states ($i \neq \eta$) by this factor.

6.5.2 Continuous projection

We now consider the following protocol: project the state vector at time t as per (6.99), evolve the resulting state vector to $t + dt$ under the action of a projected Hamiltonian, and repeat. In Section 6.5.3 I show that this continuous projection yields unphysical decoherence rates, such that the following derivation will need to be modified in order to usefully model decoherence. Nonetheless this establishes the form of the back-action caused by projections in terms of a decoherence rate, and connects this rate to the decoherence factor (6.90). The result is salvageable, and in Sections 6.5.4 and 6.5.5 I present two methods of computing physically meaningful decoherence rates for use in the resulting equation of motion for the projected state vector.

The state vector evolved to time $t + \mathrm{d}t$ and re-projected with $\hat{R}(t + \mathrm{d}t)$ is

$$|\tilde{\Psi}(t + \mathrm{d}t)\rangle = \hat{R}(t + \mathrm{d}t)\sum_i \left[\tilde{c}_i(t) - \frac{\mathrm{i}}{\hbar}\sum_j H_{ij}(t)\tilde{c}_j(t)\mathrm{d}t\right]|\chi_i\rangle \otimes |\phi_i(t + \mathrm{d}t)\rangle\,, \tag{6.104}$$

where $H_{ij}(t) = \langle\chi_i|\hat{H}_\eta(t)|\chi_j\rangle = \langle\chi_i\,\phi_\eta(t)|\hat{H}(t)|\chi_j\,\phi_\eta(t)\rangle$ are the matrix elements of the projected Hamiltonian $\hat{H}_\eta(t)$ dictating the dynamics of the internal state, given the imposed motional state $|\phi_\eta\rangle$ and the total Hamiltonian $\hat{H}(t)$ of the system. Note that because of the previous projection already performed, all motional states $|\phi_i(t)\rangle$ were 'reset' to be equal to $|\phi_\eta(t)\rangle$ at time t, and therefore the evolved motional state $|\phi_i(t + \mathrm{d}t)\rangle$ represents the evolution of the motional state corresponding to the i^{th}

internal state over an interval $\mathrm{d}t$, starting with the initial condition $|\phi_\eta(t)\rangle$. Accordingly, both motional states will still be approximately equal after this short evolution, allowing us to write

$$\langle\phi_\eta(t+\mathrm{d}t)|\phi_i(t+\mathrm{d}t)\rangle = \langle\phi_\eta(t)|\phi_i(t)\rangle + \frac{\mathrm{d}}{\mathrm{d}t}\langle\phi_\eta(t)|\phi_i(t)\rangle\,\mathrm{d}t \tag{6.105}$$

$$= 1 + \frac{\mathrm{d}R_{\eta i}(t)}{\mathrm{d}t}. \tag{6.106}$$

Using this fact in applying the projection in (6.104) yields a product state once more:

$$|\tilde{\Psi}(t+\mathrm{d}t)\rangle = \sum_i \left(1 + \frac{\mathrm{d}r_{\eta i}(t)}{\mathrm{d}t}\right)\left[\tilde{c}_i(t) - \frac{\mathrm{i}}{\hbar}\sum_j H_{ij}(t)\tilde{c}_j(t)\mathrm{d}t\right]|\chi_i\rangle \otimes |\phi_\eta(t+\mathrm{d}t)\rangle \tag{6.107}$$

$$\Rightarrow |\tilde{\chi}(t+\mathrm{d}t)\rangle = \sum_i \left(1 + \frac{\mathrm{d}r_{\eta i}(t)}{\mathrm{d}t}\right)\left[\tilde{c}_i(t) - \frac{\mathrm{i}}{\hbar}\sum_j H_{ij}(t)\tilde{c}_j(t)\mathrm{d}t\right]|\chi_i\rangle \tag{6.108}$$

which we can solve for $\frac{1}{\mathrm{d}t}\left(|\tilde{\chi}(t+\mathrm{d}t)\rangle - |\tilde{\chi}(t)\rangle\right)$ to obtain a differential equation for $|\tilde{\chi}(t)\rangle$:

$$\frac{\mathrm{d}}{\mathrm{d}t}|\tilde{\chi}(t)\rangle = \left[-\frac{\mathrm{i}}{\hbar}\hat{H}_\eta(t) - \hat{\Gamma}_\eta(t)\right]|\tilde{\chi}(t)\rangle\,, \tag{6.109}$$

where $\hat{\Gamma}_\eta(t)$ is a diagonal operator in the $\{|\chi_i\rangle\}$ basis with elements

$$(\Gamma_\eta(t))_{ii} = \gamma_{\eta i}(t) = -\frac{\mathrm{d}R_{\eta i}(t)}{\mathrm{d}t} = -\frac{\mathrm{d}}{\mathrm{d}t}\langle\phi_\eta(t)|\phi_i(t)\rangle\,, \tag{6.110}$$

where we have defined $\gamma_{ij}(t) = -\frac{\mathrm{d}R_{ij}(t)}{\mathrm{d}t}$. The diagonals of $\hat{\Gamma}_\eta(t)$ in the adiabatic basis are *decoherence rates*, and cause exponential damping of all internal states other than the $|\chi_\eta\rangle$ state. The damping rates increase with the rate at which the given internal state diverges from $|\phi_\eta(t)\rangle$.

6.5.3 The quantum Zeno effect

But now we arrive at a problem. Because all the motional states are reset to be equal to $|\phi_\eta(t)\rangle$ at each timestep as in (6.104), the relative position and wavevector of the two wavepackets are zero at all times, and so our decoherence rates are:

$$\gamma_{ij} = -\left[\frac{\mathrm{d}R_{ij}(t)}{\mathrm{d}t}\right]_{\substack{\boldsymbol{k}_{ij}=0\\ \boldsymbol{v}_{ij}=0}} \tag{6.111}$$

$$= -\left[\frac{\partial R_{ij}(t)}{\partial \boldsymbol{r}_{ij}}\cdot\frac{\mathrm{d}\boldsymbol{r}_{ij}(t)}{\mathrm{d}t} + \frac{\partial R_{ij}(t)}{\partial \boldsymbol{k}_{ij}}\cdot\frac{\mathrm{d}\boldsymbol{k}_{ij}(t)}{\mathrm{d}t}\right]_{\substack{\boldsymbol{k}_{ij}=0\\ \boldsymbol{v}_{ij}=0}} \tag{6.112}$$

$$= \left[\left(\frac{\boldsymbol{r}_{ij}}{4\sigma^2} + \frac{\mathrm{i}}{2}\boldsymbol{k}_{ij}\right)R_{ij}(t)\cdot\frac{\mathrm{d}\boldsymbol{r}_{ij}(t)}{\mathrm{d}t} + \left(\sigma^2\boldsymbol{k}_{ij} + \frac{\mathrm{i}}{2}\boldsymbol{r}_{ij}\right)R_{ij}(t)\cdot\frac{\mathrm{d}\boldsymbol{k}_{ij}(t)}{\mathrm{d}t}\right]_{\substack{\boldsymbol{k}_{ij}=0\\ \boldsymbol{v}_{ij}=0}} \tag{6.113}$$

$$= 0. \tag{6.114}$$

Every decoherence rate is zero. What is going on? The answer is the quantum Zeno effect [202, 203], which is the name given to the fact that, in the limit of infinitely frequent measurements of whether quantum evolution has occurred, the back-action of

the measurement has the effect of preventing the evolution from occurring at all. In our case, the fact that we are constantly projecting onto the followed motional state causes the amplitude flows to the other motional states to be exactly zero. A sufficiently closely watched quantum pot never boils, and a sufficiently closely watched Schrödinger's cat never may never be poisoned [203]:

> In view of the Zeno's paradox formulated above, should we conclude that the particle will never decay? Will the cat escape the cruel death awaiting it, against which it has no defense, provided its vital signs are constantly watched with loving care?

The appearance of the quantum Zeno effect ought to be a reminder that the assumption of infinitely frequent projective measurements is unphysical. In our case it certainly is—we have merely imagined a hypothetical measurement device collapsing our position states because we don't want to simulate them, not because any such measurement device actually exists. Nonetheless the internal states of the atom *do* decohere even in the absence of measurement (and there is eventual measurement when the atoms collide with other atoms or otherwise interact with anything in a position-dependent way), and we wish to model the approximate effect of this, if possible with a differential equation that does not require us to simulate all the quantum details of the motional degree of freedom.

Note that if the decoherence factor had the form of exponential decay:

$$R_{ij}^{\text{Markov}}(t) = \mathrm{e}^{-\gamma_{ij} t}, \tag{6.115}$$

then the decoherence rate would be the constant γ_{ij}, and not zero. This is the case for Markovian decoherence [201], which is when the environment has no memory of its past interaction with the system. The memory in our case is due to the wavepackets accelerating away from each other, starting from zero relative position and velocity at each timestep—the relative position and velocity comprise a memory of the past interaction, which we are erasing each time we reproject.

The lack of an environmental memory is only ever approximately true, and no decoherence factors in nature have the form of a decaying exponential at all times. At small enough times the overlap between two states can only move away from unity quadratically owing to unitary evolution on account of the Schrödinger equation, guaranteeing an initial time derivative of zero for all physical decoherence factors. However in many systems of interest, the specifics of the interaction with the system are quickly forgotten by the environmental states, and the decoherence factor does approach a decaying exponential [204]. This is the case, for example, for spontaneous photon emission by atoms, which one can consider a measurement effect in which the electromagnetic field is being regularly measured by the environment in the photon number basis [183, 184, 186]. In the various quantum trajectories methods used to simulate atoms in the presence of spontaneous emission, if one assumes that the measurements are projective, one must simply assume that the frequent measurements take place at large enough timescales that the decoherence factor is in the exponential decay regime, provided that this is much smaller than other dynamical timescales, which is true for spontaneous emission [186]. The measurement interval assumed "should be large enough to allow the photons to get away from the atom" [185]. This can be recast as a continuous *weak* measurement, rather than infrequent projective measurements [186, 205], which is more physically realistic than the assumption of projective measurements at somewhat arbitrarily chosen intervals, but results in the same differential equations.

A decoherence factor that has the form of a decaying exponential implies that any chosen measurement interval results in the same fractional reduction in state amplitudes, since the differential equation $\frac{\mathrm{d}\tilde{c}_i(t)}{\mathrm{d}t} = -\gamma \tilde{c}_i(t)$ has the solution $\tilde{c}(t) = \mathrm{e}^{-\gamma t}\tilde{c}(0)$, that is,

repeated consideration of only the first part of the decoherence factor ends up tracing out the whole decoherence curve over time.

How can we replicate something like this for our separating wavepackets? Here I present two approaches. The first, described in Section 6.5.4, is to gloss over as many of the details of the wavepacket separation as possible and replace the decoherence factor with an exponential one, describing the wavepackets separating on approximately the correct timescale. This is crude, but better than no decoherence at all. The second, described in Section 6.5.5, is to introduce a minimal memory of the separation of wavepackets. In this approach, one (but only one) trajectory is simulated for the motional state corresponding to each internal state. Whenever there is population transfer from the main trajectory to another state, the trajectory corresponding to the other state is replaced with a weighted average of its existing position and velocity with the position and (adjusted for energy conservation) velocity of the main trajectory. These *auxiliary trajectories* are used to compute a dynamic decoherence rate that traces out a more correct decoherence curve as the wavepackets separate.

6.5.4 Approximate Markovian decoherence

A crude way to include decoherence is just to approximate $R_{\eta i}(t)$—for the case of constant acceleration in (6.98)—as a decaying exponential with roughly the right decay constant. This is crude because $R_{\eta i}(t)$ does not look much like a decaying exponential (see Figure 6.5). In the limit of large t, its functional form is e^{-t^4}, not the exponential decay required to treat the decoherence as Markovian at any timescale.

To nonetheless find an approximate Markovian decoherence rate, we first construct a 'time ignorant' version $\tilde{R}_{ij}(t)$ of the decoherence factor $R_{ij}(t)$ given in (6.98) that answers the question "What is the expected value of $R_{ij}(t)$ at all times $t > 0$ if I don't know how long before $t = 0$ the two wavepackets began separating?"

We can then take the derivative of this time-ignorant decoherence factor at $t = 0$ to use as an approximate Markovian decoherence rate $\gamma_{ij}^{\text{Markov}}$. Whilst extremely approximate, this method of including decoherence is nonetheless an improvement over the Ehrenfest method (which has no decoherence), and over Tully's original fewest-hops surface-hopping method [20], which also lacked any decoherence. Others [206–208] have since developed various methods to induce approximate damping of states to achieve a similar outcome, including the approximation of an exponential decoherence factor [206], though my development of this method was independent.

Proceeding, we define the time-ignorant decoherence factor $\tilde{R}_{ij}(t)$ as the average of all decoherence factors one would obtain if the two wavepackets began separating at some point in time before $t = 0$:

$$\tilde{R}_{ij}(t) = A \int_{-\infty}^{0} \langle \phi_i(t - t') | \phi_j(t - t') \rangle \, \mathrm{d}t' \tag{6.116}$$

$$= A \int_{-\infty}^{0} R_{ij}(t - t') \, \mathrm{d}t'. \tag{6.117}$$

Here, A is a normalisation constant such that $\tilde{R}_{ij}(0) = 1$, and where we take that $|\phi_i(0)\rangle = |\phi_j(0)\rangle$ with each thereafter evolving according to the classical motion of their centre of mass with constant relative acceleration as in (6.93) and (6.94) such that the $R_{ij}(t)$ above has the form of (6.98).

Our time-ignorant decoherence rate is then the (negative of the) derivative of $\tilde{R}_{ij}(t)$

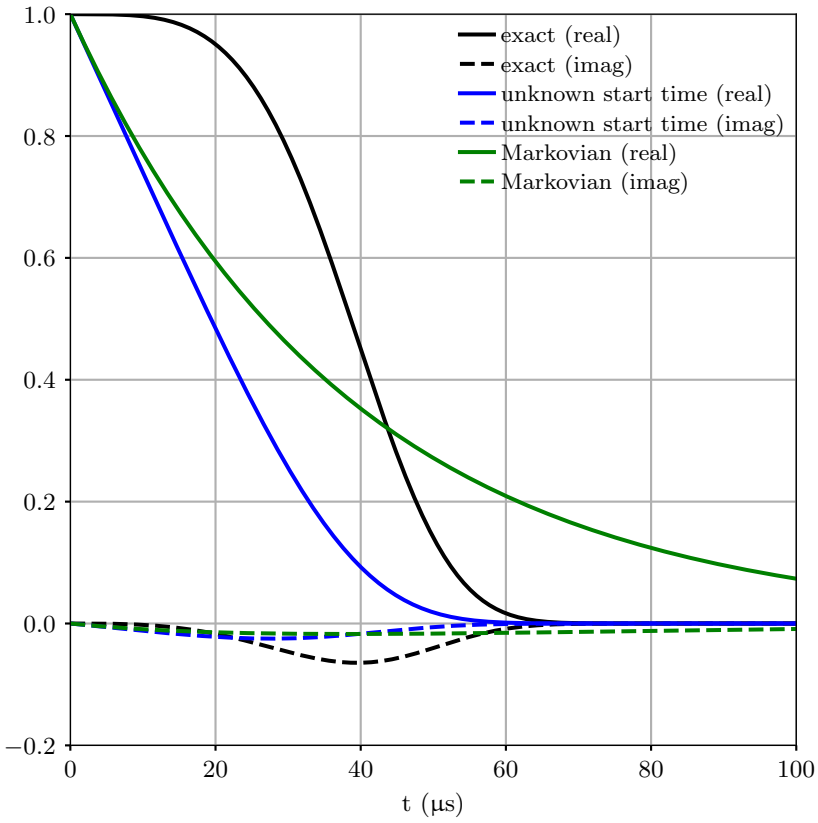

Figure 6.5: Example decoherence factor (6.98) as a function of time for an experimentally realistic parameters. Plotted in black is the exact decoherence factor $R_{ij}(t)$ given by (6.98) between two adjacent ($\Delta m_F = \pm 1$) Zeeman sublevels of the $F = 1$ ground state of ^{87}Rb, assuming constant relative acceleration due to a magnetic field gradient of 250 G cm^{-1} and a Gaussian wavepacket width $\sigma = \lambda_{\text{th}} = 2.6\,\mu\text{m}$, corresponding to the thermal wavelength $\lambda_{\text{th}} = h/\sqrt{2\pi m k_{\text{B}} T}$ at $T = 53\,\mu\text{K}$. Shown in blue is the 'time-ignorant' decoherence factor described in text. In green is the Markovian approximation to the exact decoherence factor, obtained by extracting a decay constant from the gradient of the time-ignorant decoherence factor at $t = 0$. As can be seen, whilst no decaying exponential is a particularly good fit to the exact decoherence factor, our Markovian approximation is as good as can be expected, decaying to zero over approximately the correct timescale.

at $t = 0$:

$$\gamma_{ij}^{\text{Markov}} = -\left[\frac{\mathrm{d}\tilde{R}_{ij}(t)}{\mathrm{d}t}\right]_{t=0} = -\frac{\left[\frac{\mathrm{d}}{\mathrm{d}t}\int_{-\infty}^{0} R_{ij}(t-t')\,\mathrm{d}t'\right]_{t=0}}{\left[\int_{-\infty}^{0} R_{ij}(t-t')\,\mathrm{d}t'\right]_{t=0}}. \tag{6.118}$$

Moving the derivative inside the integral, noting that $\frac{\mathrm{d}R_{ij}(t-t')}{\mathrm{d}t} = \frac{\mathrm{d}R_{ij}(t-t')}{\mathrm{d}(t-t')}$ and setting

$t = 0$, we get:

$$\gamma_{ij}^{\text{Markov}} = -\frac{\int_{-\infty}^{0} R'_{ij}(-t')\,\mathrm{d}t'}{\int_{-\infty}^{0} R_{ij}(-t')\,\mathrm{d}t'} \tag{6.119}$$

$$= -\frac{\int_{0}^{\infty} R'_{ij}(t')\,\mathrm{d}t'}{\int_{0}^{\infty} R_{ij}(t')\,\mathrm{d}t'} \tag{6.120}$$

where R'_{ij} is the derivative of R_{ij} with respect to its argument. By the fundamental theorem of calculus the numerator is -1, since $R_{ij}(t)$ decreases from unity at $t = 0$ to zero as t goes to infinity, leaving us with:

$$\frac{1}{\gamma_{ij}^{\text{Markov}}} = \int_0^\infty R_{ij}(t')\,\mathrm{d}t' \tag{6.121}$$

$$= \int_0^\infty \exp\left[-\left(\frac{1}{8\sigma^2}|\boldsymbol{r}_{ij}|^2 + \frac{\mathrm{i}}{2}\boldsymbol{r}_{ij}(t')\cdot\boldsymbol{k}_{ij}(t') + \frac{\sigma^2}{2}|\boldsymbol{k}_{ij}(t')|^2\right)\right]\mathrm{d}t' \tag{6.122}$$

The expression (6.98) requires a relative acceleration, for which we use the relative acceleration between the pair of adiabatic potentials at the current moment in time and current position of the atom during a simulation:

$$\boldsymbol{a}_{ij}(\boldsymbol{r},t) \approx -\frac{1}{m}\left(\nabla V_i(\boldsymbol{r},t) - \nabla V_j(\boldsymbol{r},t)\right). \tag{6.123}$$

This now gives a decoherence factor depending on position and time:

$$\frac{1}{\gamma_{ij}^{\text{Markov}}(\boldsymbol{r},t)} = \int_0^\infty \exp\left[-\frac{|\boldsymbol{a}_{ij}(\boldsymbol{r},t)|^2}{2}\left(\frac{1}{16\sigma^2}t'^4 + \mathrm{i}\frac{m}{2\hbar}t'^3 + \frac{m^2\sigma^2}{\hbar^2}t'^2\right)\right]\mathrm{d}t' \tag{6.124}$$

In order to obtain an approximate analytic expression for this integral, we consider two limiting cases and then stitch them together in the intermediate regime. In the limit of small wavepackets, σ is small and thus the first term in the exponent in (6.121) is largest, and the third term is smallest. In this regime, which describes when positional separation (as opposed to separation in velocity space) dominates the decoherence, we'll neglect the third term in the exponent and treat the second term as small relative to the first. This gives us:

$$\frac{1}{\gamma_{ij\,(\text{pos})}^{\text{Markov}}(\boldsymbol{r},t)} \approx \int_0^\infty \exp\left[-\frac{|\boldsymbol{a}_{ij}(\boldsymbol{r},t)|^2}{2}\left(\frac{1}{16\sigma^2}t'^4 + \mathrm{i}\frac{m}{2\hbar}t'^3\right)\right]\mathrm{d}t' \tag{6.125}$$

$$\approx \int_0^\infty \exp\left[-\frac{|\boldsymbol{a}_{ij}(\boldsymbol{r},t)|^2}{32\sigma^2}t'^4\right]\left(1 - \mathrm{i}\frac{m|\boldsymbol{a}_{ij}(\boldsymbol{r},t)|^2}{4\hbar}t'^3\right)\mathrm{d}t' \tag{6.126}$$

$$= 2^{\frac{5}{4}}\Gamma(\tfrac{5}{4})\sqrt{\frac{\sigma}{|\boldsymbol{a}_{ij}(\boldsymbol{r},t)|}} - 2\mathrm{i}\frac{m\sigma^2}{\hbar}, \tag{6.127}$$

where we used a first-order Taylor expansion of an exponential in (6.126). We similarly use a first-order expansion $(x+\varepsilon)^{-1} \approx x^{-1} - \varepsilon x^{-2}$ to take the reciprocal of (6.127) (since the second term is much smaller than the first[21]), and arrive at

$$\gamma_{ij\,(\text{pos})}^{\text{Markov}}(\boldsymbol{r},t) \approx \frac{1}{2^{\frac{5}{4}}\Gamma(\frac{5}{4})}\sqrt{\frac{|\boldsymbol{a}_{ij}(\boldsymbol{r},t)|}{\sigma}} + \frac{\mathrm{i}}{2^{\frac{3}{2}}\Gamma(\frac{5}{4})^2}\frac{m\sigma|\boldsymbol{a}_{ij}(\boldsymbol{r},t)|}{\hbar}. \tag{6.128}$$

[21] This isn't necessary in order to obtain a simple expression for $\gamma_{ij\,(\text{pos})}$—the reciprocal without this approximation is equally simple—but it leaves us with power laws for the real and imaginary parts of $\gamma_{ij\,(\text{pos})}$, which are easier to stitch together with those from the large σ regime. It also ensures we don't divide by zero when $\boldsymbol{a}_{ij}(\boldsymbol{r},t)$ is zero.

Similarly for the large σ regime, we neglect the first term in the exponent of (6.121) and consider the second term small relative to the third. This is the regime in which the decrease in overlap of the two wavepackets is dominated by their separation in velocity space. Following the same process as above gives:

$$\frac{1}{\gamma_{ij\,(\mathrm{vel})}^{\mathrm{Markov}}(\boldsymbol{r},t)} \approx \int_0^\infty \exp\left[-\frac{|\boldsymbol{a}_{ij}(\boldsymbol{r},t)|^2}{2}\left(\mathrm{i}\frac{m}{2\hbar}t'^3+\frac{m^2\sigma^2}{\hbar^2}t'^2\right)\right]\mathrm{d}t' \tag{6.129}$$

$$\approx \int_0^\infty \left(1-\mathrm{i}\frac{m|\boldsymbol{a}_{ij}(\boldsymbol{r},t)|^2}{4\hbar}t'^3\right)\exp\left[-\frac{m^2\sigma^2|\boldsymbol{a}_{ij}(\boldsymbol{r},t)|^2}{2\hbar^2}t'^2\right]\mathrm{d}t \tag{6.130}$$

$$= \sqrt{\frac{\pi}{2}}\frac{\hbar}{m\sigma|\boldsymbol{a}_{ij}(\boldsymbol{r},t)|}-\mathrm{i}\frac{\hbar^3}{2m^3\sigma^4|\boldsymbol{a}_{ij}(\boldsymbol{r},t)|^2} \tag{6.131}$$

$$\Rightarrow \gamma_{ij\,(\mathrm{vel})}^{\mathrm{Markov}}(\boldsymbol{r},t) \approx \sqrt{\frac{2}{\pi}}\frac{m\sigma|\boldsymbol{a}_{ij}(\boldsymbol{r},t)|}{\hbar}+\mathrm{i}\frac{\hbar}{\pi m\sigma^2} \tag{6.132}$$

Equations (6.128) and (6.132) are our final expressions for the Markovian decoherence rate in the limit of small and large wavepackets respectively. Adding their real parts in quadrature and adding the reciprocals of their imaginary parts then provides a reasonable approximation for $\gamma_{ij}^{\mathrm{Markov}}(\boldsymbol{r},t)$ over all wavepacket sizes:

$$\gamma_{ij}^{\mathrm{Markov}}(\boldsymbol{r},t) \approx \left[\mathrm{Re}(\gamma_{ij\,(\mathrm{pos})}^{\mathrm{Markov}}(\boldsymbol{r},t))^2+\mathrm{Re}(\gamma_{ij\,(\mathrm{vel})}^{\mathrm{Markov}}(\boldsymbol{r},t))^2\right]^{\frac{1}{2}} + \mathrm{i}\left[\mathrm{Im}(\gamma_{ij\,(\mathrm{pos})}^{\mathrm{Markov}}(\boldsymbol{r},t))^{-1}+\mathrm{Im}(\gamma_{ij\,(\mathrm{vel})}^{\mathrm{Markov}}(\boldsymbol{r},t))^{-1}\right]^{-1}. \tag{6.133}$$

We now have an approximate analytic expression for a Markovian decoherence rate between two Gaussian wavepackets that is computationally inexpensive to evaluate for each semiclassical atom in an ensemble at every timestep of a differential equation. An example showing the accuracy of (6.133), compared to the exact expression (6.124) for $\gamma_{ij}^{\mathrm{Markov}}$ over a range of wavepacket sizes is shown in Figure 6.6.

6.5.5 Decoherence with mean auxiliary trajectories

A more accurate approach is to compute a decoherence factor at each timestep of the simulation by modelling the wavepackets as they accelerate away from each other, retaining in the model the position and velocity of each wavepacket. This inclusion of additional positions and velocities in the model comprises a minimal memory of the past interaction between the system and environment—that is, the internal and motional degrees of freedom of the atom.

In quantum mechanics there is not, in general, a single Gaussian wavepacket for each state, and even if the wavefunction corresponding to the originating state is a Gaussian wavepacket, transitions to other states at different times can produce an arbitrary superposition of Gaussian wavepackets in each other state. If we were to simulate this arbitrary superposition for all components of the multi-component wavefunction, we would not be saving any computational power at all, as we would have reverted to a fully quantum description of the motional degree of freedom, the antithesis of our intention. Therefore we draw the line at modelling a single trajectory for each state, in order to maintain simplicity whilst including at least some dynamics of the system–environment interaction.

Below I will talk about how to convert a decoherence factor at each point in time into a decoherence *rate* at each point in time, that can be included in a differential equation for the state vector of the atom's internal state. Then I will introduce the averaging scheme I

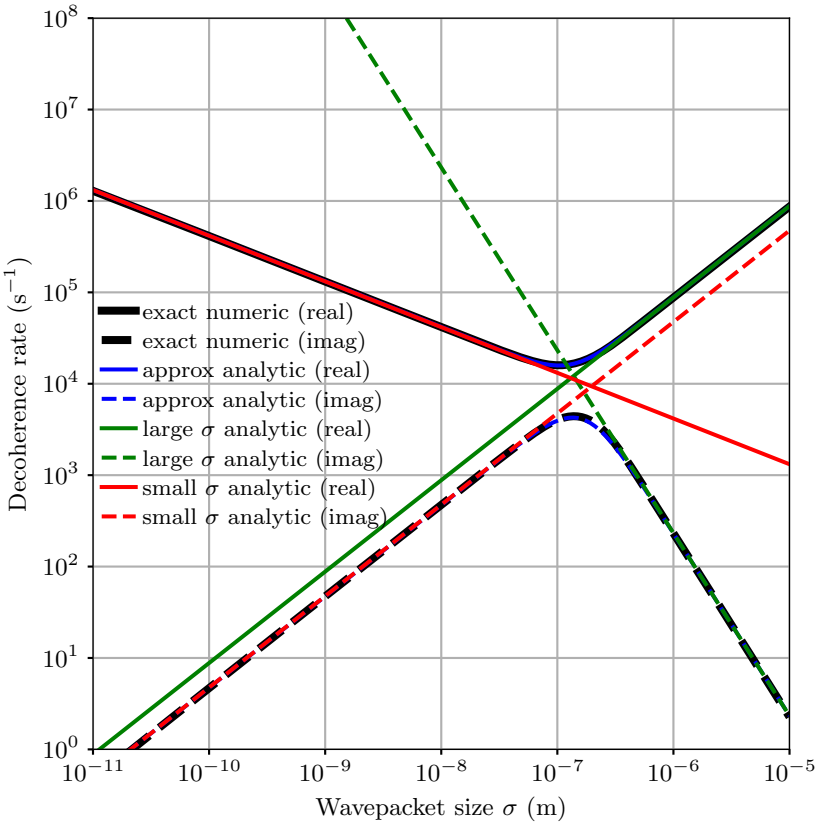

Figure 6.6: Comparison of approximate analytic Markovian decoherence rate (6.133) and exact Markovian decoherence rate (6.124) between adjacent ($\Delta m_F = \pm 1$) Zeeman sublevels of the $F = 1$ ground state of a ^{87}Rb atom in a 250 G cm^{-1} magnetic field gradient. Shown are the real (solid) and imaginary (dashed) parts of the small wavepacket limit (red) given by (6.128) and large wavepacket limit (green) given by (6.132), their stitching together in the intermediate wavepacket regime (blue) given by (6.133) and the exact expression they approximate (black, not easily visible due to being similar to the blue curves) given by (6.124). There is good agreement over all wavepacket sizes.

am using in order to turn what would be many trajectories for each state into one average trajectory.

Use of 'auxiliary trajectories' for computing decoherence rates is common in the surface-hopping literature. There is a range of methods called 'multiple spawning' methods, in which additional auxiliary trajectories regularly branch off the main trajectory (the one corresponding to the hidden variable), and are all considered in order to compute the overlap with the main wavepacket and hence the decoherence factor [209–211]. These methods generally contain a parameter controlling the likelihood of new trajectories branching off, and trajectories can be discarded once they recede far enough from the main trajectory. If the parameter is tuned far enough, there can be so many auxiliary trajectories that an extremely accurate decoherence rate can be computed, though the computational cost in this case approaches that of a fully quantum simulation. A method by Shenvi *et al.* [174], like mine, tracks only one auxiliary trajectory per internal atomic state, probabilistically replacing them with newly spawned trajectories rather than simulating multiple auxiliary trajectories per state. However, the trajectories are

spawned at specific moments—when the non-adiabatic coupling strength reaches a local maximum—leading to different behaviour in different locations in space. For example, if an atom were orbiting a magnetic field zero in a magnetic trap, this method would only spawn new trajectories at the orbit's point of closest approach, and decoherence rates would be inaccurate at other points in the orbit, even if the non-adiabatic coupling were still high throughout the orbit. My method has the potential to improve upon this due to the fact that it is in a sense spawning new trajectories all the time—rather than only at specific points in space as the method by Shenvi *et al.*—and avoids the problem of an exponential proliferation of trajectories by only retaining a single, more-or-less representative trajectory for each state rather than many.

Decoherence rate in the absence of projective measurement

Here we generalise our conception of the decoherence rate by removing the assumption that motional states are 'reset' at each timestep. This resetting, which we assumed in Section 6.5.2, set each motional state to be equal to $|\phi_\eta(t)\rangle$ at each timestep, and this led to all decoherence rates being zero due to the quantum Zeno effect (Section 6.5.3). What is the reduction in the state amplitudes at each timestep if the wavepackets are not reset, but are left as arbitrary states? We know that if we were to perform a projective measurement at time t, the i^{th} state amplitude would be reduced by a factor of $R_{\eta i}(t)$ compared to the original state amplitude $c_i(t)$ as in (6.99). If motional states are not reset at each timestep, then $R_{\eta i}(t)$ is not necessarily close to unity, as the wavepackets may be non-negligibly separated. In the absence of Hamiltonian evolution such that $\frac{\mathrm{d}c_i(t)}{\mathrm{d}t} = 0$, the rate of change in $\tilde{c}_i(t)$ due to the changing decoherence factor is:

$$\frac{\mathrm{d}\tilde{c}_i(t)}{\mathrm{d}t} = \frac{\mathrm{d}R_{\eta i}(t)}{\mathrm{d}t} c_i(t) \tag{6.134}$$

$$= \frac{1}{R_{\eta i}(t)} \frac{\mathrm{d}R_{\eta i}(t)}{\mathrm{d}t} \tilde{c}_i(t). \tag{6.135}$$

and so we see that a decoherence rate that does not reset the environment states is the (negative of the) logarithmic derivative of the decoherence factor at any time, rather than just its derivative at $t = 0$. This case encompasses the earlier case in which $R_{\eta i}(t) = 1$ at all times, in which case the logarithmic and ordinary derivatives are the same. Thus, given the decoherence factor between two states, along with its time derivative, we can compute a decoherence rate

$$\gamma_{ij}^{\text{MM}}(t) = -\frac{1}{R_{ij}(t)} \frac{\mathrm{d}R_{ij}(t)}{\mathrm{d}t}, \tag{6.136}$$

such that in the absence of Hamiltonian evolution

$$\frac{\mathrm{d}\tilde{c}_i(t)}{\mathrm{d}t} = -\gamma_{\eta i}^{\text{MM}}(t)\, \tilde{c}_i(t), \tag{6.137}$$

where 'MM" stands for 'minimal-memory'.

The (negative) logarithmic derivative of the decoherence factor (6.95) for the fixed-width Gaussian wavepackets gives the minimal-memory decoherence rates

$$\gamma_{ij}^{\text{MM}}(t) = \left(\frac{\boldsymbol{r}_{ij}}{4\sigma^2} + \frac{\mathrm{i}}{2}\boldsymbol{k}_{ij}\right) \cdot \frac{\mathrm{d}\boldsymbol{r}_{ij}(t)}{\mathrm{d}t} + \left(\sigma^2 \boldsymbol{k}_{ij} + \frac{\mathrm{i}}{2}\boldsymbol{r}_{ij}\right) \cdot \frac{\mathrm{d}\boldsymbol{k}_{ij}(t)}{\mathrm{d}t}, \tag{6.138}$$

which for a given relative acceleration $\boldsymbol{a}_{ij}$ between the states, and written in terms of relative velocity instead of wavenumber, is

$$\gamma_{ij}^{\text{MM}}(t) = \frac{1}{4\sigma^2}\boldsymbol{r}_{ij}(t) \cdot \boldsymbol{v}_{ij}(t) + \frac{\sigma^2 m^2}{\hbar^2}\boldsymbol{v}_{ij}(t) \cdot \boldsymbol{a}_{ij}(t) + \mathrm{i}\frac{m}{2\hbar}\left(|\boldsymbol{v}_{ij}(t)|^2 + \boldsymbol{r}_{ij}(t) \cdot \boldsymbol{a}_{ij}(t)\right), \tag{6.139}$$

where $\boldsymbol{r}_{ij}(t) = \boldsymbol{r}_i(t) - \boldsymbol{r}_j(t)$ and $\boldsymbol{v}_{ij}(t) = \boldsymbol{v}_i(t) - \boldsymbol{v}_j(t)$ are the relative position and velocity of the two states, and $\boldsymbol{a}_{ij}(t)$ is their relative acceleration:

$$\boldsymbol{a}_{ij}(t) = -\frac{1}{m}\left(\nabla V_i(\boldsymbol{r}_i, t) - \nabla V_j(\boldsymbol{r}_j, t)\right). \tag{6.140}$$

Note this is slightly different to the relative acceleration used in the Markovian decoherence calculation in Section 6.5.4. Here, we actually have (approximate) positions $\boldsymbol{r}_i(t)$ and $\boldsymbol{r}_j(t)$ for both states' wavepackets, and so we can evaluate the potential gradient separately at the two positions rather than having to use the position of the state currently selected by the hidden variable for both.

Equation (6.5.5) is our minimal-memory decoherence rate, and requires as input the centre-of-mass position and velocity of each Gaussian wavepacket as input. Let's now move on to how these trajectories are computed.

From many trajectories, one

As mentioned, this method track only one trajectory per state. However, it continuously considers the spawning of new trajectories, and rather than actually tracking multiple trajectories per state or probabilistically replacing the existing trajectories with the new ones, it simply averages the trajectories together, weighted by their quantum probabilities. At the end of each timestep, positions and velocities (other than those of the main trajectory) are updated according to a weighted sum of their present values and those of the main trajectory:

$$\boldsymbol{r}_{i\neq\eta}(t') \leftarrow Q_{i\eta}(t', t)\,\boldsymbol{r}_\eta(t') + \left(1 - Q_{i\eta}(t', t)\right)\boldsymbol{r}_i(t'), \tag{6.141}$$

$$\begin{aligned}\boldsymbol{v}_{i\neq\eta}(t') \leftarrow\ & Q^{\text{time}}_{i\eta}(t', t)\boldsymbol{v}_\eta(t') + \tilde{Q}^{\text{space}}_{i\eta}(t', t)(\boldsymbol{v}_\eta(t') + \Delta\boldsymbol{v}_{i\eta}(t')) \\ & + \left(1 - Q^{\text{time}}_{i\eta}(t', t) - \tilde{Q}^{\text{space}}_{i\eta}(t', t)\right)\boldsymbol{v}_i(t'),\end{aligned} \tag{6.142}$$

where $Q_{i\eta}(t', t) = P_{i\eta}(t', t)/|c_i(t')|^2$ is the fraction of the population[22] of state i at time t' that flowed to it from state η in the interval t to t', with $Q^{\text{time}}_{i\eta}(t', t)$ and $\tilde{Q}^{\text{space}}_{i\eta}(t', t)$ defined similarly in terms of $P^{\text{time}}_{i\eta}$ and $\tilde{P}^{\text{time}}_{i\eta}$ as defined in Section 6.4.3, and where $\Delta\boldsymbol{v}_{i\eta}(t')$ is the required velocity correction for a transition from state η to state i at time t', as discussed in Section 6.4.3:

$$\Delta\boldsymbol{v}_{i\eta}(\boldsymbol{r}, t') = \left[\text{sgn}(\boldsymbol{v}\cdot\hat{\boldsymbol{d}}_{i\eta})\sqrt{(\boldsymbol{v}\cdot\hat{\boldsymbol{d}}_{i\eta})^2 + \frac{2}{m}\left(V_i(\boldsymbol{r}, t') - V_\eta(\boldsymbol{r}, t')\right)} - (\boldsymbol{v}\cdot\hat{\boldsymbol{d}}_{i\eta}),\right]\hat{\boldsymbol{d}}_{i\eta}, \tag{6.143}$$

where $\boldsymbol{r} = \boldsymbol{r}_\eta(t')$ and $\boldsymbol{v} = \boldsymbol{v}_\eta(t')$ are the position and velocity of the main trajectory at time t', and the non-adiabatic coupling unit vector $\hat{\boldsymbol{d}}_{i\eta} = \hat{\boldsymbol{d}}_{i\eta}(\boldsymbol{r}_\eta(t'))$ is evaluated at the position along the main trajectory at time t'.

The above scheme is constructed to track a mean trajectory for each state, where 'mean' is defined as a weighted sum of positions and velocities with those of the main trajectory whenever transitions from the latter occur, taking into account velocity jumps and frustrated transitions, with the energy conservation calculation for the velocity jumps based on the potentials at the location of the main trajectory.

[22] Note that $|c_i(t')|^2$ is the i^{th} population at time t' due to unitary evolution only over the time interval, without taking into account decoherence, since all population flows are computed from unitary evolution only and decoherence is treated separately.

6.5.6 What are we 'following' exactly?

Now that we've worked out a way to approximate the result of projecting the multi-component wavefunction onto a specific motional state (a Gaussian wavepacket) without resetting all motional states (also Gaussian wavepackets) to be equal to the one we are

projecting onto, why not take the process further? We can use the same method to project onto any wavefunction we like, in order to inspect what the multi-component wavefunction looks like projected onto the wavefunction in question. This does not have to imply anything about the form of the motional states, which remain Gaussians at all times. It just defines what we 'see' at each timestep of our simulation.

For example, if we projected onto a Dirac delta, that would tell us what the multi-component wavefunction looks like at a specific point in space, say, the centre-of-mass position of the wavepacket corresponding to the hidden variable.

This is appealing for several reasons. Firstly, consider the projected Hamiltonian $\hat{H}_\eta(t)$ first introduced in Section 6.5.2. Since we do not want to have to actually construct Gaussian wavepackets in order to compute the integral that defines the projected Hamiltonian, our only recourse is to approximate the projected Hamiltonian as the value of the full Hamiltonian at the centre-of-mass position of the main trajectory's Gaussian wavepacket:

$$\begin{aligned}\langle\chi_i|\hat{H}_\eta(t)|\chi_j\rangle &= \langle\chi_i\,\phi_\eta(t)|\hat{H}(t)|\chi_j\,\phi_\eta(t)\rangle && (6.144)\\ &\approx \langle\chi_i\,\boldsymbol{r}_\eta(t)|\hat{H}(t)|\chi_j\,\boldsymbol{r}_\eta(t)\rangle && (6.145)\\ &\equiv H_{ij}(\boldsymbol{r}_\eta, t), && (6.146)\end{aligned}$$

which can be considered a small-wavepacket approximation, as mentioned in Section 6.1 in the context of the Ehrenfest semiclassical model. If an arbitrary position is used in place of $\boldsymbol{r}_\eta$, this results in an operator valued function of space $\hat{H}(\boldsymbol{r}, t)$, which operates only on the internal degrees of freedom of the atom.

This is the operator we actually use in the algorithm, and with it, there are no longer any Gaussian wavepackets to compute integrals over or anything else—all details of the wavepackets are encapsulated and parametrised by the approximations and analytics in this and the preceding sections, leaving us to focus on the classical dynamics of the atoms' centre-of-mass motion and the quantum evolution of their internal states according to $\hat{H}(\boldsymbol{r}, t)$ evaluated at the classical positions (representing the centre-of-mass position of the wavepackets) of the atoms.

Although this can be considered a small wavepacket approximation, we cannot use arbitrarily small wavepacket sizes in our decoherence calculations, as the decoherence rates (under both the Markovian and auxiliary trajectories approaches) do not converge to a constant as the wavepacket sizes decrease—rather they become arbitrarily large. Very high spin decoherence rates have the potential to prevent spin flips altogether, also a result of the quantum Zeno effect, since high decoherence rates imply strong measurement, and strong measurement of an observable prevents evolution of that variable.[23] So there is a possible inconsistency here: can we make the wavepackets small or not?

[23] I have sometimes wondered if there is a way to harness this to prevent Majorana losses in evaporation to BEC, by measuring very strongly whether they have occurred.

On the other hand, note in Figure 6.6 the trend in the Markovian decoherence rate as wavepackets become large: the decoherence rate becomes large as well. The expression for the minimal-memory non-Markovian decoherence rate (6.139) also yields a large decoherence rate for large wavepackets. In both cases this is decoherence due to separation in velocity space. While this is the correct result given our assumptions, it causes problems when combined with the above method of approximating the projected Hamiltonian. Essentially, rapid velocity-induced decoherence occurs when the wavepackets are large because spatially large Gaussians are small in velocity space, and hence do not need to accelerate much to be no longer overlapping in velocity space. But do we really expect our wavepackets to be minimal uncertainty wavepackets? Certainly not when they are large compared to the structure of a spatially-varying Hamiltonian. A real wavepacket, whilst undergoing a transition to another state, does not do so everywhere in space the same, as in a single-mode approximation, but our formulation so far assumes it does. In the limit of a very large spin-$\frac{1}{2}$ wavepacket undergoing a complete Majorana spin flip due to a magnetic field zero for example, would appear, as its centre-of-mass passed over the zero,

as approximately two half-Gaussians, one in each state. It is doubtful that such spatially complex wavepackets have narrow enough velocity distributions to lead to decoherence rates as high as the expressions we've derived so far suggest. Narrow velocity distributions are therefore inconsistent with the other assumptions of our model and any consequences of rapid velocity separation in the model should be viewed with scepticism.

Another argument is that, if two wavepackets are co-located in space but not in velocity space, the amplitude of one of them as seen from the centre-of-mass position of the other is not zero. It's the fact that their relative phase varies so much over the extent of the wavepackets that causes their projection onto each other to be small when their velocities are very different.[24] So it's only after this averaging process—integrating over all space—that the wavepackets look like they are small from each other's perspective. One of the common themes of this method as a whole is that we are often happy with *representative* results rather than average ones, drawing results from approximately correct statistical distributions, and only observing the complete distribution when we analyse a large sample of results. Continuing in this vein, why not consider the value of the multi-component wavefunction along a specific trajectory in space, rather than its projection onto a Gaussian? The projection of a multi-component wavefunction onto a Gaussian isn't any more useful or intuitive a quantity than tracing a curve through spacetime and asking "what is the wavefunction here?" This way, instead of a simulation telling us "There was, on average, no population transfer to such-and-such state", it would give a (more likely to be useful in my opinion) answer "there was a lot of population transfer to this state, but the phase is essentially random".

[24] Some call this 'dephasing' to distinguish from other forms of decoherence.

The final argument is empirical: attempts to implement the model with the Gaussian projections do not work well unless wavepackets are small. The decoherence rates are too high, presumably for the speculative reasons discussed above. This damps the population of states other than that selected by the hidden variable too quickly, such that the simulated population of a state is zero even though according to the Schrödinger wave equation it should not be (it merely has an inhomogeneous phase). Simply deleting the terms caused by velocity separation from the decoherence rate expressions improves the results somewhat. But if projecting onto a Dirac delta instead, there are no velocity-separation terms to argue for the deletion of, and one obtain modestly better results in any case (as shown in Section 6.7).

Shenvi *et al.* [174] also seem to find the velocity-separation term of their decoherence factor troublesome, and opt to remove it by gauge transforming the wavepackets to have the same velocity. They argue that because at least a small initial velocity difference between wavepackets is required by energy conservation (i.e. the velocity jumps from Section 6.4.3), that it is 'unfair' to have an initial sudden drop in coherence because of this. I agree that an immediate drop in coherence upon a transition is strange, but this is small compared to the decoherence that large Gaussian wavepackets undergo upon further acceleration away from each other, which also seems unphysically large and needs to be addressed as well. But velocity jumps or not, these are the actual decoherence factors between pairs of Gaussians, and if we don't like the answer, perhaps the projection of the total wavefunction onto a Gaussian isn't the question we meant to ask. Perhaps we should be asking simply what value a multi-component wavefunction has at a specific point in space. Shenvi *et al.* gauge transform away the entire velocity difference between the wavepackets, not only the part of it mandated by energy conservation, leading me to suspect that some of the same concerns I outlined above occurred to them too, or that they simply observed, like me, that the velocity separation was responsible for what appeared to be unphysically large decoherence rates in their results.

In light of all this, I now redefine 'follow' to mean "look at the wavefunction's value at the centre-of-mass position of the classical trajectory corresponding to the state being followed", the result of which can be computed by projecting the total wavefunction onto

a Dirac delta, resulting in a decoherence factor

$$R_{ij}^{\text{DD}}(t) = \langle \boldsymbol{r}_i(t) | \phi_j(t) \rangle \tag{6.147}$$

where $|\phi_j(t)\rangle$ is one of our usual Gaussian motional states and DD stands for 'Dirac delta'. Following identical arguments to the previous two sections, we obtain Dirac-delta decoherence rates for the Markovian case as

$$\gamma_{ij}^{\text{Markov DD}}(\boldsymbol{r}, t) \approx \frac{1}{2\Gamma(\frac{5}{4})} \sqrt{\frac{|\boldsymbol{a}_{ij}(\boldsymbol{r}, t)|}{\sigma}} + \mathrm{i} \frac{1}{2\Gamma(\frac{5}{4})^2} \frac{m\sigma |\boldsymbol{a}_{ij}(\boldsymbol{r}, t)|}{\hbar}, \tag{6.148}$$

which is identical to the position-only approximation (6.128) up to a factor of $2^{\frac{1}{4}}$, and for minimal-memory non-Markovian case as

$$\gamma_{ij}^{\text{MMDD}}(t) = \frac{1}{2\sigma^2} \boldsymbol{r}_{ij}(t) \cdot \boldsymbol{v}_{ij}(t) - \mathrm{i} \frac{m}{\hbar} \boldsymbol{r}_{ij}(t) \cdot \boldsymbol{a}_{ij}(t), \tag{6.149}$$

with all symbols as defined in Section 6.5.4 and Section 6.5.5, respectively.

Now that we are projecting onto a Dirac delta instead of a Gaussian, calculation of the projected Hamiltonian as simply the value of the full Hamiltonian at a specific point in space is no longer a contradiction, as it is exactly what we would get if we projected the full Hamiltonian onto a Dirac delta instead of a Gaussian.

6.6 Algorithms

Now we have presented all the pieces from which the two versions of the hidden-variable semiclassical method are constructed. In this section I present step-by-step instructions for implementing the two versions of the method—one using Dirac-delta Markovian decoherence, and the other using auxiliary trajectories with Dirac-delta minimal-memory decoherence.

6.6.1 Markovian hidden-variable semiclassical method

This is the simplest version of my method, using crude Markovian decoherence as discussed in Section 6.5.4, and simulating only one trajectory corresponding to the centre-of-mass motion of the state corresponding to the hidden variable at each moment in time.

State variables

The Markovian variant of the hidden-variable semiclassical method has the following state variables describing the state of each simulated atom at each moment of time:

- The internal state vector of the atom $|\tilde{\chi}(t)\rangle$, in any chosen basis (though the local eigenbasis is likely to be computationally simplest to perform calculations in).
- The position $\boldsymbol{r}(t)$ and velocity $\boldsymbol{v}(t)$ of the atom.
- The hidden variable $\eta(t)$, equal to an integer index specifying one of the eigenstates $|\chi_\eta(\boldsymbol{r}, t)\rangle$ of the Hamiltonian $\hat{H}(\boldsymbol{r}, t)$ atom in the local energy eigenbasis.

Initial conditions

Strictly, the hidden-variable semiclassical method is intended to simulate the classical centre-of-mass motion of thermal wavelength sized Gaussian wavepackets. Therefore the initial conditions for the positions and velocities of the atoms in an ensemble are arbitrary, classical initial conditions, and so one might draw them, along with the internal states, randomly from a Boltzmann distribution for a given confining potential and temperature, such that the initial internal state of each atom is an eigenstate $|\chi_i(\boldsymbol{r}, t = 0)\rangle$. In this case the initial value $\eta(t = 0)$ of the hidden variable for each atom should correspond to its chosen eigenstate.

However, one might want to convert an initial *wavefunction* into an ensemble of positions and velocities, in which case one can draw positions and momenta from the quantum probability distributions $\langle\psi(t = 0)|\hat{\boldsymbol{r}}|\psi(t = 0)\rangle$ and $\langle\psi(t = 0)|\hat{\boldsymbol{p}}|\psi(t = 0)\rangle$ for an initial motional state vector $|\psi\rangle$. Similarly, one can set the initial internal state vector of each atom to an arbitrary desired internal state vector appropriate for the problem at hand, in which case the initial values of the hidden variable for each atom in an ensemble being simulated should be chosen randomly with probability equal to the internal state populations $|\langle\chi_i(\boldsymbol{r}, t = 0)|\tilde{\chi}(\boldsymbol{r}, t = 0)\rangle|^2$ in the local basis at the position of each atom. More complex initial conditions are possible; there is no reason why one can't draw positions, velocities and internal states from arbitrary joint probability distributions, or use hand-crafted initial states, provided that the hidden variables are chosen randomly at the end with probabilities equal to the internal state populations in the local basis of each atom so as to be consistent with the Born rule.

Evolution

Evolving the Markovian hidden-variable semiclassical method involves three coupled differential equations, a stochastic evolution rule, and some additional steps to be performed in between numerical integration steps. Because the modifications in between steps can be discontinuous, multi-step integration schemes that retain information from previous integration steps should be avoided. One step of the Markovian hidden-variable semiclassical method comprises the following steps to evolve the state variables from time t to $t' = t + \Delta t$, where Δt is small compared to dynamical timescales of the problem.

1. Evolve the internal state vector and motional state variables of each atom from time t to t' according to the following coupled differential equations, using one or more steps of your favourite non-multi-step numerical integration method:

$$\frac{\mathrm{d}}{\mathrm{d}t}|\tilde{\chi}(t)\rangle = -\frac{\mathrm{i}}{\hbar}\hat{H}(\boldsymbol{r}, t)\,|\tilde{\chi}(t)\rangle - \hat{\Gamma}_\eta^{\text{Markov DD}}(\boldsymbol{r}, t)\,|\tilde{\chi}(t)\rangle\,, \tag{6.150}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{v}(t) = -\frac{1}{m}\nabla V_\eta(\boldsymbol{r}, t), \tag{6.151}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{r}(t) = \boldsymbol{v}(t), \tag{6.152}$$

   where $V_\eta(\boldsymbol{r}, t)$ is the (space- and/or time-dependent) eigenvalue of $\hat{H}(\boldsymbol{r}, t)$ corresponding to the eigenstate $|\chi_\eta(\boldsymbol{r}, t)\rangle$, and the Markovian decoherence operator is

$$\hat{\Gamma}_\eta^{\text{Markov DD}}(\boldsymbol{r}, t) = \sum_i |\chi_i(\boldsymbol{r}, t)\rangle\langle\chi_i(\boldsymbol{r}, t)|\,\gamma_{\eta i}^{\text{Markov DD}}(\boldsymbol{r}, t), \tag{6.153}$$

   where $\gamma_{ij}^{\text{Markov DD}}(\boldsymbol{r}, t)$ are the approximate Markovian decoherence rates given by (6.148). The integration of the internal state can be performed in the local eigenbasis using the transformed Hamiltonian defined in (6.38), in which case the decoherence operator is diagonal with the i^{th} diagonal equal to $\gamma_{\eta i}^{\text{Markov DD}}(\boldsymbol{r}, t)$.

2. Normalise the internal state vector to unit norm:

$$|\tilde{\chi}(t')\rangle \leftarrow \frac{|\tilde{\chi}(t')\rangle}{\sqrt{\langle\tilde{\chi}(t')|\tilde{\chi}(t')\rangle}} \tag{6.154}$$

3. Evaluate the S^{time}, S^{space} and S^{stay} matrices over this time interval for Tully's minimal-switches algorithm as described in Section 6.4.2, equations (6.60–6.62). This may require numerical differentiation of the basis vectors of the Hamiltonian with respect to space and time, and possibly numerical diagonalisation of the Hamiltonian to obtain the basis vectors as functions of space and time, if analytic expressions are not known. If Δt is not small enough to be considered infinitesimal for the purposes of evaluating (6.53) and (6.54), and one numerically integrates them for a more accurate result, note that these expressions require the state populations $\{c_i(t)\}$ in the local basis in the *absence* of decoherence; one therefore cannot do this integral in tandem with integrating the above differential equation for the internal state.[25] Alternately, compute the overall S matrix of your favourite hidden-variable theory for evolution over the given time interval in the local eigenbasis (similarly keeping in mind that the effective unitary given to the hidden-variable theory must be due to the evolution of $\hat{H}_{\text{eff}}(t)$ only, and not the decoherence term). If it saves computational power to do so, only compute column η of S, since only transitions from state η to other states need be considered.

[25] Though with split-operator methods and some care, the decoherence term can be treated separately such that the two integrals can be evaluated without repeating calculations unnecessarily.

4. Compute $\tilde{P}^{\text{space}}$ (6.72) and hence $\tilde{S}^{\text{space}}$ (6.74) to set the probability of classically disallowed transitions to zero as described in Section 6.4.3.

5. Modify the hidden variable with probability

$$\Pr\left(\eta \overset{\text{space}}{\rightarrow} i\right) = \tilde{S}^{\text{space}}_{i\eta} \tag{6.155}$$

that it transitions to the state ($i \neq \eta$) via a spatially induced transition, and probability

$$\Pr\left(\eta \overset{\text{time}}{\rightarrow} i\right) = S^{\text{time}}_{i\eta} \tag{6.156}$$

that it transitions to a state ($i \neq \eta$) via a temporally induced probability, and probability

$$\Pr(\eta \rightarrow \eta) = 1 - \sum_i \left(\tilde{S}^{\text{space}}_{i\eta} + S^{\text{time}}_{i\eta}\right) \tag{6.157}$$

that it keeps its current value. These choices can be randomly chosen between with the correct probabilities as described in Figure 6.4.

6. If the hidden variable did transition to a different state $i \neq \eta$, and if it was a spatially induced transition, instantaneously adjust the atom's velocity as required by energy conservation:

$$\boldsymbol{v}(t') \leftarrow \boldsymbol{v}(t') + \Delta\boldsymbol{v}_{\eta i}(\boldsymbol{r}, t), \tag{6.158}$$

where $\Delta\boldsymbol{v}_{\eta i}(\boldsymbol{r}, t)$ is given by (6.76).

This completes the Markovian hidden-variable semiclassical method.

6.6.2 Mean auxiliary trajectories hidden-variable semiclassical method

This is the more complex version of my method. Additional classical trajectories are propagated—one for each internal state of each atom—but other than classically evolving these additional trajectories, the increase in computational cost is minimal.

State variables

The non-Markovian/Mean auxiliary trajectories variant of the hidden-variable semiclassical method has the following state variables describing the state of each simulated atom at each moment of time:

- The internal state vector of the atom $|\tilde{\chi}(t)\rangle$, in any chosen basis (though the local eigenbasis is likely to be computationally simplest to perform calculations in).
- A set of positions $\{\boldsymbol{r}_i(t)\}$ and velocities $\{\boldsymbol{v}_i(t)\}$, one position and velocity for each internal state of the atom.
- The hidden variable $\eta(t)$, equal to an integer index specifying one of the eigenstates $|\chi_\eta(\boldsymbol{r}_\eta, t)\rangle$ of the Hamiltonian $\hat{H}(\boldsymbol{r}_\eta, t)$ atom in the local energy eigenbasis at the location $\boldsymbol{r}_\eta$ of the main trajectory.

Initial conditions

The manner of generating initial conditions of the mean auxiliary trajectories variant of the hidden-variable semiclassical method is identical to that of the Markovian variant, with the exception that multiple initial positions and velocities are required instead of just one. The initial positions and velocities of all auxiliary trajectories can however simply be set equal to those of the main trajectory for each atom.

Evolution

The evolution of the state variables for each atom from time t to $t' = t + \Delta t$ in the non-Markovian hidden-variable semiclassical method proceeds similarly to the Markovian variant.

1. Evolve the internal state vector and set of motional state variables of each atom from time t to t' according to the following coupled differential equations, using one or more steps of your favourite non-multi-step numerical integration method:

$$\frac{\mathrm{d}}{\mathrm{d}t}|\tilde{\chi}(t)\rangle = -\frac{\mathrm{i}}{\hbar}\hat{H}(\boldsymbol{r}_\eta, t)\,|\tilde{\chi}(t)\rangle - \hat{\Gamma}_\eta^{\text{MMDD}}(t)\,|\tilde{\chi}(t)\rangle\,, \tag{6.159}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{v}_i(t) = -\frac{1}{m}\nabla V_i(\boldsymbol{r}_i, t), \tag{6.160}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{r}_i(t) = \boldsymbol{v}_i(t), \tag{6.161}$$

   where $V_i(\boldsymbol{r}_i, t)$ is the (space- and/or time-dependent) eigenvalue of $\hat{H}(\boldsymbol{r}_i, t)$ corresponding to the eigenstate $|\chi_i(\boldsymbol{r}_i, t)\rangle$ at the position of the i^{th} trajectory, which may be the main ($i = \eta$) or an auxiliary ($i \neq \eta$) trajectory; and the non-Markovian decoherence operator is

$$\hat{\Gamma}_\eta^{\text{MMDD}}(t) = \sum_i |\chi_i(\boldsymbol{r}_\eta, t)\rangle\langle\chi_i(\boldsymbol{r}_\eta, t)|\,\gamma_{\eta i}^{\text{MMDD}}(t), \tag{6.162}$$

   where $\gamma_{ij}^{\text{MMDD}}(t)$ are the minimal-memory non-Markovian decoherence rates given by (6.149); $\gamma_{\eta j}^{\text{MMDD}}(t)$ comprising the diagonals of $\hat{\Gamma}_\eta^{\text{MMDD}}(t)$ if working in the local eigenbasis along the main trajectory. When computing $\gamma_\eta^{\text{MMDD}}(t)$, clip its real part from below to zero. Although such recoherence can in principle be physically meaningful, it can arise out of tiny amplitudes of states with auxiliary trajectories far from the main trajectory, and is hence subject to large floating-point error.[26] I have not observed this problem with the Dirac-delta based decoherence rates, but it does seem to appear if the Gaussian-projection based decoherence rates are used.

[26] Large error in *positive* (real parts of) decoherence rates in the same context is not a problem, as the state amplitudes are already very close to zero, so the accuracy of the rate at which they more closely approach zero is of no consequence.

2. Normalise the internal state vector to unit norm:

$$|\tilde{\chi}(t')\rangle \leftarrow \frac{|\tilde{\chi}(t')\rangle}{\sqrt{\langle\tilde{\chi}(t')|\tilde{\chi}(t')\rangle}} \tag{6.163}$$

3. Evaluate the S^{time}, S^{space} and S^{stay} matrices as in Step 3 for the Markovian case

4. Compute $\tilde{S}^{\text{space}}$ as in Step 4 for the Markovian case. We treat transitions to have occurred at the location of the originating trajectory, therefore for the purposes of computing $\tilde{S}^{\text{space}}$, the expressions for $\Delta v_{ij}^2(\boldsymbol{r}, t)$ (6.71) and $\hat{\boldsymbol{d}}_{ij}(\boldsymbol{r}, t)$ (6.69) must be evaluated using $\boldsymbol{r} = \boldsymbol{r}_i(t')$, and the conditional in $\tilde{P}^{\text{space}}$ (6.72) evaluated with $\boldsymbol{v} = \boldsymbol{v}_i(t')$. Since only transitions from the current state specified by the hidden variable to other states need to be considered, one only needs to compute column η of $\tilde{S}^{\text{space}}$, in which case only $\boldsymbol{r}_\eta$ and $\boldsymbol{v}_\eta$ are necessary. This is then identical to the non-Markovian case, in which the main trajectory is the only trajectory simulated.

5. Compute the energy-conservation mandated velocity difference $\Delta\boldsymbol{v}_{\eta i}$ given by (6.76) for all $i \neq \eta$. As with Step 4, since we treat transitions to have occurred at the location of the main trajectory, the velocity difference expression should be evaluated using the position and velocity of the main trajectory, as emphasised in (6.143).

6. Instantaneously modify the positions and velocities of all auxiliary trajectories ($i \neq \eta$) according to (6.141) and (6.142). This in essence spawns new auxiliary trajectories at the location of the main trajectory, but does not keep them: the averaging expressions replace each auxiliary trajectory with a weighted sum of its previous position and velocity and the position and velocity of the new trajectory. If the population of the state corresponding to a given auxiliary trajectory falls below some threshold (I use $|c_{i\neq\eta}(t)|^2 < 10^{-8}$), then discard the trajectory and replace it with the position and velocity of the main trajectory.

7. Probabilistically make a transition of the hidden variable as in Step 5 for the Markovian case.

8. In the case that the hidden variable transitioned, replace the trajectory of the new state with the position and velocity of the previous main trajectory. In the case that it was a spatially-induced non-adiabatic transition, adjust the velocity of the new state's trajectory with the previously computed $\Delta\boldsymbol{v}_{\eta\xi}$, where η is the previous value of the hidden variable and ξ is the new value.

This completes the mean auxiliary trajectories hidden-variable semiclassical method.

6.7 Results

Here I demonstrate the two variants of the method, each for two different spatially dependent Hamiltonians in one dimension, and compare them to the numerical result from the full Schrödinger wave equation. All simulations are for ^{87}Rb atoms in the $F = 1$ ground state subject to the Zeeman Hamiltonian for a given magnetic field profile. In scenario I, atoms in the trapped state (for ^{87}Rb, the locally spin-down sate is magnetically trappable) pass over a field minimum, which induces Majorana transitions, allowing the spin-flipped atoms to escape. In scenario II, the magnetic field profile is modified to create two closely spaced field minima. This scenario demonstrates the importance of accurate decoherence in surface-hopping/hidden-variable semiclassical methods, as the two regions of non-adiabatic coupling are encountered in a time comparable to the decoherence timescale where the exact form of the decoherence model can strongly

influence the outcome of transitions. I use Schrödinger theory for the hidden-variable theory throughout. Since the Hamiltonians are time-independent, as mentioned these results do not test my proposed method of conditionally applying velocity jumps to transitioned atoms—velocity jumps are always applied.

I simulated both scenarios first using the Schrödinger wave equation, as the gold-standard against which to benchmark the other methods. Numerically I solved for the three-component wavefunction using fourth order Runge–Kutta (Section 3.3) and the Fourier method of evaluating spatial derivatives (Section 3.4.1). I added imaginary potentials near the boundaries of the region I wished to simulate to absorb untrapped wavepackets rather than have them reflect or wrap around due to periodic boundary conditions inherent in the Fourier method of derivatives. The initial condition was a Gaussian wavepacket in the locally spin-down eigenstate, $m_F = -1$, though I simulated in the fixed eigenbasis of $\hat{F}_z$, transforming as necessary to create the initial conditions, and later to compare state populations in the local eigenbasis.

I then simulated both scenarios with 10^4 semiclassical atoms using the Ehrenfest semiclassical method (Section 6.1.1), the Markovian hidden-variable semiclassical method (Section 6.6.1), and the mean auxiliary trajectories hidden-variable semiclassical method (Section 6.6.2). The initial conditions were set by drawing positions and velocities randomly for each semiclassical particle from the position and velocity probability distributions implied by the Gaussian wavepacket.

For both semiclassical methods I used simple first-order split-step evolution (Section 3.2.4) for the quantum evolution in the local eigenbasis, and evolved the classical trajectories analytically using constant acceleration over each timestep:

$$|\tilde{\chi}_H(t')\rangle \approx \mathrm{e}^{-\hat{I}_\eta(t)\Delta t}\hat{U}_H(z',t')\,\hat{U}_H^\dagger(z,t)\,\mathrm{e}^{-\frac{\mathrm{i}}{\hbar}\hat{V}(z,t)\Delta t}\,|\tilde{\chi}_H(t)\rangle\,, \tag{6.164}$$

$$z' \approx z - \frac{\Delta t^2}{2m}\nabla V_\eta(z) \tag{6.165}$$

$$v' \approx v - \frac{\Delta t}{m}\nabla V_\eta(z), \tag{6.166}$$

where z, v and t are the position, velocity and time at the start of a timestep, z', v' and t' at the end of the timestep, and $\Delta t = t' - t$, is the timestep, which was $\Delta t = 500$ ns. These methods are not particularly accurate, but were sufficient for these demonstrations and allowed the implementation to be simple. In particular, the ordering of the individual operators for evolving the state vector over a timestep was chosen to allow everything to the right of the decoherence factor be the unitary part of the evolution for the timestep, which could be input to Schrödinger theory as the unitary describing evolution in the local basis for that timestep.

A summary of all the simulation results presented in this section is given in Table 6.1.

Scenario I

In this scenario, a ^{87}Rb atom begins as an initially stationary Gaussian wavepacket of width 600 nm (equal to the thermal de Broglie wavelength at $T = 0.2\,\mu$K) centred at $z_0 = -20\,\mu$m in the Zeeman potential experienced by the $F = 1$ ground-state manifold (Section 2.3.3) with the following magnetic field profile:

$$\boldsymbol{B}_{\text{trap}}(z) = (B_x, B_y, B_z) = \left(B_\perp, 0, B'_z z\right) \tag{6.167}$$

where the transverse magnetic field is $B_\perp = 162.5\,$mG and the z-gradient is $B'_z = 250\,\mathrm{G\,cm^{-1}}$. This potential provides a region close to $z = 0$ where the magnetic field rapidly reverses direction too quickly for the atom's spin to adiabatically follow, inducing Majorana transitions. The atom accelerates under the initial potential gradient toward

this region. The adiabatic potentials resulting from the Zeeman Hamiltonian with this magnetic field are shown in Figure 6.7. Results are shown in Figure 6.9 for the Markovian model and Figure 6.10 for the auxiliary trajectories model.

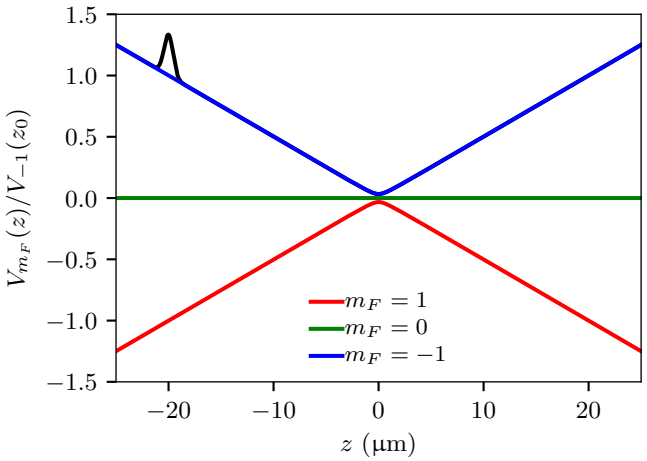

Figure 6.7: Adiabatic potentials for scenario I with a single region of coupling. The initial probability distribution of the atoms is depicted (in arbitrary units), offset for clarity by the adiabatic potential of the initial state.

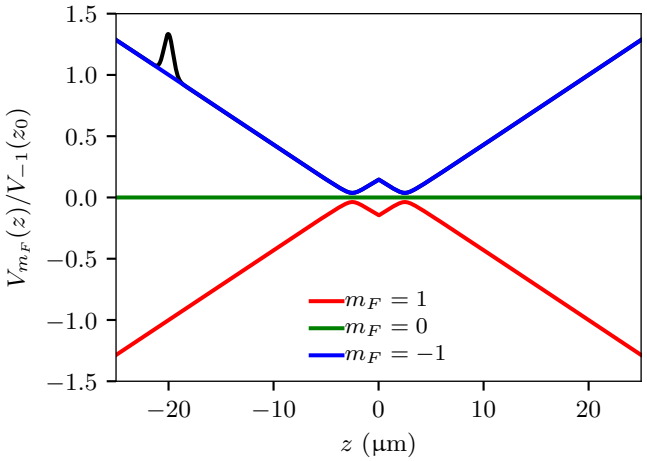

Figure 6.8: Adiabatic potentials for scenario II, with two closely spaced coupling regions. The initial probability distribution of the atoms is depicted (in arbitrary units), offset for clarity by the adiabatic potential of the initial state.

Scenario II

Scenario II is identical to scenario I, except that the magnetic field profile is instead[27]

$$\boldsymbol{B}_{\text{trap}}(z) = (B_x, B_y, B_z) = \begin{cases} \left(B_\perp, 0, B_z'(z + z_{\text{fieldmin}})\right) & z < 0 \\ \left(B_\perp, 0, -B_z'(z + z_{\text{fieldmin}})\right) & z >= 0 \end{cases} \tag{6.168}$$

where $z_{\text{fieldmin}} = 2.5\,\mu\text{m}$ is the distance of each field minimum to the origin, such that the two minima are separated by 5 μm, and with the other variables having the same

[27] This magnetic field profile is unphysical—though similar Hamiltonians can be constructed by means of dressed potentials using fast-rotating fields—it serves as a good test that highlights the improvement of the mean auxiliary trajectories method over the Markovian method.

values as in scenario I. The adiabatic potentials resulting from the Zeeman Hamiltonian with this magnetic field are shown in Figure 6.8. Results are shown in Figure 6.11 for the Markovian model and Figure6.12 for the auxiliary trajectories model.

6.7.1 Gaussian projection results

Here I demonstrate the difference between the above results, which use the Dirac-delta based decoherence rates (Section 6.5.6), with what one obtains if projecting onto a Gaussian instead as originally proposed in Sections 6.5.4 and 6.5.5. Here I only use the mean-auxiliary-trajectories method; though as argued in Section 6.5.6, the Markovian decoherence rate has its problems too when derived assuming Gaussian projections.

For these results I simulated scenario II, in which there are two closely spaced magnetic field minima, using two different decoherence rates. In the first, I take the decoherence rate $\gamma_{ij}^{\text{MM}}(t)$ given by (6.139) at face value, velocity-space separation included. This leads to poor results as the decoherence is clearly too fast, as shown in Figure 6.13.

In the second, I make a similar change to that of Shenvi *et al.* [174], setting $\boldsymbol{k}_{ij} = 0$ for the relative wavenumber of the two Gaussian wavepackets in the decoherence factor (6.95), and otherwise proceed as normal per Section 6.5.5. This gives a minimal-memory decoherence rate

$$\gamma_{ij}^{\text{MM}}|_{\boldsymbol{k}_{ij}=0}(t) = \frac{1}{4\sigma^2}\boldsymbol{r}_{ij}(t)\cdot\boldsymbol{v}_{ij}(t). \tag{6.169}$$

This leads to much better results, shown in Figure 6.14 but still not as good as the Dirac-delta projection based decoherence rate, shown in Figure 6.12.

6.8 Discussion and conclusion

The methods presented in this chapter give acceptable results in different scenarios. The Markovian decoherence model (Section 6.5.4 and modifications in Section 6.5.6) is able to obtain accurate trajectories and state populations for semiclassical atoms in the case of non-adiabatic coupling regions that are encountered at intervals that are large compared to the decoherence time (Figure 6.9), but produces inaccurate populations if multiple coupling regions are encountered at smaller time intervals (Figure 6.11). The mean auxiliary trajectories method (Section 6.5.5 and modifications in Section 6.5.6) solves this problem by more accurately modelling dynamic decoherence curves, and is therefore able to obtain accurate trajectories and state populations even in the presence of multiple regions of coupling encountered at intervals comparable to the decoherence timescale (Figure 6.12). Both methods are improvements over the Ehrenfest semiclassical method, which has no decoherence and no separation of trajectories, and over Tully's original model, which had separation of trajectories but no decoherence.

The problem of velocity separation of wavepackets causing unexpected and undesired additional decoherence is resolved by instead modelling a projected state vector that is a projection of the multi-component wavefunction onto a Dirac delta—rather than a Gaussian wavepacket—equivalent to evaluating the wavefunction at a specific point in space following a classical trajectory over time (Section 6.5.6). The total multi-component wavefunction is still modelled as a collection of Gaussian wavepackets following classical trajectories. This is a more natural way to obtain state vectors that are statistically representative of the underlying multi-component wavefunctions we are approximating. As this is not based on wavepacket overlaps integrals, population in a state with a different phase gradient is not treated as having vanished merely because it has an inhomogeneous phase—instead we obtain some population, and some (perhaps rapidly varying) phase.

The above solution to the velocity separation problem, along with my method of averaging auxiliary trajectories in order to model only one trajectory per internal state

Figure	Scenario	Decoherence	Projection	Δv ignored	Correct populations		Summary
					Total	Local	
Figure 6.9	I	Markovian	Dirac delta	No	✓✓	✗	For a single region of non-adiabatic coupling, Markovian decoherence gives correct total populations despite inaccurate local populations.
Figure 6.10	I	Aux. trajectories	Dirac delta	No	✓✓	✓✓	In addition accurate total populations, Aux. trajectories method also gives accurate local populations for a single region of non-adiabatic coupling.
Figure 6.11	II	Markovian	Dirac delta	No	✗	✗	Scenario II exposes that Markovian decoherence gives inaccurate total populations for two closely spaced non-adiabatic coupling regions.
Figure 6.12	II	Aux. trajectories	Dirac delta	No	✓✓	✓✓	Aux. trajectories decoherence is accurate even for the case of two closely spaced non-adiabatic coupling regions.
Figure 6.13	II	Aux. trajectories	Gaussian	No	✗	✗	Separation in velocity space gives unphysically large decoherence rates when using Gaussian projections.
Figure 6.14	II	Aux. trajectories	Gaussian	Yes	✓	✓	Ignoring the velocity difference between the wavepackets, but otherwise still using Gaussian projection yields better results than including velocity separation, but not as accurate as when using Dirac-delta based projection.

Table 6.1: Summary of results as shown in the six figures following this table, specifying the scenario simulated and methods used for each, and a qualitative indication of the accuracy of the results, when compared to the Schrödinger wave equation. ✓✓ indicates excellent agreement, ✓ indicates good agreement, and ✗ poor agreement. The level of agreement with both the total populations and with local population—i.e. the population along specific classical trajectories—are indicated. A summary of the conclusions drawn from each simulation is given.

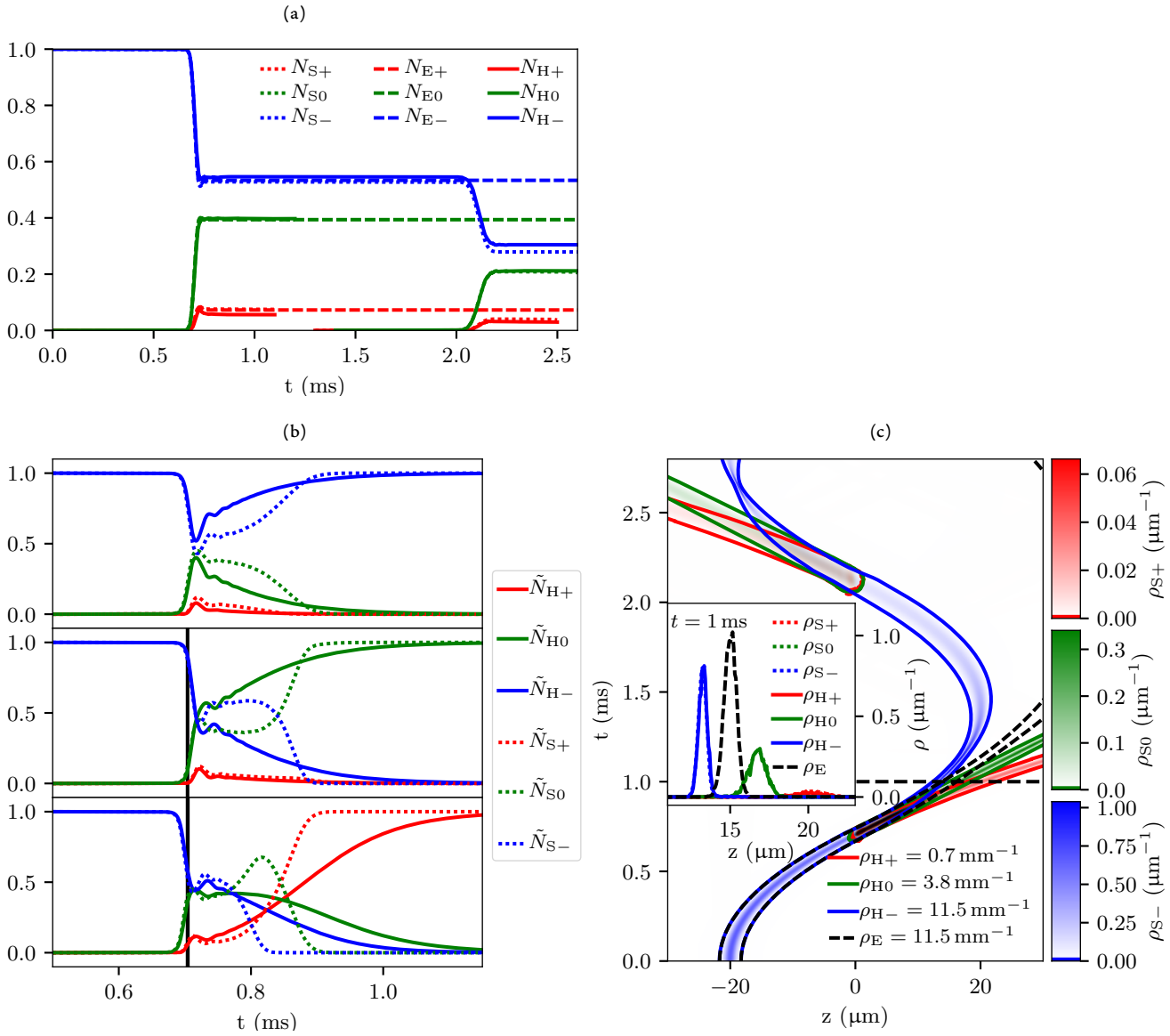

Figure 6.9: Scenario I results for the approximate Markovian decoherence hidden-variable semiclassical method (Section 6.6.1), compared to the Schrödinger wave equation and Ehrenfest semiclassical method. (a) The populations N in each spin projection state (denoted $N_\pm$, N_0 for $m_F = \pm 1$ and $m_F = 0$ respectively) over time, obtained for the Schrödinger wave equation (N_S) by integration in the local basis over all space, for the Ehrenfest semiclassical model (N_E) as an expectation value over all semiclassical atoms' internal state vectors, and for the hidden-variable semiclassical method (N_H) as the proportion of atoms with the hidden variable specifying each state at each moment of time. The lines ending just after 1 ms is due to atoms/amplitude leaving the integration region. There is good agreement between all methods during the first approach of the field minimum, though the Ehrenfest trajectories do not make a second approach within the simulated interval. The remaining deviations between the Schrödinger and hidden-variable curves are within the random variation of the stochastic method. (b) The local state populations along specific trajectories corresponding to classical motion in $m_F = -1$ (top) for all time, switching to $m_F = 0$ at the first field minimum (middle), and to $m_F = 1$ (bottom). These were obtained by interpolation and normalisation of the Schrödinger wave equation results, and by selecting the semiclassical atoms making the chosen hidden-variable transitions closest to the field minimum (black lines). One trajectory resulting from the Ehrenfest model was chosen for comparison, its trajectory is close to those of the other two models until the field minimum, after which it diverges. This plot shows the shape of the decoherence curves and how our exponential approximation—while a poor fit—does have the correct timescale, and is better than the Ehrenfest method's absence of any decoherence. (c) Probability density of trajectories. Schrödinger wave equation results (shaded), and Ehrenfest and hidden-variable semiclassical results shown with a single contour line for a chosen density. Inset: The probability distributions at 1 ms. There is good agreement between the hidden-variable semiclassical and Schrödinger wave equation results.

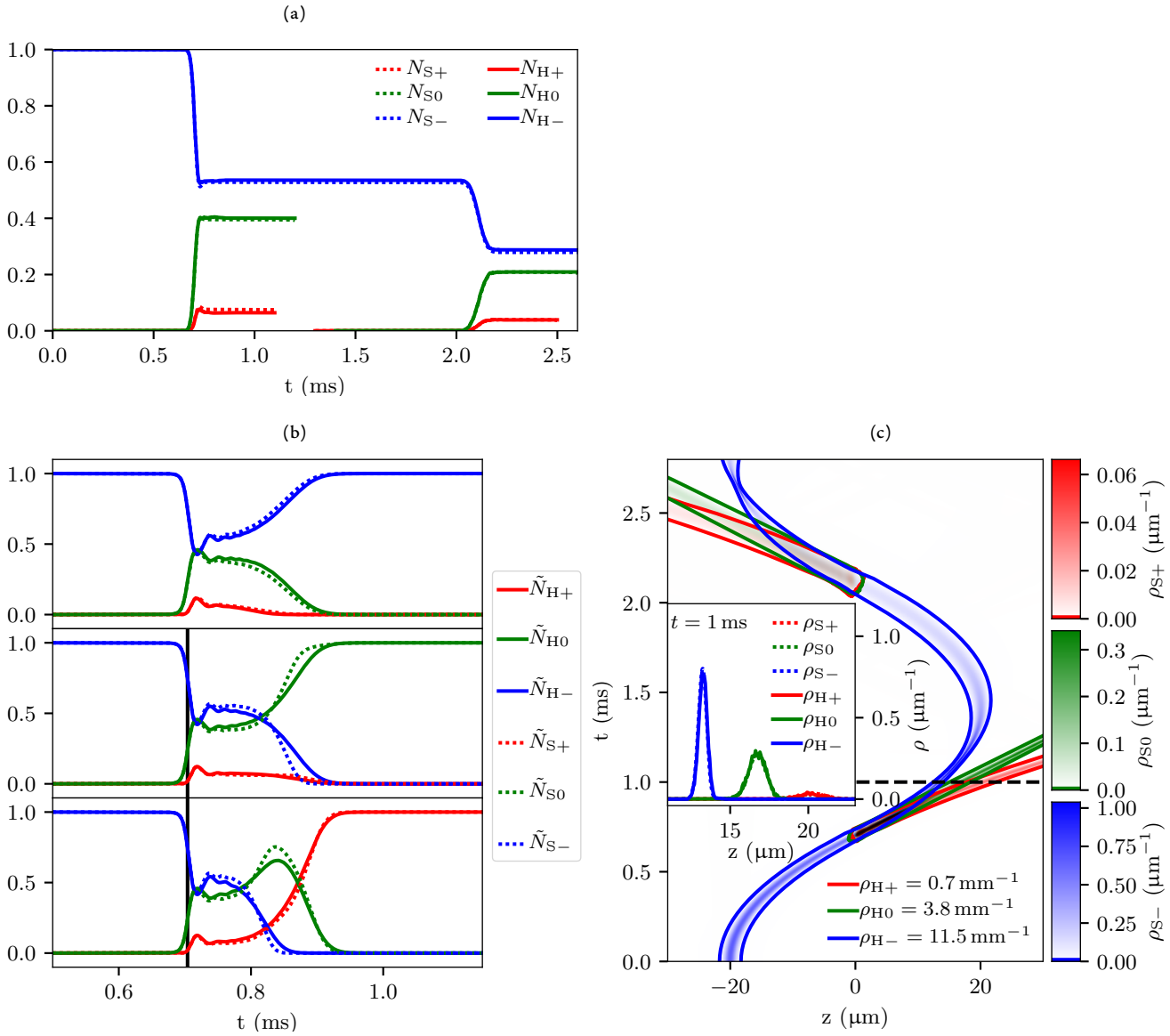

Figure 6.10: Scenario I results for the mean-auxiliary-trajectories hidden-variable semiclassical method (Section 6.6.2) compared to the Schrödinger wave equation (Ehrenfest results excluded from this and subsequent results plots for clarity). (a) As per Figure 6.9. Once again, there is good agreement between all methods during the first approach of the field minimum, and good agreement between the hidden-variable semiclassical method during the second approach, though the Ehrenfest trajectories do not approach a second time. Again, the remaining deviation is within the range expected given the stochastic algorithm and could be reduced by increasing the sample size. (b) Populations after projecting onto specific classical trajectories, as and comparing with semiclassical atoms closest to those trajectories, as per Figure 6.9. Here we see that the decoherence curves traced out by the auxiliary-trajectories method are in much closer agreement with the results of the Schrödinger wave equation than those of the Markovian model. Again, the discrepancies are within the statistical variation of the model; shown in each subplot is the atom that was closest to the given trajectory. (c). Probability density of trajectories of the two methods, as described in Figure 6.9. Once again there is excellent agreement between the Schrödinger wave equation and hidden-variable semiclassical methods, with the Ehrenfest trajectories showing the expected unphysical behaviour.

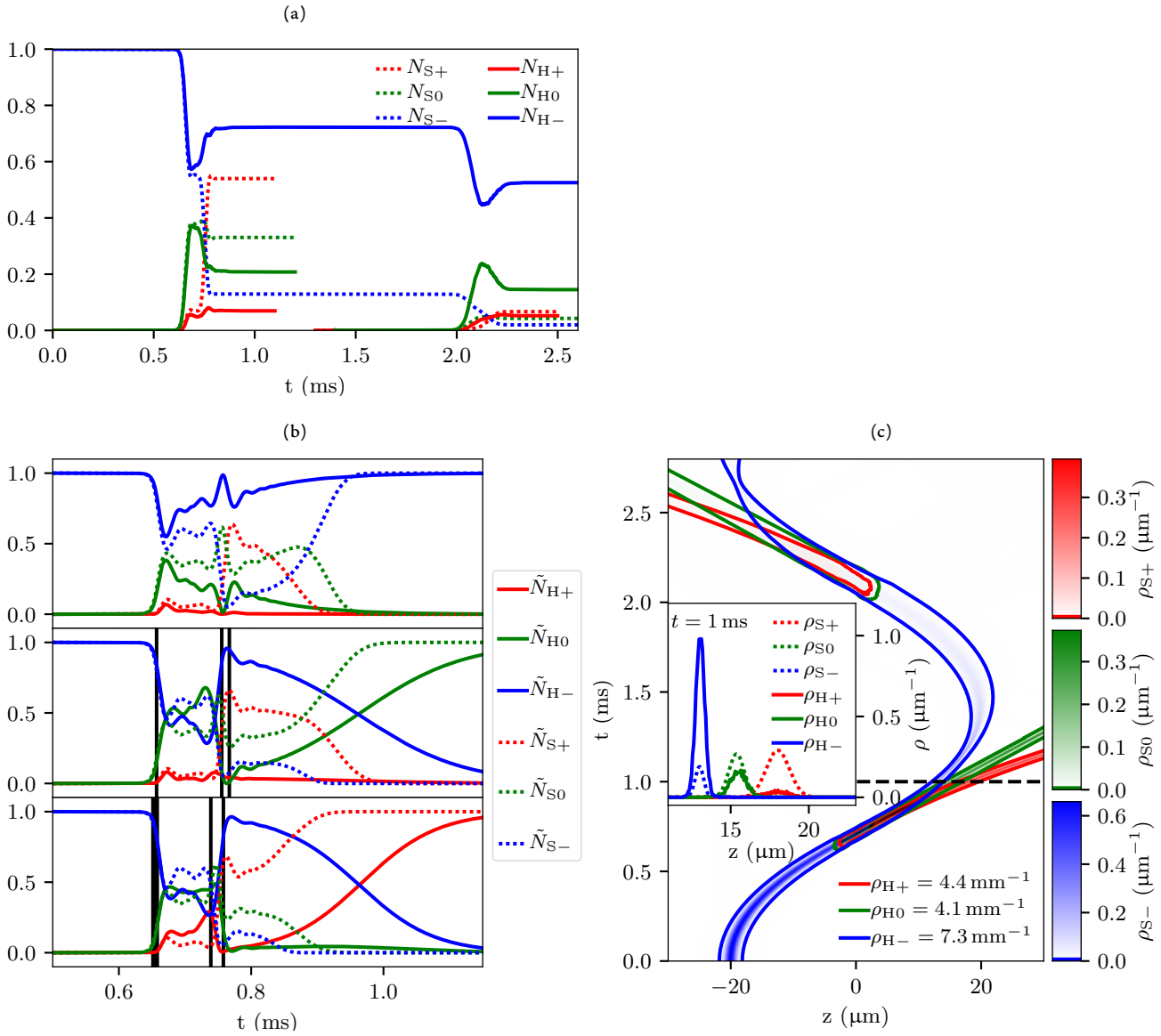

Figure 6.11: Scenario II results for the approximate Markovian decoherence hidden-variable semiclassical method (Section 6.6.1) compared to the Schrödinger wave equation. All subfigures as described in Figure 6.9. There are two minima in the magnetic field, closely spaced near $z = 0$. Here we see the breakdown of the Markovian model in the presence of a second region of non-adiabatic coupling that the atoms encounter before they have fully decohered after the first. (a) The inaccurate modelling of decoherence has resulted in the wrong relative populations and phases between the states when the atom encounters the second region of coupling, leading to completely different populations after it emerges from the second field minimum. (b) Local populations along specific classical trajectories. Incorrect decoherence transients were inconsequential when there were no closely spaced field minima, but here the wrong decoherence results in the wrong initial conditions as the atom enter the region of non-adiabatic coupling near the second field minimum. If the minima were farther apart, the exponential decoherence would decay the system back to a single eigenstate, which would be correct. At shorter times, it matters whether the decoherence curves actually have the right shape. The semiclassical atoms near the centre of the wavepacket have very small population in $m_F = 0$ and $m_F = -1$ states after the second region of coupling, as such there were no simulated atoms whose hidden variable remained in these two states after the first region of coupling. Plotted instead are results from atoms that made additional spin flips, but quickly transitioned back to the desired state. This can be seen as groups of multiple black lines indicating a number of closely spaced spin flips. As mentioned earlier, this is one disadvantage of Schrödinger theory: it often makes multiple transitions in a region of coupling in cases where Tully's fewest-switches algorithm would make only one, making it harder to pick out which atoms made certain transitions of interest. (c) Probability density of trajectories. The trajectories themselves are in excellent agreement with the Schrödinger wave equation—but there are the wrong proportion of atoms in each cluster of trajectories.

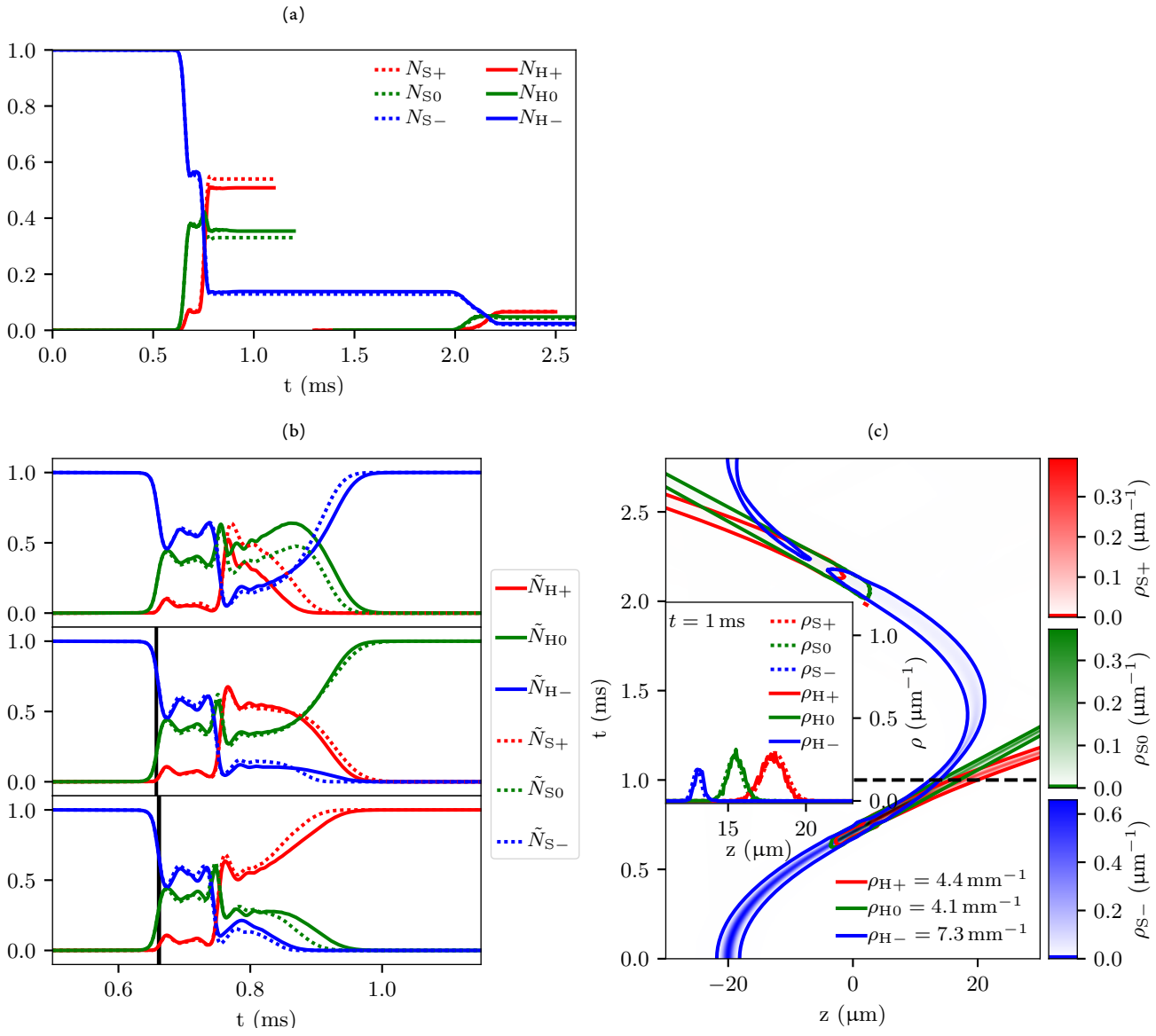

Figure 6.12: Scenario II results for the mean-auxiliary-trajectories hidden-variable semiclassical method (Section 6.6.2) compared to the Schrödinger wave equation. All subfigures as described in Figure 6.9. (a) Here we see that the auxiliary-trajectories method of modelling decoherence leads to correct populations (again, within statistical error), even when there are two field minima closely spaced. (b) Here we see why: the decoherence curves match those of the Schrödinger wave equation in between the two field minima (identifiable by the two large vertical drops in the $m_F = -1$ population), providing the correct initial conditions as the atoms approach the second field minimum. (c) The trajectories and probability density functions agree well.

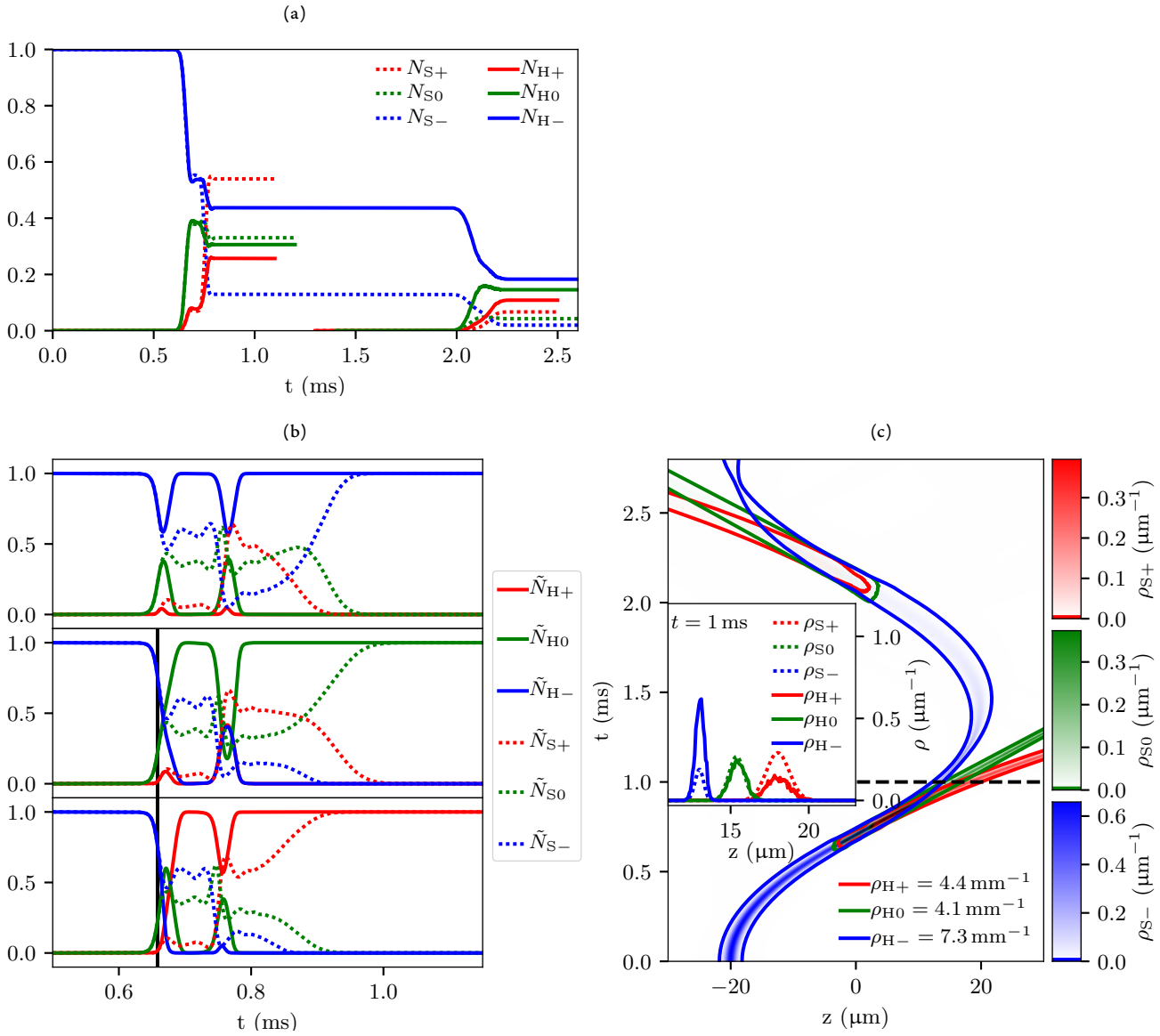

Figure 6.13: Scenario II results with the mean-auxiliary-trajectories hidden-variable semiclassical model (Section 6.6.2), modified to use Gaussian-projection based decoherence rates, compared to the Schrödinger wave equation. All subplots as described in Figure 6.9. (a) Populations after the first region of coupling are correct, but are clearly not correct after the second region of coupling. (b) We see that the cause of this is decoherence that is much too fast decoherence, with population of states other than that selected by the hidden variable decaying away at a rate as fast or faster than that at which it appears, leading to symmetrical profiles in population around the coupling regions. Of course, since this plot is comparing the state vector's local populations to those of the exact wavefunction along specific trajectories, and the decoherence model for this simulation is one based on projecting onto Gaussians rather than Dirac deltas centred exactly on the trajectory, we should not expect them to exactly agree (for that we would need to compare to projections of the exact wavefunctions onto Gaussians). However the disagreement is stark enough to make our point despite this. (c) As with the other results with poor modelling of decoherence, the trajectories are correct but the populations are not.

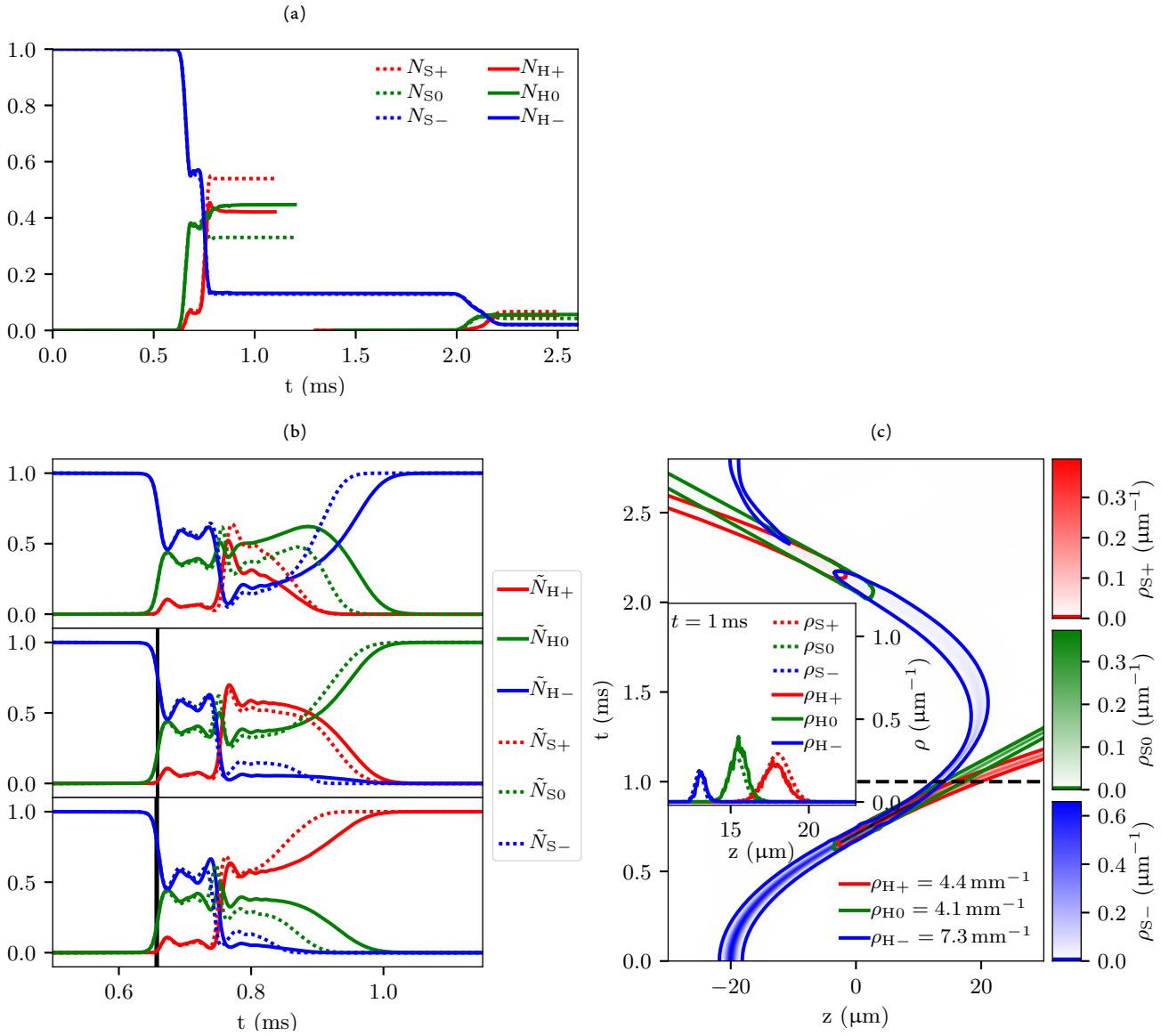

Figure 6.14: Scenario II results with Gaussian-projection based decoherence under the assumption of zero relative wavenumber between the Gaussians being projected onto each other when computing the decoherence rate. This decoherence rate $\gamma_{ij}^{\mathrm{MM}}|_{k_{ij}=0}(t)$ is used with the mean-auxiliary-trajectories hidden-variable semiclassical model (Section 6.6.2), and the results compared to the Schrödinger wave equation. All subplots as described in Figure 6.9. (a) State populations are in much closer agreement with the Schrödinger wave equation results than when we did not set the relative wavenumber of the Gaussians to zero (Figure 6.13). However, they are still not as accurate as our Dirac-delta based decoherence model (Figure 6.12). (b) As with Figure 6.13, we should not expect the local Schrödinger wave equation populations along the trajectories of the semiclassical atoms to agree with the state populations. However, for the three semiclassical atoms plotted (chosen as their trajectories were closest to the centre of the Gaussians and their transitions closest to the centre of the first region of coupling), they appear to agree quite well, including immediately after the second region of coupling. This may be a coincidence, or perhaps population transfers for atoms near the centre of the Gaussian are computed accurately with this decoherence model, but less accurately for atoms on other trajectories. (c). As usual, the trajectories are correct. We can see the minor error in populations in the inset, and it does not appear that the populations are more accurate in the centre as we just speculated. So any region of high accuracy in population transfer must be small enough to be hidden within the bins of the histogram, and is therefore not particularly of interest.

of the atom, are modest improvements over similar methods such as that by Shenvi *et al.* [174]. The decoherence rate that results when Shenvi *et al.* remove velocity separation from their decoherence calculation are still based on the overlap of two Gaussians, and is therefore still a kind of average over the entire wavepacket. The real part of the resulting decoherence rate is different from that presented here, based on Dirac deltas, by a factor of two, and lacks an imaginary part (velocity separation in my model still causes a phase shift). As I've argued, using the value at the centre of the wavepacket is more natural than averaging, as this gives you something closer to a value the wavefunction actually has at some point in space, and this is borne out by the simulation results in the previous section.

Schrödinger theory has not been applied to surface-hopping prior to this work, and I've shown in the results section of this chapter that it gives good results. However, as discussed in Section 6.4.1, it is considerably expensive to compute transition probabilities from, though it can take as input arbitrary unitaries, allowing one to compute transition probabilities consistent with quantum evolution over arbitrary time intervals, rather than integrating infinitesimal transition probabilities over a finite time interval as with Tully's fewest-switches.

The fact that Tully's fewest-switches algorithm gives smaller transition probabilities than Schrödinger theory (even though both are consistent with the quantum probabilities) makes it more desirable for use in a surface-hopping/hidden-variable semiclassical model. This is because every transition the hidden variable of a semiclassical atom makes is an opportunity for the atom to spend some time subject to a different potential. Even if the probability of the atom ending up in the right state is correct, the more potential surfaces it visits on its way there, the larger the variance in the integrated force it experienced will be, and hence the larger the variance in its final trajectory. A larger number of transitions also implies a larger variance in the number of atoms in each state (as given by their hidden variable), even though the expectation value might agree with the quantum populations. A 'fewest-switches' algorithm is therefore likely to be the most accurate in this sense of reduced statistical variation. One final advantage of fewest hops in the context of my mean-auxiliary-trajectories method is that it means the algorithm will not discard as many auxiliary trajectories upon switching to them, resulting in possibly more accurate decoherence.

On that note, one could also imagine a variation of the method where an auxiliary trajectory is simply considered the main trajectory upon a transition, and not replaced by the previous main trajectory. In that case the number of hops would not influence whether the decoherence history was destroyed more than necessary.

Although the core idea of using a stochastically evolving variable to choose which classical force to subject a semiclassical atom to wasn't original, in rediscovering it I have identified that hidden-variable theories and hopping algorithms are in fact the same thing. With the ideas from the surface-hopping literature that my method previously lacked, and with my own improvements, the method has applications to a range of problems in cold atom physics, in simulating evaporative cooling, laser cooling, and other phenomena in cold atom physics whenever thermal atoms are subject to state-dependent forces.

References

[1] A. Einstein. *Quantentheorie des einatomigen idealen Gases*. Sitzungsberichte der Preussischen Akademie der Wissenschaften **3** (1925). [p 1]

[2] S. N. Bose. *Plancks Gesetz und Lichtquantenhypothese*. Zeitschrift für Physik A Hadrons and Nuclei **26**, 178 (1924). DOI: 10.1007/BF01327326. [p 1]

[3] M. H. Anderson, J. R. Ensher, M. R. Matthews, C. E. Wieman, and E. A. Cornell. *Observation of Bose–Einstein Condensation in a Dilute Atomic Vapor*. Science **269**, 198 (1995). DOI: 10.1126/science.269.5221.198. [pp 1 and 7]

[4] K. B. Davis, M. O. Mewes, M. R. Andrews, N. J. van Druten, D. S. Durfee, D. M. Kurn, and W. Ketterle. *Bose–Einstein Condensation in a Gas of Sodium Atoms*. Physical Review Letters **75**, 3969 (1995). DOI: 10.1103/PhysRevLett.75.3969. [p 1]

[5] G. Modugno, G. Ferrari, G. Roati, R. J. Brecha, A. Simoni, and M. Inguscio. *Bose–Einstein Condensation of Potassium Atoms by Sympathetic Cooling*. Science **294**, 1320 (2001). DOI: 10.1126/science.1066687. [p 1]

[6] C. C. Bradley, C. A. Sackett, and R. G. Hulet. *Bose–Einstein Condensation of Lithium: Observation of Limited Condensate Number*. Physical Review Letters **78**, 985 (1997). DOI: 10.1103/PhysRevLett.78.985. [p 1]

[7] T. Weber, J. Herbig, M. Mark, H.-C. Nägerl, and R. Grimm. *Bose–Einstein Condensation of Cesium*. Science **299**, 232 (2003). DOI: 10.1126/science.1079699. [p 1]

[8] N. P. Robins, P. A. Altin, J. E. Debs, and J. D. Close. *Atom lasers: Production, properties and prospects for precision inertial measurement*. Physics Reports **529**, 265 (2013). DOI: 10.1016/j.physrep.2013.03.006. [p 1]

[9] A. D. Cronin, J. Schmiedmayer, and D. E. Pritchard. *Optics and interferometry with atoms and molecules*. Rev. Mod. Phys. **81**, 1051 (2009). DOI: 10.1103/RevModPhys.81.1051. [p 1]

[10] T. D. Ladd, F. Jelezko, R. Laflamme, Y. Nakamura, C. Monroe, and J. L. O'Brien. *Quantum computers*. Nature **464**, 45 (2010). DOI: 10.1038/nature08812. [p 1]

[11] A. Negretti, P. Treutlein, and T. Calarco. *Quantum computing implementations with neutral particles*. Quantum Information Processing **10**, 721 (2011). DOI: 10.1007/s11128-011-0291-5. [p 1]

[12] I. Bloch, J. Dalibard, and S. Nascimbène. *Quantum simulations with ultracold quantum gases*. Nature Physics **8**, 267 (2012). DOI: 10.1038/nphys2259. [p 1]

[13] R. Blatt and C. F. Roos. *Quantum simulations with trapped ions*. Nature Physics **8**, 277 (2012). DOI: 10.1038/nphys2252. [p 1]

[14] A. A. Norrie, R. J. Ballagh, and C. W. Gardiner. *Quantum turbulence and correlations in Bose-Einstein condensate collisions*. Physical Review A **73**, 043617 (2006). DOI: 10.1103/PhysRevA.73.043617. [p 1]

[15] P. T. Starkey, C. J. Billington, S. P. Johnstone, M. Jasperse, K. Helmerson, L. D. Turner, and R. P. Anderson. *A scripted control system for autonomous hardware-timed experiments*. Review of Scientific Instruments **84**, 085111 (2013). DOI: 10.1063/1.4817213. [pp 2, 3, 88, and 99]

[16] W. F. Vinen. *The Detection of Single Quanta of Circulation in Liquid Helium II*. Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences **260**, 218 (1961). DOI: 10.1098/rspa.1961.0029. [p 2]

[17] M. R. Matthews, B. P. Anderson, P. C. Haljan, D. S. Hall, C. E. Wieman, and E. A. Cornell. *Vortices in a Bose–Einstein Condensate*. Physical Review Letters **83**, 2498 (1999). DOI: 10.1103/PhysRevLett.83.2498. [p 2]

[18] C. F. Barenghi, R. J. Donnelly, and W. F. Vinen. *Quantized Vortex Dynamics and Superfluid Turbulence*. Springer (2001). [p 2]

[19] W. Gerlach and O. Stern. *Der experimentelle Nachweis der Richtungsquantelung im Magnetfeld*. Zeitschrift für Physik **9**, 349 (1922). DOI: 10.1007/BF01326983. [pp 3, 145, and 148]

[20] J. C. Tully. *Molecular dynamics with electronic transitions*. The Journal of Chemical Physics **93**, 1061 (1990). DOI: 10.1063/1.459170. [pp 3, 146, 147, 154, 158, 165, and 171]

[21] S. Aaronson. *Quantum computing and hidden variables*. Phys. Rev. A **71**, 032325 (2005). DOI: 10.1103/PhysRevA.71.032325. [pp 3, 145, 151, 152, 153, and 158]

[22] S. F. Owen and D. S. Hall. *Fast line-based experiment timing system for LabVIEW*. Review of Scientific Instruments **75**, 259 (2003). DOI: 10.1063/1.1630833. [pp 3 and 105]

[23] D. J. Wineland, R. E. Drullinger, and F. L. Walls. *Radiation-Pressure Cooling of Bound Resonant Absorbers*. Physical Review Letters **40**, 1639 (1978). DOI: 10.1103/PhysRevLett.40.1639. [p 5]

[24] H. J. Metcalf and P. V. der Straten. *Laser Cooling and Trapping*. Springer (1999). [pp 6, 10, 19, 137, and 139]

[25] E. L. Raab, M. Prentiss, A. Cable, S. Chu, and D. E. Pritchard. *Trapping of Neutral Sodium Atoms with Radiation Pressure*. Physical Review Letters **59**, 2631 (1987). DOI: 10.1103/PhysRevLett.59.2631. [p 6]

[26] P. Zeeman. *On the influence of magnetism on the nature of the light emitted by a substance*. Philosophical Magazine Series 5 **43**, 226 (1897). [pp 6 and 14]

[27] D. M. Brink and C. V. Sukumar. *Majorana spin-flip transitions in a magnetic trap*. Physical Review A **74**, 035401 (2006). DOI: 10.1103/PhysRevA.74.035401. [pp 6 and 7]

[28] A. Ashkin. *Acceleration and Trapping of Particles by Radiation Pressure*. Physical Review Letters **24**, 156 (1970). DOI: 10.1103/PhysRevLett.24.156. [p 7]

[29] J. Dalibard and C. Cohen-Tannoudji. *Laser cooling below the Doppler limit by polarization gradients: Simple theoretical models*. JOSA B **6**, 2023 (1989). DOI: 10.1364/JOSAB.6.002023. [pp 7 and 28]

[30] P. J. Ungar, D. S. Weiss, E. Riis, and S. Chu. *Optical molasses and multilevel atoms: Theory*. Journal of the Optical Society of America B **6**, 2058 (1989). DOI: 10.1364/JOSAB.6.002058. [p 7]

[31] P. D. Lett, W. D. Phillips, S. L. Rolston, C. E. Tanner, R. N. Watts, and C. I. Westbrook. *Optical molasses*. Journal of the Optical Society of America B **6**, 2084 (1989). DOI: 10.1364/JOSAB.6.002084. [p 7]

[32] C. Salomon, J. Dalibard, W. D. Phillips, A. Clairon, and S. Guellati. *Laser Cooling of Cesium Atoms Below 3 μK*. Europhysics Letters (EPL) **12**, 683 (1990). DOI: 10.1209/0295-5075/12/8/003. [p 7]

[33] H. F. Hess. *Evaporative cooling of magnetically trapped and compressed spin-polarized hydrogen*. Physical Review B **34**, 3476 (1986). DOI: 10.1103/PhysRevB.34.3476. [p 7]

[34] Y.-J. Lin, A. R. Perry, R. L. Compton, I. B. Spielman, and J. V. Porto. *Rapid production of ^{87}Rb Bose–Einstein condensates in a combined magnetic and optical potential*. Physical Review A **79**, 063631 (2009). DOI: 10.1103/PhysRevA.79.063631. [p 7]

[35] C. Chin, R. Grimm, P. Julienne, and E. Tiesinga. *Feshbach resonances in ultracold gases*. Reviews of Modern Physics **82**, 1225 (2010). DOI: 10.1103/RevModPhys.82.1225. [p 8]

[36] E. Tiesinga, B. J. Verhaar, and H. T. C. Stoof. *Threshold and resonance phenomena in ultracold ground-state collisions*. Physical Review A **47**, 4114 (1993). DOI: 10.1103/PhysRevA.47.4114. [p 8]

[37] S. Inouye, M. R. Andrews, J. Stenger, H.-J. Miesner, D. M. Stamper-Kurn, and W. Ketterle. *Observation of Feshbach resonances in a Bose–Einstein condensate*. Nature **392**, 151 (1998). DOI: 10.1038/32354. [p 8]

[38] G. Thalhammer, G. Barontini, L. De Sarlo, J. Catani, F. Minardi, and M. Inguscio. *Double Species Bose–Einstein Condensate with Tunable Interspecies Interactions*. Physical Review Letters **100**, 210402 (2008). DOI: 10.1103/PhysRevLett.100.210402. [pp 9, 127, and 137]

[39] E. Madelung. *Quantentheorie in hydrodynamischer Form*. Zeitschrift für Physik A Hadrons and Nuclei **40**, 322 (1927). DOI: 10.1007/BF01400372. [p 9]

[40] D. A. Steck. *Rubidium 87 D Line Data*, (2015). (revision 2.1.4) http://steck.us/alkalidata. [pp 10, 11, 14, 15, 18, 23, 24, and 141]

[41] D. A. Steck. *Quantum and Atom Optics*, (2017). (revision 0.12.0) http://steck.us/teachinng. [pp 10, 19, 20, 22, and 24]

[42] B. E. King. *Angular Momentum Coupling and Rabi Frequencies for Simple Atomic Transitions*. arXiv:0804.4528 [physics] (2008). arXiv: 0804.4528. [pp 10 and 20]

[43] P. M. Farrell and W. R. MacGillivray. *On the consistency of Rabi frequency calculations*. Journal of Physics A: Mathematical and General **28**, 209 (1995). DOI: 10.1088/0305-4470/28/1/023. [p 10]

[44] E. Arimondo, M. Inguscio, and P. Violino. *Experimental determinations of the hyperfine structure in the alkali atoms*. Rev. Mod. Phys. **49**, 31 (1977). DOI: 10.1103/RevModPhys.49.31. [p 11]

[45] J. J. Sakurai. *Modern Quantum Mechanics*. Pearson (1994). [p 12]

[46] G. Breit and I. I. Rabi. *Measurement of Nuclear Spin*. Physical Review **38**, 2082 (1931). DOI: 10.1103/PhysRev.38.2082.2. [p 19]

[47] L. M. Bennie, P. B. Wigley, S. S. Szigeti, M. Jasperse, J. J. Hope, L. D. Turner, and R. P. Anderson. *Precise wavefunction engineering with magnetic resonance*. arXiv:1412.6854 (2014). ARXIV: 1412.6854. [p 27]

[48] A. Görlitz, J. M. Vogels, A. E. Leanhardt, C. Raman, T. L. Gustavson, J. R. Abo-Shaeer, A. P. Chikkatur, S. Gupta, S. Inouye, T. Rosenband, and W. Ketterle. *Realization of Bose-Einstein Condensates in Lower Dimensions*. Physical Review Letters **87**, 130402 (2001). DOI: 10.1103/PhysRevLett.87.130402. [p 28]

[49] S. Hofferberth, I. Lesanovsky, B. Fischer, T. Schumm, and J. Schmiedmayer. *Non-equilibrium coherence dynamics in one-dimensional Bose gases*. Nature **449**, 324 (2007). DOI: 10.1038/nature06149. [p 28]

[50] B. Rauer, P. Grišins, I. E. Mazets, T. Schweigler, W. Rohringer, R. Geiger, T. Langen, and J. Schmiedmayer. *Cooling of a one-dimensional Bose gas*. Physical Review Letters **116**, 030402 (2016). DOI: 10.1103/PhysRevLett.116.030402. [p 28]

[51] P. D. Lett, R. N. Watts, C. I. Westbrook, W. D. Phillips, P. L. Gould, and H. J. Metcalf. *Observation of Atoms Laser Cooled below the Doppler Limit*. Physical Review Letters **61**, 169 (1988). DOI: 10.1103/PhysRevLett.61.169. [p 28]

[52] M. Saffman, T. G. Walker, and K. Mølmer. *Quantum information with Rydberg atoms*. Reviews of Modern Physics **82**, 2313 (2010). DOI: 10.1103/RevModPhys.82.2313. [p 28]

[53] E. Urban, T. A. Johnson, T. Henage, L. Isenhower, D. D. Yavuz, T. G. Walker, and M. Saffman. *Observation of Rydberg blockade between two atoms*. Nature Physics **5**, 110 (2009). DOI: 10.1038/nphys1178. [p 28]

[54] B. B. Blinov, J. R. N. Kohn, M. J. Madsen, P. Maunz, D. L. Moehring, and C. Monroe. *Broadband laser cooling of trapped atoms with ultrafast pulses*. JOSA B **23**, 1170 (2006). DOI: 10.1364/JOSAB.23.001170. [p 28]

[55] A. J. McCulloch, D. V. Sheludko, M. Junker, and R. E. Scholten. *High-coherence picosecond electron bunches from cold atoms*. Nature Communications **4**, 1692 (2013). DOI: 10.1038/ncomms2699. [p 28]

[56] T. Brabec and F. Krausz. *Intense few-cycle laser fields: Frontiers of nonlinear optics*. Reviews of Modern Physics **72**, 545 (2000). DOI: 10.1103/RevModPhys.72.545. [p 28]

[57] C. Brezinski. *Extrapolation algorithms and Padé approximations: A historical survey*. Applied Numerical Mathematics **20**, 299 (1996). DOI: 10.1016/0168-9274(95)00110-7. [p 30]

[58] C. Moler and C. Van Loan. *Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later*. SIAM Review **45**, 3 (2003). DOI: 10.1137/S00361445024180. [pp 30 and 31]

[59] J. J. Sakurai. *Modern quantum mechanics*. Addison-Wesley Pub. Co, Reading, Mass (1994). [pp 31 and 77]

[60] D. J. Tannor. *Introduction to Quantum Mechanics: A Time-Dependent Perspective*. University Science Books (2007). [pp 33, 36, 55, 62, and 85]

[61] S. Weinzierl. *Introduction to Monte Carlo methods*. arXiv:hep-ph/0006269 (2000). ARXIV: hep-ph/0006269. [p 34]

[62] F. J. Dyson. *The S Matrix in Quantum Electrodynamics*. Physical Review **75**, 1736 (1949). DOI: 10.1103/PhysRev.75.1736. [p 34]

[63] V. Kaplunovsky. *On Perturbation Theory, Dyson Series, and Feynman Diagrams*, (2016). Online lecture notes http://bolvan.ph.utexas.edu/~vadim/Classes/2016f/dyson.pdf. [p 34]

[64] M. Suzuki. *Quantum Monte Carlo Methods in Condensed Matter Physics*. World Scientific (1993). [p 38]

[65] B. I. Schneider and L. A. Collins. *The discrete variable method for the solution of the time-dependent Schrödinger equation*. Journal of Non-Crystalline Solids **351**, 1551 (2005). DOI: 10.1016/j.jnoncrysol.2005.03.028. [pp 38, 62, 63, 65, and 67]

[66] M. Suzuki. *General Decomposition Theory of Ordered Exponentials*. Proceedings of the Japan Academy, Series B **69**, 161 (1993). DOI: 10.2183/pjab.69.161. [p 38]

[67] B. I. Schneider, L. A. Collins, and S. X. Hu. *Parallel solver for the time-dependent linear and nonlinear Schrödinger equation*. Physical Review E **73**, 036708 (2006). DOI: 10.1103/PhysRevE.73.036708. [pp 42, 62, 65, 72, 81, and 84]

[68] J. Javanainen and J. Ruostekoski. *Symbolic calculation in development of algorithms: Split-step methods for the Gross–Pitaevskii equation*. Journal of Physics A: Mathematical and General **39**, L179 (2006). DOI: 10.1088/0305-4470/39/12/L02. [p 44]

[69] R. D. Skeel, G. Zhang, and T. Schlick. *A family of symplectic integrators: stability, accuracy, and molecular dynamics applications*. SIAM J. Sci. Comput **18**, 203 (1997). [p 45]

[70] G. Dahlquist and Å. Björck. *Numerical Methods*. Dover Books on Mathematics. Dover Publications (2003). LCCB: 2002072867. [p 45]

[71] K. Mølmer and Y. Castin. *Monte carlo wavefunctions in quantum optics*. Quantum and Semiclassical Optics: Journal of the European Optical Society Part B **8**, 49 (1996). [p 45]

[72] M. B. Plenio and P. L. Knight. *The quantum-jump approach to dissipative dynamics in quantum optics*. Rev. Mod. Phys. **70**, 101 (1998). DOI: 10.1103/RevModPhys.70.101. [p 45]

[73] A. Bruckner, J. Bruckner, and B. Thomson. *Real Analysis*. Prentice-Hall (1997). LCCB: 96022123. [p 53]

[74] H. Levi. *A geometric construction of the Dirichlet kernel applications*. Transactions of the New York Academy of Sciences **36**, 640 (1974). DOI: 10.1111/j.2164-0947.1974.tb03023.x. [p 53]

[75] S. A. Orszag. *Comparison of pseudospectral and spectral approximation*. Studies in Applied Mathematics **51**, 253 (1972). DOI: 10.1002/sapm1972513253. [p 55]

[76] N. Phillips. *An example of non-linear computational instability*. In *The Atmosphere and the Sea in Motion*, pages 501–504. Rockefeller Univ. Press (1959). [p 56]

[77] B. Fornberg. *The pseudospectral method: Comparisons with finite differences for the elastic wave equation*. GEOPHYSICS **52**, 483 (1987). DOI: 10.1190/1.1442319. [pp 58, 59, and 68]

[78] B. Fornberg. *Generation of finite difference formulas on arbitrarily spaced grids*. Mathematics of Computation **51**, 699 (1988). DOI: 10.1090/S0025-5718-1988-0935077-0. [pp 59, 61, and 72]

[79] W. Press. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press (2007). LCCB: 89015841. [p 60]

[80] A. Gupta and V. Kumar. *The scalability of FFT on parallel computers*. IEEE Transactions on Parallel and Distributed Systems **4**, 922 (1993). [pp 60 and 81]

[81] J.-P. Berrut and L. N. Trefethen. *Barycentric lagrange interpolation*. SIAM Review **46**, 501 (2004). DOI: 10.1137/S0036144502417715. [p 63]

[82] R. M. Caplan and R. Carretero-González. *Numerical stability of explicit runge-kutta finite difference schemes for the nonlinear schrödinger equation*. CoRR **abs/1107.4810** (2011). [p 66]

[83] B. M. C. Davies. *Vortex dynamics in Bose–Einstein condensates*. PhD thesis, University of Otago, Dunedin, New Zealand (2000). [pp 66 and 78]

[84] L. Lehtovaara, J. Toivanen, and J. Eloranta. *Solution of time-independent Schrödinger equation by the imaginary time propagation method*. J. Comput. Phys. **221**, 148 (2007). DOI: http://dx.doi.org/10.1016/j.jcp.2006.06.006. [p 72]

[85] D. M. Young. *Iterative Methods for Solving Partial Difference Equations of Elliptical Type*. PhD thesis, Harvard University (1950). [p 74]

[86] D. Ahn and S. L. Chuang. *Variational calculations of subbands in a quantum well with uniform electric field: Gram–Schmidt orthogonalization approach*. Applied Physics Letters **49**, 1450 (1986). DOI: 10.1063/1.97299. [p 76]

[87] W. Press. *The art of scientific computing*. Cambridge University Press (1992). [p 78]

[88] C. Pethick and H. Smith. *Bose–Einstein condensation in dilute gases*. Cambridge University Press (2002). [p 79]

[89] M. L. Chiofalo, S. Succi, and M. P. Tosi. *Ground state of trapped interacting Bose–Einstein condensates by an explicit imaginary-time algorithm*. Phys. Rev. E **62**, 7438 (2000). DOI: 10.1103/PhysRevE.62.7438. [pp 80 and 83]

[90] G. M. Muslu and H. A. Erbay. *Higher-order split-step Fourier schemes for the generalized nonlinear Schrödinger equation*. Mathematics and Computers in Simulation **7**, 581 (2005). DOI: 10.1016/j.matcom.2004.08.002. [p 80]

[91] M. Heroux, P. Raghavan, and H. Simon. *Parallel processing for scientific computing*. Society for Industrial and Applied Mathematics, Philadelphia, PA (2006). [p 81]

[92] P. Gulshani and D. J. Rowe. *Quantum mechanics in rotating frames. I. The impossibility of rigid flow*. Canadian Journal of Physics **56**, 468 (1978). DOI: 10.1139/p78-060. [p 83]

[93] E. G. M. van Kempen, S. J. J. M. F. Kokkelmans, D. J. Heinzen, and B. J. Verhaar. *Interisotope determination of ultracold rubidium interactions from three high-precision experiments*. Phys. Rev. Lett. **88**, 093201 (2002). DOI: 10.1103/PhysRevLett.88.093201. [p 83]

[94] Python Software Foundation. *Python language reference, version 3.7*, (2018). http://www.python.org. [p 87]

[95] National Instruments. *Laboratory Virtual Instrument Engineering Workbench (LabVIEW)*, (2018). http://www.ni.com/labview. [p 87]

[96] N. P. Robins, P. A. Altin, J. E. Debs, and J. D. Close. *Atom lasers: Production, properties and prospects for precision inertial measurement*. Physics Reports **529**, 265 (2013). DOI: 10.1016/j.physrep.2013.03.006. [p 87]

[97] A. D. Cronin, J. Schmiedmayer, and D. E. Pritchard. *Optics and interferometry with atoms and molecules*. Reviews of Modern Physics **81**, 1051 (2009). DOI: 10.1103/RevModPhys.81.1051. [p 87]

[98] A. Negretti, P. Treutlein, and T. Calarco. *Quantum computing implementations with neutral particles*. Quantum Information Processing **10**, 721 (2011). DOI: 10.1007/s11128-011-0291-5. [p 87]

[99] T. D. Ladd, F. Jelezko, R. Laflamme, Y. Nakamura, C. Monroe, and J. L. O'Brien. *Quantum computers*. Nature **464**, 45 (2010). DOI: 10.1038/nature08812. [p 87]

[100] I. Bloch, J. Dalibard, and S. Nascimbène. *Quantum simulations with ultracold quantum gases*. Nature Physics **8**, 267 (2012). DOI: 10.1038/nphys2259. [p 87]

[101] R. Blatt and C. F. Roos. *Quantum simulations with trapped ions*. Nature Physics **8**, 277 (2012). DOI: 10.1038/nphys2252. [p 87]

[102] P. Starkey. *A software framework for control and automation of precisely timed experiments*. PhD thesis, Monash University (2018). (in preparation). [pp 88 and 96]

[103] The HDF Group. *Hierarchical Data Format, version 5*, (1997–2018). http://www.hdfgroup.org/HDF5/. [p 89]

[104] C. Klempt, T. van Zoest, T. Henninger, O. Topic, E. Rasel, W. Ertmer, and J. Arlt. *Ultraviolet light-induced atom desorption for large rubidium and potassium magneto-optical traps*. Physical Review A **73**, 013410 (2006). DOI: 10.1103/PhysRevA.73.013410. [p 92]

[105] J. D. Hunter. *Matplotlib: A 2D graphics environment*. Computing In Science & Engineering **9**, 90 (2007). DOI: 10.1109/MCSE.2007.55. [p 92]

[106] L. Campagnola. *Pyqtgraph, version 0.10.0*, (2016). http://www.pyqtgraph.org. [pp 92 and 105]

[107] W. McKinney. *Data Structures for Statistical Computing in Python*. In S. van der Walt and J. Millman (editors), *Proceedings of the 9th Python in Science Conference*, pages 51–56 (2010). [pp 92 and 98]

[108] F. Perez and B. E. Granger. *IPython: A System for Interactive Scientific Computing*. Computing in Science Engineering **9**, 21 (2007). DOI: 10.1109/MCSE.2007.53. [p 92]

[109] S. P. Johnstone, A. J. Groszek, P. T. Starkey, C. J. Billington, T. P. Simula, and K. Helmerson. *Order from chaos: Observation of large-scale flow from turbulence in a two-dimensional superfluid*. arXiv:1801.06952 [cond-mat, physics:physics] (2018). ARXIV: 1801.06952. [pp 94 and 121]

[110] P. Gill and P. Baird. *An Optical Lattice Clock with Neutral Strontium*. PhD thesis, University of Oxford (2016). [p 96]

[111] M. Gancarz. *The UNIX Philosophy*. Digital Press, Newton, MA, USA (1995). [p 96]

[112] P. B. Wigley, P. J. Everitt, A. van den Hengel, J. W. Bastian, M. A. Sooriyabandara, G. D. McDonald, K. S. Hardman, C. D. Quinlivan, P. Manju, C. C. N. Kuhn, I. R. Petersen, A. N. Luiten, J. J. Hope, N. P. Robins, and M. R. Hush. *Fast machine-learning online optimization of ultra-cold-atom experiments*. Scientific Reports **6**, 25890 (2016). DOI: 10.1038/srep25890. [p 97]

[113] R. Speirs, (2018). Private Communication. [p 97]

[114] T. E. Oliphant. *Guide to NumPy*. CreateSpace Independent Publishing Platform, USA, 2nd edition (2015). [p 98]

[115] Brian E. Granger and contributors. *Pyzmq, version 17.1.0*, (2018). http://pyzmq.readthedocs.io. [p 98]

[116] Andrew Collette and contributors. *H5py, version 2.8.0*, (2018). http://www.h5py.org. [p 98]

[117] P. Hintjens. *ZeroMQ: The Guide*, (2010). http://zguide.zeromq.org. [p 98]

[118] Zachtronics Industries. *SpaceChem*, (2011). http://www.zachtronics.com/spacechem. [p 98]

[119] C. J. Billington. *Zprocess, version 2.4.12*, (2018). http://bitbucket.org/cbillington/zprocess. [pp 99 and 105]

[120] Monash University School of Physics and Astronomy. *Measurement of β-ray spectra*, (2016). Undergraduate laboratory materials. [p 99]

[121] The Qt Company. *Qt, version 5.11*, (2018). http://www.qt.io. [p 99]

[122] P. T. Starkey and C. J. Billington. *Qtutils, version 2.1.0*, (2018). http://bitbucket.org/philipstarkey/qtutils. [pp 99, 105, and 107]

[123] The GNOME Project. *GTK+, version 3*, (2018). http://gtk.org. [p 99]

[124] Riverbank Computing. *PyQt, version 5.10.1*, (2018). http://gtk.org. [p 99]

[125] The GNOME Project. *GNOME 3.28.2*, (2018). http://gnome.org. [p 99]

[126] Continuum Analytics. *Anaconda Distribution, version 5.1*, (2018). http://anaconda.org. [p 100]

[127] Torsten Bronger, Gregor Thalhammer, Florian Bauer, and Hernan E. Grecco. *Pyvisa, version 1.9.0*, (2018). http://pyvisa.readthedocs.io. [p 103]

[128] Peter Johnson, FRC Team 294. *Pynivision, version 2015.0.0*, (2015). http://pyvisa.readthedocs.io. [p 103]

[129] Monash Univeristy. *The labscript suite*, (2018). http://bitbucket.org/philipstarkey/qtutils. [p 105]

[130] Tom Preston-Werner. *Semantic Versioning, version 2.0.0*, (2018). http://semver.org. [p 107]

[131] G. P. Bewley. *The generation of particles to observe quantized vortex dynamics in superfluid helium*. Cryogenics **49**, 549 (2009). DOI: 10.1016/j.cryogenics.2008.10.018. [p 119]

[132] G. P. Bewley, D. P. Lathrop, and K. R. Sreenivasan. *Superfluid Helium: Visualization of quantized vortices*. Nature **441**, 588 (2006). DOI: 10.1038/441588a. [p 119]

[133] R. E. Packard. *Vortex photography in liquid helium*. Physica B+C **109–110**, 1474 (1982). DOI: 10.1016/0378-4363(82)90171-1. [p 119]

[134] W. S. Bakr, J. I. Gillen, A. Peng, S. F|[ouml]|lling, and M. Greiner. *A quantum gas microscope for detecting single atoms in a Hubbard-regime optical lattice*. Nature **462**, 74 (2009). DOI: 10.1038/nature08482. [p 119]

[135] B. P. Anderson, P. C. Haljan, C. A. Regal, D. L. Feder, L. A. Collins, C. W. Clark, and E. A. Cornell. *Watching Dark Solitons Decay into Vortex Rings in a Bose-Einstein Condensate*. Physical Review Letters **86**, 2926 (2001). DOI: 10.1103/PhysRevLett.86.2926. [p 119]

[136] V. Bretin, P. Rosenbusch, F. Chevy, G. V. Shlyapnikov, and J. Dalibard. *Quadrupole Oscillation of a Single-Vortex Bose-Einstein Condensate: Evidence for Kelvin Modes*. Physical Review Letters **90**, 100403 (2003). DOI: 10.1103/PhysRevLett.90.100403. [p 119]

[137] M. Leadbeater, T. Winiecki, D. C. Samuels, C. F. Barenghi, and C. S. Adams. *Sound Emission due to Superfluid Vortex Reconnections*. Physical Review Letters **86**, 1410 (2001). DOI: 10.1103/PhysRevLett.86.1410. [p 119]

[138] C. N. Weiler, T. W. Neely, D. R. Scherer, A. S. Bradley, M. J. Davis, and B. P. Anderson. *Spontaneous vortices in the formation of Bose-Einstein condensates*. Nature **455**, 948 (2008). DOI: 10.1038/nature07334. [p 120]

[139] D. V. Freilich, D. M. Bianchi, A. M. Kaufman, T. K. Langin, and D. S. Hall. *Real-Time Dynamics of Single Vortex Lines and Vortex Dipoles in a Bose-Einstein Condensate*. Science **329**, 1182 (2010). DOI: 10.1126/science.1191224. [p 120]

[140] G. Ahlers. *Trend: Turbulent convection*. Physics **2** (2009). [p 120]

[141] C. Foias, O. Manley, R. Rosa, and R. Temam. *Navier-Stokes Equations and Turbulence*. Cambridge University Press (2001). [p 120]

[142] S. W. Hawking. *A Brief History of Time: From the Big Bang to Black Holes*. Bantam Books (1988). [p 120]

[143] S. Corrsin. *Turbulent Flow*. American Scientist **49**, 300 (1961). [p 121]

[144] P. A. Davidson. *Turbulence: An Introduction for Scientists and Engineers*. Oxford University Press (2004). [p 121]

[145] A. Kolmogorov. *The Local Structure of Turbulence in Incompressible Viscous Fluid for Very Large Reynolds' Numbers*. Akademiia Nauk SSSR Doklady **30**, 301 (1941). [p 121]

[146] D. B. Spalding. *Kolmogorov's Two-Equation Model of Turbulence*. Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences **434**, 211 (1991). DOI: 10.1098/rspa.1991.0089. [p 121]

[147] L. F. Richardson. *Weather Prediction by Numerical Process*. Cambridge University Press (2007). (First published 1922). [p 121]

[148] M. Tsubota and M. Kobayashi. *Energy Spectra of Quantum Turbulence*. In *Progress in Low Temperature Physics: Quantum Turbulence*, volume 16, pages 1–43. Elsevier (2009). [p 121]

[149] W. F. Vinen. *How is turbulent energy dissipated in a superfluid?* Journal of Physics: Condensed Matter **17**, S3231 (2005). DOI: 10.1088/0953-8984/17/45/006. [p 121]

[150] L. Onsager. *Statistical hydrodynamics*. Il Nuovo Cimento (1943-1954) **6**, 279 (1949). DOI: 10.1007/BF02780991. [p 121]

[151] R. H. Kraichnan. *Inertial Ranges in Two-Dimensional Turbulence*. Physics of Fluids **10**, 1417 (1967). DOI: doi:10.1063/1.1762301. [p 121]

[152] T. Simula, M. J. Davis, and K. Helmerson. *Emergence of Order from Turbulence in an Isolated Planar Superfluid*. Physical Review Letters **113**, 165302 (2014). DOI: 10.1103/PhysRevLett.113.165302. [p 121]

[153] A. J. Groszek, M. J. Davis, D. M. Paganin, K. Helmerson, and T. P. Simula. *Vortex Thermometry for Turbulent Two-Dimensional Fluids*. Physical Review Letters **120**, 034504 (2018). DOI: 10.1103/PhysRevLett.120.034504. [p 121]

[154] G. Gauthier, M. T. Reeves, X. Yu, A. S. Bradley, M. Baker, T. A. Bell, H. Rubinsztein-Dunlop, M. J. Davis, and T. W. Neely. *Negative-Temperature Onsager Vortex Clusters in a Quantum Fluid*. arXiv:1801.06951 [cond-mat, physics:physics] (2018). arXiv: 1801.06951. [p 121]

[155] A. J. Leggett. *Superfluidity*. Reviews of Modern Physics **71**, S318 (1999). DOI: 10.1103/RevModPhys.71.S318. [p 122]

[156] A. J. Leggett. *Quantum Liquids: Bose Condensation and Cooper Pairing in Condensed-Matter Systems*. Oxford University Press (2006). [p 123]

[157] C. J. Billington. *Particle Velocimetry of Vortices in Bose-Einstein Condensates*. Honours thesis, Monash University (2010). [pp 123, 124, 125, and 126]

[158] R. Côté, A. Dalgarno, H. Wang, and W. C. Stwalley. *Potassium scattering lengths and prospects for Bose-Einstein condensation and sympathetic cooling*. Physical Review A **57**, R4118 (1998). DOI: 10.1103/PhysRevA.57.R4118. [p 127]

[159] M. Tsubota, K. Kasamatsu, and M. Ueda. *Vortex lattice formation in a rotating Bose-Einstein condensate*. Physical Review A **65**, 023603 (2002). DOI: 10.1103/PhysRevA.65.023603. [p 127]

[160] E. J. M. Madarassy and C. F. Barenghi. *Vortex Dynamics in Trapped Bose-Einstein Condensate*. Journal of Low Temperature Physics **152**, 122 (2008). DOI: 10.1007/s10909-008-9811-9. [p 127]

[161] B. H. Bransden, C. J. Joachain, and T. J. Plivier. *Physics of Atoms and Molecules*. Prentice Hall (2003). [p 128]

[162] J. Stenger, S. Inouye, M. R. Andrews, H.-J. Miesner, D. M. Stamper-Kurn, and W. Ketterle. *Strongly Enhanced Inelastic Collisions in a Bose-Einstein Condensate near Feshbach Resonances*. Physical Review Letters **82**, 2422 (1999). DOI: 10.1103/PhysRevLett.82.2422. [p 144]

[163] J. J. McClelland. *Atom-optical properties of a standing-wave light field*. JOSA B **12**, 1761 (1995). DOI: 10.1364/JOSAB.12.001761. [pp 145 and 147]

[164] H. Wallis. *Quantum theory of atomic motion in laser light*. Physics Reports **255**, 203 (1995). DOI: 10.1016/0370-1573(94)00090-P. [pp 145 and 147]

[165] C. S. Adams and E. Riis. *Laser cooling and trapping of neutral atoms*. Progress in Quantum Electronics **21**, 1 (1997). DOI: 10.1016/S0079-6727(96)00006-7. [pp 145 and 147]

[166] S. Stenholm. *The semiclassical theory of laser cooling*. Reviews of Modern Physics **58**, 699 (1986). DOI: 10.1103/RevModPhys.58.699. [pp 145 and 147]

[167] V. G. Minogin and V. S. Letokhov. *Laser Light Pressure on Atoms*. CRC Press (1987). [pp 145 and 147]

[168] M. Genovese. *Research on hidden variable theories: A review of recent progresses*. Physics Reports **413**, 319 (2005). DOI: 10.1016/j.physrep.2005.03.003. [p 145]

[169] J. S. Bell. *On the Einstein Podolsky Rosen paradox*. Physics Physique Fizika **1**, 195 (1964). DOI: 10.1103/PhysicsPhysiqueFizika.1.195. [p 145]

[170] M. Schlosshauer, J. Kofler, and A. Zeilinger. *A Snapshot of Foundational Attitudes Toward Quantum Mechanics*. Studies in History and Philosophy of Science Part B: Studies in History and Philosophy of Modern Physics **44**, 222 (2013). ARXIV: 1301.1069, DOI: 10.1016/j.shpsb.2013.04.004. [p 145]

[171] J. C. Tully. *Mixed quantum-classical dynamics*. Faraday Discuss. **110**, 407 (1998). DOI: 10.1039/A801824C. [p 146]

[172] J. E. Subotnik, A. Jain, B. Landry, A. Petit, W. Ouyang, and N. Bellonzi. *Understanding the Surface Hopping View of Electronic Transitions and Decoherence*. Annual Review of Physical Chemistry **67**, 387 (2016). DOI: 10.1146/annurev-physchem-040215-112245. [pp 146, 154, 158, 159, and 164]

[173] J. C. Tully and R. K. Preston. *Trajectory Surface Hopping Approach to Nonadiabatic Molecular Collisions: The Reaction of H+ with D2*. The Journal of Chemical Physics **55**, 562 (1971). DOI: 10.1063/1.1675788. [p 146]

[174] N. Shenvi, J. E. Subotnik, and W. Yang. *Simultaneous-trajectory surface hopping: A parameter-free algorithm for implementing decoherence in nonadiabatic dynamics*. The Journal of Chemical Physics **134**, 144102 (2011). DOI: 10.1063/1.3575588. [pp 146, 175, 179, 187, and 195]

[175] M. F. Herman. *Nonadiabatic semiclassical scattering. I. Analysis of generalized surface hopping procedures*. The Journal of Chemical Physics **81**, 754 (1984). DOI: 10.1063/1.447708. [p 146]

[176] G. Granucci, M. Persico, and A. Zoccante. *Including quantum decoherence in surface hopping*. The Journal of Chemical Physics **133**, 134111 (2010). DOI: 10.1063/1.3489004. [p 146]

[177] G. Granucci and M. Persico. *Critical appraisal of the fewest switches algorithm for surface hopping*. The Journal of Chemical Physics **126**, 134114 (2007). DOI: 10.1063/1.2715585. [pp 146 and 154]

[178] A. White, S. Tretiak, and D. Mozyrsky. *Coupled wave-packets for non-adiabatic molecular dynamics: A generalization of Gaussian wave-packet dynamics to multiple potential energy surfaces*. Chem. Sci. **7**, 4905 (2016). DOI: 10.1039/C6SC01319H. [p 146]

[179] J.-Y. Fang and S. Hammes-Schiffer. *Comparison of surface hopping and mean field approaches for model proton transfer reactions*. The Journal of Chemical Physics **110**, 11166 (1999). DOI: 10.1063/1.479058. [p 146]

[180] J. E. Subotnik, W. Ouyang, and B. R. Landry. *Can we derive Tully's surface-hopping algorithm from the semiclassical quantum Liouville equation? Almost, but only with decoherence*. The Journal of Chemical Physics **139**, 214107 (2013). DOI: 10.1063/1.4829856. [p 146]

[181] C. J. Billington, C. J. Watkins, R. P. Anderson, and L. D. Turner. *A Monte Carlo wavefunction method for semiclassical simulations of spin-position entanglement*. arXiv:1502.06674 [physics, physics:quant-ph] (2015). ARXIV: 1502.06674. [pp 146 and 157]

[182] S. Dietrich and I. D. Boyd. *Scalar and Parallel Optimized Implementation of the Direct Simulation Monte Carlo Method*. Journal of Computational Physics **126**, 328 (1996). DOI: 10.1006/jcph.1996.0141. [p 146]

[183] K. Mølmer, Y. Castin, and J. Dalibard. *Monte Carlo wave-function method in quantum optics*. J. Opt. Soc. Am. B **10**, 524 (1993). DOI: 10.1364/JOSAB.10.000524. [pp 146, 147, and 170]

[184] H. M. Wiseman. *Quantum trajectories and quantum measurement theory*. Quantum and Semiclassical Optics: Journal of the European Optical Society Part B **8**, 205 (1996). [pp 146 and 170]

[185] G. C. Hegerfeldt. *The Quantum Jump Approach and Quantum Trajectories*. In F. Benatti and R. Floreanini (editors), *Irreversible Quantum Dynamics*, volume 622 of *Lecture Notes in Physics, Berlin Springer Verlag*, pages 233–242 (2003). [pp 146 and 170]

[186] M. B. Plenio and P. L. Knight. *The quantum-jump approach to dissipative dynamics in quantum optics*. Rev. Mod. Phys. **70**, 101 (1998). DOI: 10.1103/RevModPhys.70.101. [pp 147 and 170]

[187] E. Majorana. *Atomi orientati in campo magnetico variabile*. Il Nuovo Cimento (1924-1942) **9**, 43 (1932). DOI: 10.1007/BF02960953. [p 149]

[188] W. Petrich, M. H. Anderson, J. R. Ensher, and E. A. Cornell. *Stable, Tightly Confining Magnetic Trap for Evaporative Cooling of Neutral Atoms*. Phys. Rev. Lett. **74**, 3352 (1995). DOI: 10.1103/PhysRevLett.74.3352. [p 149]

[189] P. Ehrenfest. *Bemerkung über die angenäherte Gültigkeit der klassischen Mechanik innerhalb der Quantenmechanik*. Zeitschrift für Physik **45**, 455 (1927). DOI: 10.1007/BF01329203. [p 149]

[190] M. Born. *Zur Quantenmechanik der Stoßvorgänge*. Zeitschrift für Physik **37**, 863 (1926). DOI: 10.1007/BF01397477. [p 150]

[191] S. Aaronson. *Quantum Computing and Dynamical Quantum Models*. arXiv:quant-ph/0205059 (2002). arXiv: quant-ph/0205059. [p 150]

[192] P. Knight. *The Sinkhorn–Knopp Algorithm: Convergence and Applications*. SIAM Journal on Matrix Analysis and Applications **30**, 261 (2008). DOI: 10.1137/060659624. [p 156]

[193] N. Linial, A. Samorodnitsky, and A. Wigderson. *A Deterministic Strongly Polynomial Algorithm for Matrix Scaling and Approximate Permanents*. Combinatorica **20**, 545 (2000). DOI: 10.1007/s0049300700007. [p 156]

[194] P. A. Knight and D. Ruiz. *A fast algorithm for matrix balancing*. IMA Journal of Numerical Analysis **33**, 1029 (2013). DOI: 10.1093/imanum/drs019. [p 157]

[195] S. Aaronson. *Quantum Computing Since Democritus*. Cambridge University Press, New York, NY, USA (2013). [p 158]

[196] E. Fabiano, G. Groenhof, and W. Thiel. *Approximate switching algorithms for trajectory surface hopping*. Chemical Physics **351**, 111 (2008). DOI: 10.1016/j.chemphys.2008.04.003. [p 158]

[197] J. Schwinger, M. O. Scully, and B. G. Englert. *Is spin coherence like Humpty-Dumpty?* Zeitschrift für Physik D Atoms, Molecules and Clusters **10**, 135 (1988). DOI: 10.1007/BF01384847. [p 165]

[198] S. Machluf, Y. Japha, and R. Folman. *Coherent Stern–Gerlach momentum splitting on an atom chip*. Nature Communications **4**, 2424 (2013). DOI: 10.1038/ncomms3424. [p 165]

[199] D. W. Keith, C. R. Ekstrom, Q. A. Turchette, and D. E. Pritchard. *An interferometer for atoms*. Phys. Rev. Lett. **66**, 2693 (1991). DOI: 10.1103/PhysRevLett.66.2693. [p 165]

[200] E. M. Rasel, M. K. Oberthaler, H. Batelaan, J. Schmiedmayer, and A. Zeilinger. *Atom Wave Interferometry with Diffraction Gratings of Light*. Phys. Rev. Lett. **75**, 2633 (1995). DOI: 10.1103/PhysRevLett.75.2633. [p 165]

[201] M. A. Schlosshauer. *Decoherence: And the Quantum-To-Classical Transition*. The Frontiers Collection. Springer-Verlag, Berlin Heidelberg (2007). [pp 166 and 170]

[202] W. M. Itano, D. J. Heinzen, J. J. Bollinger, and D. J. Wineland. *Quantum Zeno effect*. Phys. Rev. A **41**, 2295 (1990). DOI: 10.1103/PhysRevA.41.2295. [p 169]

[203] B. Misra and E. C. G. Sudarshan. *The Zeno's paradox in quantum theory*. Journal of Mathematical Physics **18**, 756 (1977). DOI: 10.1063/1.523304. [pp 169 and 170]

[204] L. Fonda, G. C. Ghirardi, and A. Rimini. *Decay theory of unstable quantum systems*. Reports on Progress in Physics **41**, 587 (1978). [p 170]

[205] K. Jacobs and D. A. Steck. *A straightforward introduction to continuous quantum measurement*. Contemporary Physics **47**, 279 (2006). DOI: 10.1080/00107510601101934. [p 170]

[206] B. R. Landry and J. E. Subotnik. *How to recover Marcus theory with fewest switches surface hopping: Add just a touch of decoherence*. The Journal of Chemical Physics **137**, 22A513 (2012). DOI: 10.1063/1.4733675. [p 171]

[207] K. F. Wong and P. J. Rossky. *Solvent-induced electronic decoherence: Configuration dependent dissipative evolution for solvated electron systems*. The Journal of Chemical Physics **116**, 8429 (2002). DOI: 10.1063/1.1468887. [p 171]

[208] E. R. Bittner and P. J. Rossky. *Quantum decoherence in mixed quantum-classical systems: Nonadiabatic processes*. The Journal of Chemical Physics **103**, 8130 (1995). DOI: 10.1063/1.470177. [p 171]

[209] M. Ben-Nun, J. Quenneville, and T. J. Martínez. *Ab Initio Multiple Spawning: Photochemistry from First Principles Quantum Molecular Dynamics*. The Journal of Physical Chemistry A **104**, 5161 (2000). DOI: 10.1021/jp9941741. [p 175]

[210] M. Ben-Nun and T. J. Martıńez. *Nonadiabatic molecular dynamics: Validation of the multiple spawning method for a multidimensional problem*. The Journal of Chemical Physics **108**, 7244 (1998). DOI: 10.1063/1.476142. [p 175]

[211] S. Yang, J. D. Coe, B. Kaduk, and T. J. Martínez. *An "optimal" spawning algorithm for adaptive basis set expansion in nonadiabatic dynamics*. The Journal of Chemical Physics **130**, 134113 (2009). DOI: 10.1063/1.3103930. [p 175]

Word count

```
Total
Words in text: 66660
Words in headers: 487
Words outside text (captions, etc.): 11911
Number of headers: 132
Number of floats/tables/figures: 56
Number of math inlines: 2119
Number of math displayed: 332
Files: 7
Subcounts:
  text+headers+captions (#headers/#floats/#inlines/#displayed)
  241+3+0 (3/0/0/0) File(s) total: front_matter.tex
  1975+6+97 (2/0/12/0) File(s) total: introduction.tex
  7424+77+1305 (23/3/562/73) File(s) total: atomic_physics.tex
  23265+155+3406 (31/15/826/137) File(s) total: numerics.tex
  9364+84+1622 (26/7/7/0) File(s) total: software.tex
  7285+44+1990 (14/16/220/14) File(s) total: velocimetry.tex
  17106+118+3491 (33/15/492/108) File(s) total: hidden_variables.tex
```