The Dissertation Committee for Timothy Michael Coffey
certifies that this is the approved version of the following dissertation:

Non-Dynamical Quantum Trajectories

Committee:

Robert E. Wyatt, Co-Supervisor

William C. Schieve, Co-Supervisor

Herbert L. Berk

Linda E. Reichl

E.C. George Sudarshan

Non-Dynamical Quantum Trajectories

by

Timothy Michael Coffey, B.A.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2010

Non-Dynamical Quantum Trajectories

Publication No. _____

Timothy Michael Coffey, Ph.D.
The University of Texas at Austin, 2010

Supervisors: Robert E. Wyatt
William C. Schieve

Commonly held opinion is that particle trajectory descriptions are incompatible with quantum mechanics. Louis de Broglie (1926) first proposed a way to include trajectories in quantum mechanics, but the idea was abandoned until David Bohm (1952) re-invented and improved the theory. Bohm interprets the particle trajectories as physically real; for example, an electron actually is a particle moving on a well defined trajectory with a position and momentum at all times. By design, Bohm's trajectories never make predictions that differ from standard quantum mechanics, and their existence cannot be experimentally verified.

Three new methods to obtain Bohm's particle trajectories are presented. The methods are non-dynamical, and utilize none of Bohm's equations of motion; in fact, two of the methods have no equations for a particle's trajectory. Instead, all three methods use only the evolving probability density $\rho = \psi^* \psi$ to extract the trajectories. The first two methods rest upon probability conservation and density sampling, while the third method employs the

informational or geometrical construction of centroidal Voronoi tessellations. In one-dimension all three methods are proved to be equivalent to Bohm's particle trajectories. For higher dimensional configuration spaces, the first two methods can be used in limited situations, but the last method can be applied in all cases. Typically, the resulting higher dimensional non-dynamical trajectories are also identical to Bohm.

Together the three methods point to a new interpretation of Bohm's particle trajectories, namely, the Bohm trajectories are simply a kinematic portrayal of the evolution of the probability density. In addition, the new methods can be used to measure Schrödinger's wave function and Planck's constant.

Table of Contents

Chapter 1

Introduction

The trajectory concept proved to be quite successful in classical physics. Even to this day Newton's laws of motion are able to successfully describe and predict the trajectories of objects in ordinary situations. In the early stages of quantum mechanics, experiments with electrons challenged the viability of the trajectory description. Trajectories were abandoned as a successful fundamental descriptor of nature as a result of Heisenberg's uncertainty principle. In 1952, David Bohm created a new interpretation of quantum mechanics that duplicated all the predictions of standard quantum mechanics, but maintained the particle trajectory description. Debate occurred over the physicality of Bohm's trajectories, and even though debate still rages on, Bohm's theory has proved quite successful in the solution and interpretation of difficult quantum problems. None the less, the true nature of Bohm's trajectories is still an open question. In an effort to understand their true physical nature, new non-dynamical algorithms or methods are created, *without any equations of motion* from Bohm's theory, to generate the Bohm trajectories.

1.1 Trajectories in Quantum Mechanics

The motion of a classical object is not measured continuously but instead is measured at several discrete times. Shown by the dots in Figure 1.1 are the measurements at various times of some object's position. The motion

of the object between the measurements is now assumed to be a smooth trajectory (the dotted line in the figure) which passes near the observed events. Later this assumption of the object's trajectory can be experimentally verified by measuring the object's motion for times different than those used for the model.

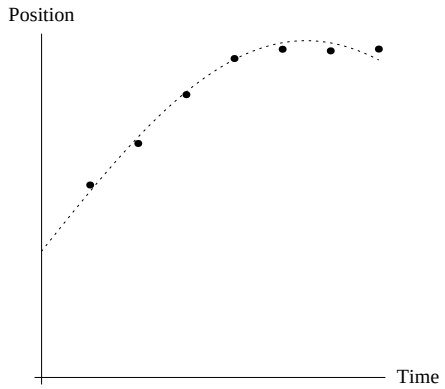


Figure 1.1: An assumed trajectory (dotted line) for a classical object given the discrete positional data (dots) collected at various times.

Historically, this method of exploration proved quite successful in describing the motion of objects in everyday situations. The synthesis of all the experimental trajectories led to the creation of Newton's second law of motion,

$$m\mathbf{a} = \mathbf{F}_n, \tag{1.1}$$

where $\mathbf{a}$ is the acceleration of the object, with mass m, due to the net force $\mathbf{F}_n$ acting on it. Armed with Newton's second law one could predict the trajectory of a classical object given only its mass, the net force, and some boundary conditions: initial event, initial velocity, etc.

Now suppose one wanted to perform a similar series of experiments with an electron. One immediate problem is that generally to detect an electron's

position, it must be absorbed into the detector, making later measurements of its position impossible. Therefore, instead of measuring the initial location of the electron, it is made to pass through a very small slit (see Figure 1.2), and then some time later its position is measured on a screen beyond the slit.

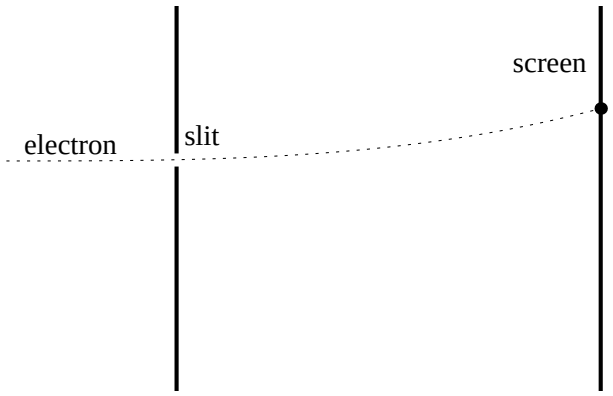


Figure 1.2: Instead of measuring the electron's initial position, it is made to pass through a small single slit. At some time later its position is measured on a screen beyond the slit.

With a small slit the confidence is high in the initial location of the electron as it passes through the slit. Between the slit and the screen there is no classical force, therefore, Newton's second law predicts that the electron should follow a uniformly straight trajectory from the slit to the screen. Therefore, classically the electron should be measured on the screen directly beyond the slit. If the experiment is repeated many times with identically prepared electrons passing through the slit, classical mechanics predicts that the electrons are grouped evenly right beyond the slit on the screen (Classical density in Figure 1.3). When the experiment is actually done, however, the electrons do not make a sort of inverse-shadow of the slit, but instead are spread out with some probability density on the screen (Quantum density in Figure 1.3). Given that an electron has passed through the slit, no exact prediction can be

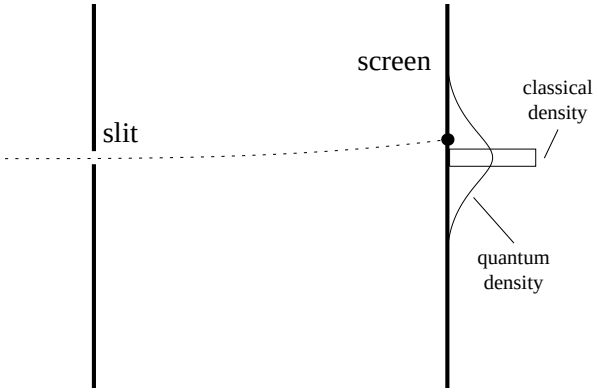


Figure 1.3: Each electron passing through the slit impinges upon a different location on the screen. Classically all of the electrons would land evenly directly beyond the slit. Experimentally, though, what is found is that the electrons form a smoothed out density on the screen that peeks at the location on the screen just beyond the slit.

made of the position it will land on the detection screen, since experimentally a smeared out probability density is obtained. If the slit width is altered the corresponding density pattern changes as well. For a smaller slit, the density pattern is more spread out, and vice-a-versa. This means that if the slit was infinitely small the density pattern on the screen would be infinitely spread out. In the attempt to pin down the initial position of the electron as it passes through the slit, the ability to predict the later position is completely lost.

> It is clear that this result can in no way be reconciled with the idea that electrons move in paths. ... In quantum mechanics there is no such concept as the path of a particle. ... The fact that an electron [or particle] has no definite path means that it has also, in itself, no other dynamical characteristics [except the parameters of mass and charge]. [46]

This electron slit experiment, though coming after the birth of quantum mechanics, exemplifies the need for quantum mechanics. One of the fundamental

principles of quantum mechanics is known as the *Heisenberg uncertainty principle*,

$$\Delta x \Delta p \geq \frac{\hbar}{2}. \tag{1.2}$$

The measured uncertainty of an object's position, Δx, implies a corresponding uncertainty of the object's simultaneously measured momentum Δp. Together these uncertainties cannot become infinity small, and are bounded below by Planck's constant $\hbar/2$. It is this principle that led many to conclude that the classical trajectory description is no longer accurate at the quantum level of nature. In David Bohm's 1951 quantum book he states,

> Since, in classical theory, a knowledge of the initial momentum and position of every particle is needed before the future orbits can be determined from the equations of motion, it is clear why this principle [Heisenberg's uncertainty principle] implies a quantum-mechanical limitation on the extent to which the deterministic description of classical theory can be applied. [10]

Classically, Newton's second law allowed one to calculate the trajectory of an object if the initial position and initial momentum are both known simultaneously. In quantum mechanics the uncertainty principle implies two possibilities for the descriptors position and momentum. The first possibility is that the position and momentum of a particle is actually undefined at the quantum level and hence do not exist, which implies that at the quantum level the uncertainty is fundamental, and it simply doesn't make sense to talk about a particle having a trajectory.

The second possibility is that the position and momentum do in fact always have definite values, but these values are simply unknown experimentally. The position and momentum would have definite values in reality, but

these variables are *hidden* from experiments, which means that the particle has a definite trajectory, but it can not be measured. The uncertainty in the measured position and momentum values comes only from the measurement interactions during the experiment. Again David Bohm, however, writes,

> The idea that a particle has simultaneously well-defined values of position and momentum, which are uncertain to us, is equivalent to the assumption of hidden variables ... that actually determine what these quantities are at all times, but in a way that, in practice, we cannot predict or control with complete precision. We shall see ... that quantum theory is inconsistent with the assumption of such hidden variables. [10]

The topic of hidden variables in quantum mechanics is quite extensive and could fill the pages of an entire book (or many books). In 1932, von Neumann [68] introduced the first argument to demonstrate that hidden variables were not compatible with quantum mechanics. This proof held for many years until Bell in 1964 [7] demonstrated that von Neumann's derivation contained a wrong assumption. Bell then showed that only a certain kinds of hidden variables were inconsistent with the predictions of quantum mechanics. Assuming that the hidden variables were local to the particle, the derived statements were in direct disagreement with quantum predictions. Thus Bell showed that local hidden variables were incompatible in quantum mechanics. Bell's theorem was later confirmed by the experiments of Aspect [3]. Since Bell, several hidden variable proofs or *no-go* theorems have been created [43, 51, 34, 6], but all basically reconfirm Bell's theorem, that local hidden variables lead to predictions that conflict with experiment.

With the overwhelming evidence against local hidden variable version of quantum mechanics, physicists concluded that nature is fundamentally un-

certain at the quantum level, and that particles do not move along definite trajectories. Quantum mechanics can produce an expression, however, that is quite similar to Newton's second law. This expression was first derived by Ehrenfest,

$$m\frac{d^2\langle \mathbf{x}\rangle}{dt^2} = m\langle \mathbf{a}\rangle = \langle \mathbf{F}\rangle \tag{1.3}$$

Therefore, Newton's second law is not an accurate description of each individual particle at the quantum level, but is only true statistically through an ensemble of identically prepared non-interacting particles.

In summary, the argument against a particle trajectory description in quantum mechanics begins with Heisenberg's uncertainty principle. Surprisingly, the principle is not sufficient by itself to forbid the trajectories, but with the combination of the various no-go and local hidden variable theorems has led many to conclude that the particle trajectory description is not viable in quantum mechanics.

1.2 Bohm's Theory

Prior to 1951 David Bohm subscribed to the standard version of quantum mechanics. In fact, his quotes included in the previous section came from his quantum mechanics book [10]. At that time, he agreed with other physicists that the trajectory description could not be used in quantum mechanics because of Heisenberg's uncertainty principle and the various hidden variable theorems (Bell's theorems had not been derived yet!). After the publication of the book, Bohm had a general distaste for the lack of a realism in quantum mechanics, and in 1952 published two papers creating a trajectory based description of quantum mechanics [11]. The Bohm trajectories circumvent the hidden variable theorems by being explicitly *non-local*. In fact, it was Bohm's

new theory that encouraged Bell to investigate the early hidden variable arguments, and then later show that only local hidden variables are forbidden in quantum mechanics.

1.2.1 Overview

Bohm's theory begins with the Schrödinger wave equation,

$$i\hbar\frac{\partial\psi}{\partial t} = -\frac{\hbar^2}{2m}\nabla^2\psi + V\psi. \tag{1.4}$$

In general the wave function ψ is a complex function in a multi-dimensional configuration space. A complex function can be written in polar form $\psi = Re^{iS/\hbar}$ for real and single-valued functions R and S. Substituting this general form of the wave function back into Schrödinger's equation, and into the complex conjugate equation as well, one obtains two expressions that are equivalent to the original equation. The first is a continuity equation,

$$\frac{\partial R^2}{\partial t} + \nabla\cdot\left(R^2\frac{\nabla S}{m}\right) = 0, \tag{1.5}$$

and the second is a quantum Hamilton-Jacobi equation,

$$\frac{\partial S}{\partial t} + \frac{1}{2}m\left(\frac{\nabla S}{m}\right)^2 + V + Q = 0, \tag{1.6}$$

where Q is known as the *quantum potential*,

$$Q = -\frac{\hbar^2}{2m}\frac{\nabla^2 R}{R}. \tag{1.7}$$

The quantum Hamilton-Jacobi equation is exactly the same as the classical Hamilton-Jacobi equation with only the addition the quantum potential. Classically the momentum of a particle is given by $\mathbf{p} = \nabla S$. In addition, the term

$\nabla S/m$ in the continuity equation looks like a velocity. Bohm posits, therefore, that the particle trajectories have a velocity given by the *guidance law*,

$$\mathbf{v} = \frac{\nabla S}{m}. \tag{1.8}$$

Defining the total time derivative as $d/dt = \partial/\partial t + \mathbf{v} \cdot \nabla$, one can derive a quantum version of Newton's second law,

$$m\frac{d\mathbf{v}}{dt} = -\nabla V - \nabla Q = F_{\text{classical}} + F_{\text{quantum}}, \tag{1.9}$$

which is true for each individual particle, unlike the situation with Ehrenfest's expression in Eq. 1.3.

The expressions above comprise Bohm's theory, which he summarized for an electron (though electron could mean any particle) [12]:

1. The electron actually *is* a particle with a well-defined position $x(t)$ which varies continuously and is causally determined.
2. This particle is never separate from a new type of quantum field that fundamentally affects it. The field is given by R and S or alternatively by $\psi = R\exp(iS/\hbar)$. ψ then satisfies Schrödinger's equation ... so that it too changes continuously and is causally determined.
3. The particle has an equation of motion

   $$m\frac{d\mathbf{v}}{dt} = -\nabla(V) - \nabla(Q).$$

   This means that the forces acting on it are not only the classical force $-\nabla V$, but also the quantum force, $-\nabla Q$.
4. The particle momentum is restricted to $\mathbf{p} = \nabla S$. Since the quantum field ψ is single valued it follows (as can easily be shown) that

   $$\oint p\, dx = nh.$$

   ...

5. In a statistical ensemble of particles, selected so that all have the same quantum field ψ, the probability density is $P = R^2$. ... if $P = R^2$ holds initially, then the conservation equation guarantees that it will hold for all time.[12]

Beyond particle trajectories, one of the most fundamental differences between Bohm's theory and standard quantum mechanics is the role of probability. In the widely accepted view of quantum mechanics, probability is a fundamental, which means a particle behaves probabilistic at the foundation of its evolution. In Bohm's theory, however, a particle moves along a specific trajectory at all times. The probabilities for Bohm are classical in that they are not fundamental but from the lack of knowledge about the initial positions of the particles. Bohm's theory also makes evident or obvious the *wholeness* that Bohr [13] referred to when discussing quantum mechanics,

> In our interpretation of quantum theory, we see that the interaction of parts is determined by something that cannot be described solely in terms of these parts and their preassigned relationships. Rather it depends on the many-body wave function ... that refers directly to the whole system ... *this is the most fundamentally new aspect* of the quantum theory. [12]

1.2.2 Properties

In this section are highlighted several more properties of Bohm's theory. In 1927 de Broglie introduced a similar theory, but it remained unrecognized until 1952 when Bohm re-discovered and improved it. The duality between wave and particle is non-existent in Bohmian mechanics, and the particle trajectories themselves behave uncommonly due to the non-local action of the quantum potential.

1.2.2.1 de Broglie Connection

Around 1926 de Broglie developed a particle trajectory based quantum mechanics which he called the *Double Solution*. At the time the theory had many mathematical difficulties that de Broglie was still wrestling with, so when he presented a paper at the 1927 Solvay conference he decided to discuss a simplified version of his ideas, which he named the *pilot-wave theory*. At the conference Pauli criticized de Broglie's pilot-wave idea by pointing out that the theory was inconsistent in dimensions greater than one [54]. Then in 1952 Bohm independently rediscovered the mathematics of de Broglie's pilot-wave theory, but supplied the necessary interpretation to counter Pauli's arguments.

The definition of de Broglie's Double Solution was,

> To every continuous solution $\psi = ae^{i\varphi/\hbar}$ of the equation of propagation of Wave Mechanics [Schrödinger's equation] there must correspond a singularity solution $u = fe^{i\varphi/\hbar}$ having the same phase φ of ψ, but with an amplitude f involving a generally mobile singularity [21].

The vision that de Broglie sought was a wave function ψ that was a solution of Schrödinger's equation. The wave function, which generally is in a high-dimensional configuration space, contained a phase φ. The phase was duplicated by a real space function u that contained singularities at the locations of the moving particles. The particles or singularities would need to move under the guidance law $\mathbf{p} = \nabla\varphi$. When de Broglie spoke at the Solvay conference he changed the Double Solution to the pilot wave theory, in which the u field is abandoned, and the particles are simply guided or piloted by the phase of the wave function through the same guidance law.

The notion of a pilot wave that guides the particles through the guidance law $\mathbf{p} = \nabla\varphi$, and the subsequent quantum potential, are really the overlap between de Broglie and Bohm's ideas. For de Broglie the wave function ψ could not be physically real since it generally propagates in a high-dimensional configuration space,

> It [Schrödinger's wave] must be merely a fictitious wave function of subjective character, capable only of giving us information of a statistical order about the various possible motions of the particles... [21]

On the other hand, Bohm considers the wave function ψ as physically real. From one of Bohm's adherents,

> ... we ascribe to configuration space as much physical reality as we do to three-dimensional Euclidean space ... [41]

And from Bell,

In the literature one usually finds references to the *de Broglie-Bohm* theory in an effort to give credit to both men for their contributions. But beyond the commonality of the guidance law and the idea of a pilot wave guiding the particles, the two approaches are different. The Double Solution theory of de Broglie has never successfully been extended to account for many body systems, while Bohm's interpretation easily deals with many body systems. In these respects it might be a mistake to refer to Bohm's theory by de Broglie-Bohm.

1.2.2.2 Wave and Particle Relationship

In Bohm's theory the wave-particle duality of standard quantum mechanics is replaced by *neither* a particle only description *nor* a wave only description, but rather a particle and wave description. An individual physical system is made of two parts, a point particle that evolves according to Eqs. 1.8 and 1.9, and a wave ψ which is a solution of Schrödinger's equation. The wave ψ is taken to be *physically real* even though it generally evolves in a multi-dimensional configuration space. In Bohm's picture, ψ only informs the particle where it needs to move *itself*, which is strikingly contrary to a classical wave in which the wave imparts energy and momentum to a particle being influenced by the wave, and vice versa. But the Bohm particle doesn't have influence on the guiding ψ wave, since Schrödinger's wave equation is sourceless.

1.2.2.3 Trajectory Behavior

Bohm's trajectories are similar to classical trajectories in only two aspects: 1) they are deterministic, and 2) they avoid *nodes* (regions of zero probability). All other behavior of Bohm's trajectories are generally *non-classical*. Bohm's trajectories do not cross while they evolve in configuration space, which is typically not true of classical trajectories. Classically a free particle moves in a uniformly straight line according to Newton's second law. But in Bohm's theory the free particle is being influenced by the quantum potential, which might not be zero, so a free particle will *not* move in a uniformly straight line.

One of the more interesting behaviors of the Bohm trajectories is for a stationary wave function. If the wave function can be written as $\psi = f(\mathbf{x})e^{ig(t)}$

for arbitrary functions f and g, then the Bohm particle is *at rest*! For example, the ground state of a hydrogen atom has a stationary wave function, and hence the electron will be at rest. This seemingly strange behavior is dismissed by Bohm since he states that the particle only has an intrinsic position that changes in time, and not an intrinsic momentum. The momentum that we measure for the particle actually comes from the measurement interaction,

> ... the momentum is not ... an intrinsic property. This will be true for all properties other than the position. [12]

Therefore, the particle does not have an intrinsic energy as well, and energy is not conserved along a Bohm trajectory.

1.2.2.4 Quantum Potential

The quantum potential is not a potential in the classical sense. A classical potential typically depends on the location of the particle (and maybe perhaps its velocity) and are time-independent, that is they are a pre-assigned function of particle positions. At each moment the particle at a particular place feels a certain force given by the gradient of the classical potential. The quantum potential, on the other hand, depends on the entire ensemble through the state of the guiding wave function. Therefore, it is not guaranteed that the force a particle feels at some location will be the same the next time it is at that location since the quantum potential is depends on the evolving wave function.

A more striking difference about the quantum potential is that it is independent of the intensity or strength of the guiding wave. Multiplying ψ by an arbitrary constant leads to exactly the same quantum potential due to

the fact that the wave amplitude is on both sides of the fraction in Eq. 1.7. So unlike an object being influenced by a classical wave, Bohm's quantum particle feels a force that is independent of the strength or amplitude of the wave. Therefore, one concludes that the quantum force is not mechanical in nature, and hence it does not conserve mechanical energy or momentum.

The most important characteristic about the quantum potential is that it brings quantum *non-locality* and *non-separability* explicitly to the forefront. Suppose a system is comprised of two particles. Using the Schrödinger equation for a two body system, and again writing $\psi = Re^{iS/\hbar}$ one gets that the quantum potential in this case is,

$$Q = -\frac{\hbar^2}{2m}\frac{(\nabla_1^2 + \nabla_2^2)R}{R}, \tag{1.10}$$

with subscripts 1 and 2 referring to each body in the system. We see here that even if the classical potential vanishes at large distances, the quantum potential generally does not. Therefore, even at very large distances the two bodies are still non-locally being influenced by each through the guiding ψ wave. If one of these bodies were a measuring apparatus, then it is obvious the measuring process interacts with the other body and what is measured depends on the contextuality, or environment, of the observed body.

1.2.3 Objections

Over the years Bohm's trajectory based version of quantum mechanics has weathered many objections. Passon's paper [53] provides a good account of the objections and the responses by Bohm supporters. In Table 1.1 is a list of the objections discussed in Passon's paper. Below are summarized several of these arguments that are relevant to the discussion herein.

Past Objections to Bohm's Theory
• Meta-theoretic debate
• Ockham's Razor
• Asymmetry
• Return to classical ideas
• Departure from established principles
• Under-determination of probability current
• Quantum equilibrium hypothesis
• Theory immanent debate
• Surreal trajectory objection
• Non-locality and relativistic generalization
• Cannot be extended to quantum field theory

Table 1.1: A list of past objections that have been used against Bohm's particle trajectory based theory of quantum mechanics. A complete summary of these objections and their responses is provided in O. Passon, *Why isn't every physicist a Bohmian?*, arXiv:quant-ph/0412119v2 (2005).

1.2.3.1 Ockham's Razor

Ockham's razor states that if two theories make the same predictions, then the theory that utilizes less assumptions (i.e. is simpler) should be the preferred theory. Bohm's theory, by design, makes the same predictions as standard non-relativistic quantum mechanics, yet it assumes the further construct of particle trajectories. Hence, Bohm's theory should not be the preferred theory and should be discarded. The following quote from Weinberg summarizes this sentiment,

> ... Bohm's quantum mechanics uses the same formalism as ordinary quantum mechanics, including a wave function that satisfies the Schrödinger equation, but adds an extra element, the particle trajectory. The predictions of the theory are the same as for ordinary quantum mechanics, so there seems little point in the extra

> complication, except to satisfy some a priori ideas about what a physical theory should be like. [70]

However, even though Bohm's theory makes the same predictions as standard quantum mechanics, and barring the additional construct of particle trajectories, the two theories are actually different. In Bohmian mechanics the notion of quantum measurement is totally dispensed with. A particle (or a pointer) is measured at some location because simply that is where the particle (pointer) was prior to the measurement; there was no collapse of the wave function. More importantly, in standard quantum mechanics probability is taken as fundamental, that is, nature is fundamentally probabilistic. But Bohm's theory introduces probability into quantum mechanics in a classical way. For Bohm the quantum mechanical probabilities are classical, due to the lack of knowledge of the initial starting positions of the ensemble of particles. This important distinction between Bohm's theory and standard quantum mechanics leads one to conclude that the two theories in fact are not entirely the same with just particle trajectories added on. Thus invoking Ockham's razor is not a valid objection against Bohm's theory.

1.2.3.2 Probability Current Under-determination

From Schrödinger's wave equation one can derive a continuity equation

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \mathbf{j} = 0, \tag{1.11}$$

where $\mathbf{j} = \frac{\hbar}{2mi}(\psi \nabla \psi^* - \psi^* \nabla \psi)$. This definition though is under-determined. That is one could add a divergence-less gauge to the probability current $\mathbf{j} \rightarrow \mathbf{j} + \mathbf{j}_a$ such that $\nabla \cdot \mathbf{j}_a = 0$, and the gauged current will still satisfy the required continuity equation. Therefore, it is unclear what the definition

of the probability current should be. The Bohm trajectories are equally under-determined, since the Bohm trajectories are defined $\mathbf{v} = \frac{\mathbf{j}}{\rho}$. There is an infinite number of possible definitions of the Bohm velocity field, each one satisfying the predictions of standard quantum mechanics. There has been several papers written to argue that the gauge freedom is not allowed. Holland used the non-relativistic limit of the Dirac equation to show that for spin-$\frac{1}{2}$ particles the guidance law is that of Bohm, but with an added spin dependent term [40]. A few years later a similar proof was done for spin-0 and spin-1 particles [66].

In one dimension though we can begin with Schrödinger's wave equation,

$$i\hbar\frac{\partial\psi}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2\psi}{\partial x^2} + V(x)\psi, \tag{1.12}$$

and the similar expression for the complex conjugate,

$$-i\hbar\frac{\partial\psi^*}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2\psi^*}{\partial x^2} + V(x)\psi^*. \tag{1.13}$$

Multiply the first expression by ψ^*, the second expression by ψ, and then subtract the second from the first,

$$i\hbar\left(\psi^*\frac{\partial\psi}{\partial t} + \psi\frac{\partial\psi^*}{\partial t}\right) = -\frac{\hbar^2}{2m}\left(\psi^*\frac{\partial^2\psi}{\partial x^2} - \psi\frac{\partial^2\psi^*}{\partial x^2}\right). \tag{1.14}$$

Writing the complex wave functions in polar form, $\psi(x,t) = R(x,t)e^{iS(x,t)/\hbar}$ for real functions R and S, and recognizing that $\rho(x,t) = R(x,t)^2 = \psi^*\psi$, we get that,

$$-\frac{\partial\rho}{\partial t} = \frac{\partial\rho}{\partial x}\frac{\partial S}{m\partial x} + \rho\frac{\partial^2 S}{m\partial x^2}. \tag{1.15}$$

Notice, that the right hand side of this continuity equation is not written as $\partial j/\partial x$ where $j = \rho\partial S/m\partial x$—this would have introduced an ambiguity or under-determinedness of the probability current since there is no unique anti-derivative. A generic one-dimensional continuity equation with velocity field

v is,

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x}(\rho v) = 0, \tag{1.16}$$

which can be solved for v,

$$v = -\frac{1}{\rho}\int_{-\infty}^{x} \frac{\partial \rho}{\partial t}\, dx. \tag{1.17}$$

Substituting Eq. 1.15 into this expression and performing an integration by parts with the boundary condition that the density ρ is zero at infinity leads to,

$$v = \frac{\partial S}{m \partial x} = v_{\text{Bohm}}, \tag{1.18}$$

by Eq. 1.8. Therefore, in one dimension the trajectories that satisfy the continuity equation are in fact the Bohm trajectories, and they are unique. Mentioned above were several arguments that showed that in higher dimensions as well that the Bohm guidance law is correct and, in general, unique.

1.2.3.3 Surreal Trajectories

An interesting approach to challenge Bohm's particle trajectory description in quantum mechanics was based upon presenting examples in which the Bohm trajectories appear to behave *unphysically*. For example, the two-slit experiment has the Bohm trajectories shown in Figure 1.5. The two slits are on the left side of the figure, and the Bohm trajectories depict the familiar bright-dark pattern on the screen located on the right of the figure. One immediately notices that particles from each slit are trapped by the horizontal line running down between the middle of the two slits. That is a detection on the top side of the screen is from a particle that went through the top slit, and likewise for the bottom side of the screen. Now suppose that single atom detectors are placed on each slit. The familiar bright-dark pattern on the screen

is lost due to the interaction with the atom detectors. In this situation the Bohm trajectories still are trapped on either side of the horizontal line between the slits, so every time a particle is detected on the top half of the screen the particle must have come from the top slit—similarly for the bottom half. But this is at odds with what might be recorded experimentally [28, 29, 25, 64], since half of the detections on the top half of the screen will have come from the bottom slit as known by the atom detectors at the slits.

Other examples of supposedly *surrealistic* Bohm trajectories involve protective measurements [2, 1]. It was shown that during the measurement the Bohm particles participate in the local interaction of the measurement though they might not be in the local region of the interaction. Again it was suggested that the Bohm trajectories can not be an accurate depiction of reality,

> Therefore we can hardly avoid the conclusion that the formally introduced Bohm trajectories are just mathematical constructs with no relation to the actual motion of the particle. [1]

These examples and the surreal objection are simply dismissed by realizing that one can not *a priori* judge the Bohm trajectories. The fact remains that the Bohm trajectories do what they have to do in order to maintain the predictions of standard quantum mechanics [38]. If they behave in such a way that is contrary to our classical prejudices then it is us that must move beyond our preconceived notions!

> These predictions are exactly the same as those obtained from standard quantum mechanics. There are no observable differences between standard quantum mechanics and the Bohm approach nor

> can there be simply because the Bohm approach uses the same wave functions and the same formalism as is used in the usual approach and therefore both approaches must end up with exactly the same probabilities. [37]

1.2.4 Successes and Applications

Despite the various objections raised against Bohm's particle trajectory description of quantum mechanics, it has had many successes and applications. Research has been pursued along two lines, referred to as the *analytic* and *synthetic* approaches. The analytic approach works from a solution of Schrödinger's wave equation to compute the Bohm trajectories with the aim of gleaning additional insight into quantum phenomena. The synthetic approach, however, aims to solve the time-dependent Schrödinger's wave equation by utilizing the Bohm trajectories as a computational platform.

Numerous examples have been done using the analytic approach. Beginning with simple diffraction, barrier tunneling, interference problems, to more complex problems of the Einstein-Podolsky-Rosen experiment (see Holland [41]). More modern examples include decoherence [61], atom surface diffraction [62], the Talbot effect [63], and vortices in semiconductor devices [5].

The most celebrated example, however, is still the analytic approach for the two-slit experiment. The calculation was first done by Philippidis et al. in 1979 [56] for two Gaussian slits. In Figure 1.4 the quantum potential for the two slits is shown. The figure is from the detection screen back towards the slits, which are the small peaks on either side of the central peaks. The quantum potential forces the particles from the tiny troughs onto the plateaus, thus making the familiar bright dark pattern of the two slit experiment. In standard quantum mechanics one can not discuss a particle going through one

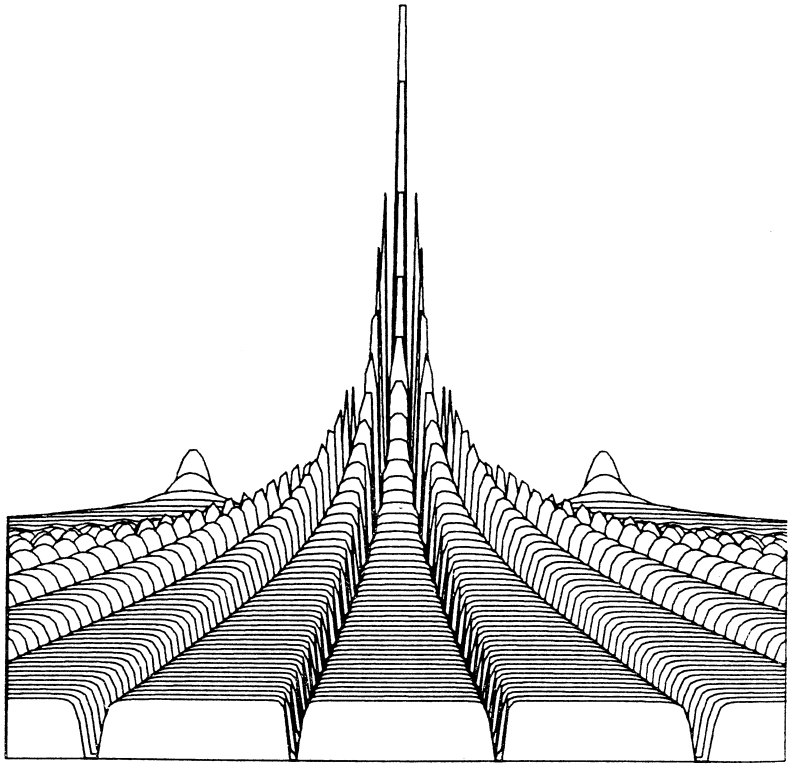


Figure 1.4: The quantum potential for a two Gaussian slit experiment as viewed from the detection screen back towards the slits. The two slits are the small peaks on the left and right side of the central peaks. The particles are forced from the small troughs onto the plateaus making the familiar bright dark pattern. From Philippidis et al., *Il Nuovo Cimento*, **52 B** (1979), 15.

slit or the other. In fact, one must say that the particle *went through both slits!* Bohm's trajectory description, however, shows (see Figure 1.5) that a particle always goes through one slit or the other (on the left of the figure). The ensemble of particles passing through the slits builds up the well known intensity pattern on the detection screen located on the right side of the figure.

Unlike the analytic approach, the synthetic approach *does not* solve Schrödinger's wave equation first to find Bohm's trajectories. Instead, the

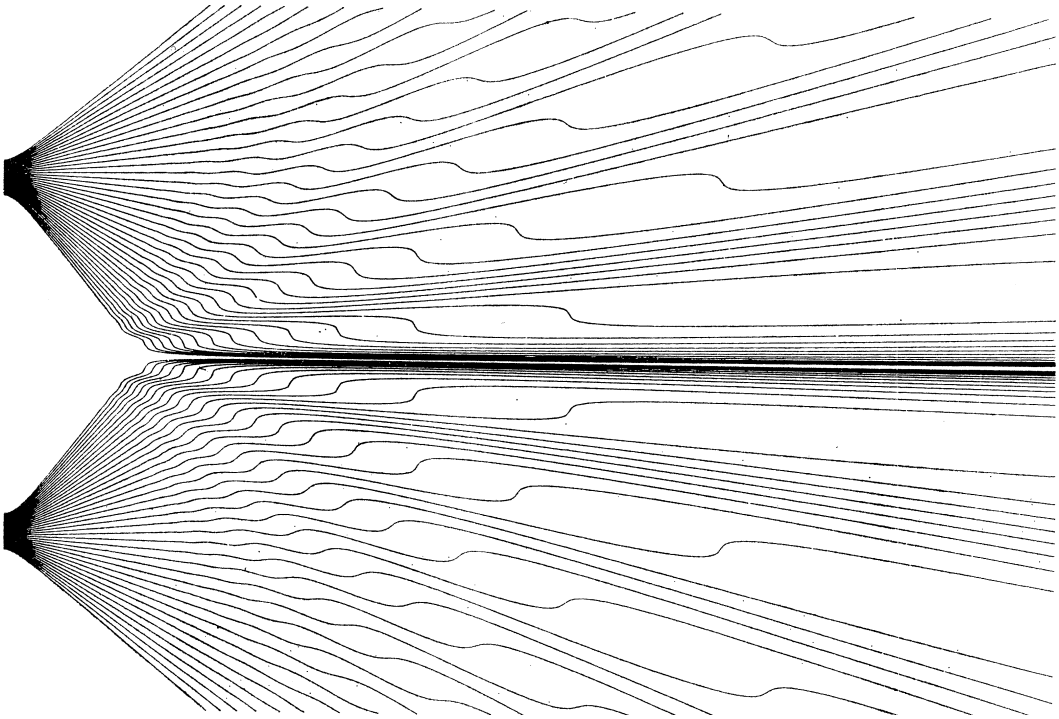


Figure 1.5: The Bohm trajectories for a two Gaussian slit experiment. The slits are located on the left side of the figure. On the right side of the figure one recognizes the well known bright-dark bands. From Philippidis et al., *Il Nuovo Cimento*, **52 B** (1979), 15.

wave function is computed in step with Bohm trajectories. The most common synthetic approach is the *quantum trajectory method* (QTM) [49, 72]. Here Schrödinger's wave equation is replaced by either one of two sets of three equations. The first set is referred to as the *force version* and include,

$$\frac{d\rho}{dt} = -\rho\nabla\cdot\mathbf{v} \tag{1.19}$$

$$m\frac{d\mathbf{v}}{dt} = -\nabla V - \nabla Q \tag{1.20}$$

$$\frac{dS}{dt} = L(t) = \frac{1}{2}m\mathbf{v}\cdot\mathbf{v} - (V+Q). \tag{1.21}$$

The second set of three is called the *potential energy version*,

$$\frac{d\rho}{dt} = -\rho\nabla\cdot\mathbf{v} \tag{1.22}$$

$$\frac{dS}{dt} = L(t) = \frac{1}{2}m\mathbf{v}\cdot\mathbf{v} - (V+Q) \tag{1.23}$$

$$\frac{d\mathbf{x}}{dt} = \mathbf{v} = \frac{1}{m}\nabla S. \tag{1.24}$$

Either version begins with an ensemble of particles located at what are known as *launch points*. From these launch points the Bohm trajectories are derived in conjunction with the density ρ and wave function phase S along the trajectories. The wave function along the trajectories $\mathbf{x}(t)$ is given by,

$$\psi(\mathbf{x},t) = \exp\left[-\frac{1}{2}\int_{t_0}^{t}(\nabla\cdot\mathbf{v})_{\mathbf{x}(\tau)}\ d\tau\right]\exp\left[\frac{i}{\hbar}\int_{t_0}^{t}L(\tau)\ d\tau\right]\psi(\mathbf{x}_0,t_0). \tag{1.25}$$

QTM solves the time-dependent Schrödinger equation on a set of grid points that are along the Bohm trajectories. This feature makes for an efficient scheme since the Bohm trajectories follow the main features of the evolution of the probability density [72]. An extreme example of the power of the synthetic approach calculated the trajectories and wave function for a multi-dimensional system with 200 vibrational modes, which was then used to calculate the time-dependent reaction probabilities [4].

Whether or not Bohm's particle trajectory description of quantum mechanics is an accurate depiction of nature, the theory does arm one with additional language and new computational tools in order to understand quantum phenomena, which in itself should warrant its study. But despite the many successes and applications of Bohm's trajectory description, the theory still is quite un-popular in the scientific community since the question of the physicality of the trajectories still remains open.

1.3 Non-Dynamical Quantum Trajectories

Bohm's particle trajectory description of quantum mechanics is a *dynamical* theory since there are equations of motion that describe the causes of how the trajectories evolve (Eqs. 1.5, 1.6, 1.7, 1.8, 1.9). What follows in the rest of this dissertation is an approach to understand more about the true nature of Bohm's trajectories. The following chapters demonstrate that numerous particle trajectory descriptions, other than Bohm's theory, can be created to still be consistent with the predictions of standard quantum mechanics. These other models need only to satisfy three requirements:

1. The particles trajectories must not cross in configuration space.

2. The density of particle trajectories must be equal to the probability density of quantum mechanics, $\psi^*\psi$.

3. The particle trajectories must be conserved since in non-relativistic quantum mechanics there is no particle creation or annihilation.

In order to understand Bohm's trajectories our alternative particle descriptions were developed to also abide by the following: 1) each model did not solve any dynamical equations of motion (contrary to Bohm's theory), and 2) that the model's trajectories were identical to Bohm's trajectories. Further, all models created utilized only the probability density $\psi^*\psi$, since this is the only experimentally verifiable quantity in quantum mechanics,

> ... in physics the only observations we must consider are position observations, if only the positions of instrument pointers. It is a great merit of the de Broglie-Bohm picture to force us to consider this fact. [8]

The main motivation for these additional restrictions was to address the question of the true nature of Bohm trajectories. An added benefit of these new methods of computing the Bohm trajectories allows one to experimentally determine Schrödinger's wave function (amplitude and phase), and Planck's constant.

In Chapter 2, the first method is discussed which utilizes a probability conservation statement [17] in order to generate the Bohm trajectories. This approach is shown to be identical to Bohm in one dimension, and in higher dimensions for *separable* wave functions. The next method based on density sampling [18] appears in Chapter 3, and employs no equations at all. The sampling method works in the same domain as the probability conservation method, and is shown to be a consequence of the first method. In Chapter 4 a final approach is described, which again uses no equations of motion, but appears to yield Bohm trajectories for many higher-dimensional separable and *non-separable* wave functions. This last method constructs the quantum trajectories by chaining together *centroidal Voronoi tessellations* (CVT) done at different times [19]. The ramifications to interpretation of the Bohm trajectories, and applications of these methods are discussed in Chapter 5. Finally in Chapter 6 a short summary and discussion of future work is provided.

Chapter 2

Probability Conservation Trajectories

Using only the probability density $\rho = \psi^*\psi$, trajectories can be defined by requiring that the each particle conserves total left (or right) probability[1]. Brandt et al. [15] first proposed this idea and described it as *quantile motion*. They argued that the quantile trajectories are identical to the Bohm trajectories, which while true in one dimension, their proof in higher dimensions failed to account for the gauge freedom in the definition of the quantum probability current. Their argument is refined to show that the method only works for one dimension, and in higher dimensions if the wave function can be written as a simple product of wave functions for each coordinate, in other words a *separable* wave function. Demonstrated are several numerical examples, which includes a two-slit experiment.

2.1 Probability Conservation Trajectory Method

Brandt et al. [15, 14] show quantum trajectories can be constructed by simply requiring that the total right (or left) probability is conserved for any particular trajectory,

$$Q = \int_{x_Q(t)}^{+\infty} \rho(x,t)\, dx = \text{constant}, \tag{2.1}$$

[1]Adapted from T.M. Coffey, R.E. Wyatt, and Wm.C. Schieve, *Uniqueness of Bohmian Mechanics, and Solutions From Probability Conservation*, arXiv:quant-ph:0710.4099v1

or for the total left probability (noting that $P + Q = 1$ for all time),

$$P = \int_{-\infty}^{x_P(t)} \rho(x,t)\, dx = \text{constant}. \tag{2.2}$$

The total left probability is also known as the *cumulative probability function*(CPF) for the probability density $\rho(x,t)$. The CPF is one-to-one and monotonically increasing. At each time, there is only one x_P that satisfies Eq. (2.2) for a constant value of P. Therefore, there is a *unique* trajectory such that the total left probability is conserved for all time. Given that P is constant,

$$\frac{dP}{dt} = \int_{-\infty}^{x_P(t)} \frac{\partial \rho}{\partial t}\, dx + \rho(x_P,t)\dot{x}_P = 0. \tag{2.3}$$

Where it's assumed that the density is zero at the lower boundary. This is solved for the unique velocity field for trajectories that conserve total left probability,

$$\dot{x}_P = -\frac{1}{\rho(x_P,t)} \int_{-\infty}^{x_P(t)} \frac{\partial \rho}{\partial t}\, dx. \tag{2.4}$$

The discussion has been quite general so far and Eq. (2.4) is the definition for trajectories given any density $\rho(x,t)$, *whether it be quantum or not.* To apply the quantile trajectories to quantum mechanics, Eq. (1.15) is inserted into Eq. (2.4),

$$\dot{x}_P = \frac{1}{\rho(x_P,t)} \int_{-\infty}^{x_P(t)} \left(\frac{\partial \rho}{\partial x} \frac{\partial S}{m \partial x} + \rho \frac{\partial^2 S}{m \partial x^2} \right)\, dx, \tag{2.5}$$

and an integration by parts of the integrand's second term (again assuming that the density is zero at the lower boundary) gives,

$$\dot{x}_P = \frac{\partial S}{m \partial x}, \tag{2.6}$$

which is the one-dimensional Bohm velocity field Eq. (1.8). The unique one-dimensional quantile trajectories—those that conserved total left (or right) probability—are in fact the Bohm trajectories in quantum mechanics.

The extension of the quantile motion into higher dimensions was also discussed in Brandt et al. [15]. They showed that instead of the total left (or right) probability being conserved in one dimension, that in higher dimensions the probability is conserved inside a volume enclosed by a surface of Bohmian trajectories. This property, however, is not unique to Bohmian mechanics. Any velocity field $\dot{\mathbf{x}}$ will conserve the probability inside a volume in configuration space since [72],

$$\frac{d\rho}{dt} = -\rho \nabla \cdot \dot{\mathbf{x}} \quad \text{and} \quad \frac{dJ}{dt} = +J \nabla \cdot \dot{\mathbf{x}}, \tag{2.7}$$

where ρ is the probability density and J is the Jacobian that describes the volume changes $dV(t) = J dV_0$. The probability inside this evolving volume is,

$$P_{\text{in}} = \int \rho \; dV(t) = \int \rho J \; dV_0, \tag{2.8}$$

which implies that $dP_{\text{in}}/dt = 0$. Any velocity field $\dot{\mathbf{x}}$, therefore, will conserve total probability inside a volume enclosed by a surface of trajectories following $\dot{\mathbf{x}}$. This is in contrast to what was found for the one-dimensional case above, where there was a unique velocity field that satisfied the total left (or right) probability conservation.

However, the quantile motion concept can be used to generate trajectories in higher dimensions if the marginal distribution for each coordinate is used, which is analogous to the CPF in one dimension. Suppose the system can be described by N Cartesian coordinates, then the corresponding definition of Eq. (2.2) is,

$$P_i = \int_{-\infty}^{x_i(t)} \rho_i(x_i, t) \; dx_i \qquad \text{for} \quad i = 1, 2, \ldots, N, \tag{2.9}$$

where $\rho_i(x_i, t)$ is the marginal distribution for the i-th coordinate. It's assumed that the particles are conserved so,

$$\frac{\partial \rho}{\partial t} + \sum_{i=1}^{N} \frac{\partial}{\partial x_i}(\rho \dot{x}_i) = 0. \tag{2.10}$$

Partially integrating the continuity equation yields,

$$\begin{aligned} &\int_{-\infty}^{+\infty} .. \int_{-\infty}^{x_i(t)} .. \int_{-\infty}^{+\infty} \frac{\partial \rho}{\partial t}\, dx_1 \ldots dx_N \\ &+ \sum_{j=1}^{N} \int_{-\infty}^{+\infty} .. \int_{-\infty}^{x_i(t)} .. \int_{-\infty}^{+\infty} \frac{\partial(\rho \dot{x}_j)}{\partial x_j}\, dx_1 \ldots dx_N = 0. \end{aligned} \tag{2.11}$$

Interchanging the partial derivative with respect to time and performing the $\pm\infty$ integrations, and again assuming that the density is zero at $\pm\infty$, only the $j = i$ integrals on the second term survive,

$$\begin{aligned} &\int_{-\infty}^{x_i(t)} \frac{\partial \rho_i}{\partial t}\, dx_i \\ &\qquad + \int_{-\infty}^{+\infty} .. \int_{-\infty}^{+\infty} \rho \dot{x}_i\, dx_1 \ldots dx_{\neq i} \ldots dx_N = 0. \end{aligned} \tag{2.12}$$

Then with the expression above, and differentiating Eq. (2.9) with respect to time,

$$\frac{dP_i}{dt} = \rho_i \dot{x}_i - \int_{-\infty}^{+\infty} .. \int_{-\infty}^{+\infty} \rho \dot{x}_i\, dx_1 \ldots dx_{\neq i} \ldots dx_N. \tag{2.13}$$

Hence, the i-th coordinate, in general, doesn't conserve total left probability of the marginal distribution since $\dot{x}_i$ could depend on the other coordinates. Suppose, however, that $\dot{x}_i = \dot{x}_i(x_i, t)$ (i.e. the motion along the i-th coordinate is independent), then $dP_i/dt = 0$, and the total left probability is conserved for the marginal distribution ρ_i.

In higher-dimensional Bohmian problems the guidance law Eq. (1.8) for the i-th coordinate becomes,

$$\dot{x}_i = \frac{1}{m} \frac{\partial S(x_1, x_2, \ldots, x_N; t)}{\partial x_i}. \tag{2.14}$$

If the wave function is separable, then,

$$\psi = \psi_1(x_1, t)\psi_2(x_2, t)\cdots\psi_N(x_N, t). \tag{2.15}$$

The probability density is also separable, $\rho = \rho_1(x_1, t)\rho_2(x_2, t)\ldots\rho_N(x_N, t)$, and the phase becomes $S = S_1(x_1, t)+S_2(x_2, t)+\cdots+S_N(x_N, t)$, which implies that $\dot{x}_i = (1/m)\partial S_i(x_i, t)/\partial x_i$. The velocity field for the i-th coordinate, therefore, is independent of the other coordinates for all $i = 1, 2, \ldots, N$. Hence, the one-dimensional total left (or right) probability conservation method can be used independently for each coordinate, to generate higher-dimensional Bohmian trajectories for a separable wave function.

2.2 Computing the Trajectories

The quantum probability density $\rho = \psi^*\psi$ is assumed to be known, and none of Bohm's equations of motion are used in the calculation. From Eq. (2.2) the total left probability is solved for $x_P(t)$ for each trajectory for constant values of P. Solving for $x_P(t)$ is known as the *inverse* CPF and can not, in general, be solved in closed form, and must be solved numerically.

A possible first approach to numerically solve the inverse CPF might be to find the root of or minimize,

$$\int_{-\infty}^{x_P(t)} \rho(x, t)\, dx - P, \tag{2.16}$$

where P is a constant value between zero and one for each particle trajectory. Typically though this avenue will most likely result in many numerical integrations of the the integral above. A simpler way to solve the inverse CPF is to first approximate the function by a series of trapezoids of equal width Δx, see Figure (2.1). From the P value for a particular trajectory, the corresponding

trapezoid is found. The x_P value at each time is then calculated by solving the linear equation for the top segment of the corresponding trapezoid. The number of trapezoids can be increased for a better approximation of the CPF curve, which will yield more accurate positions for the particles.

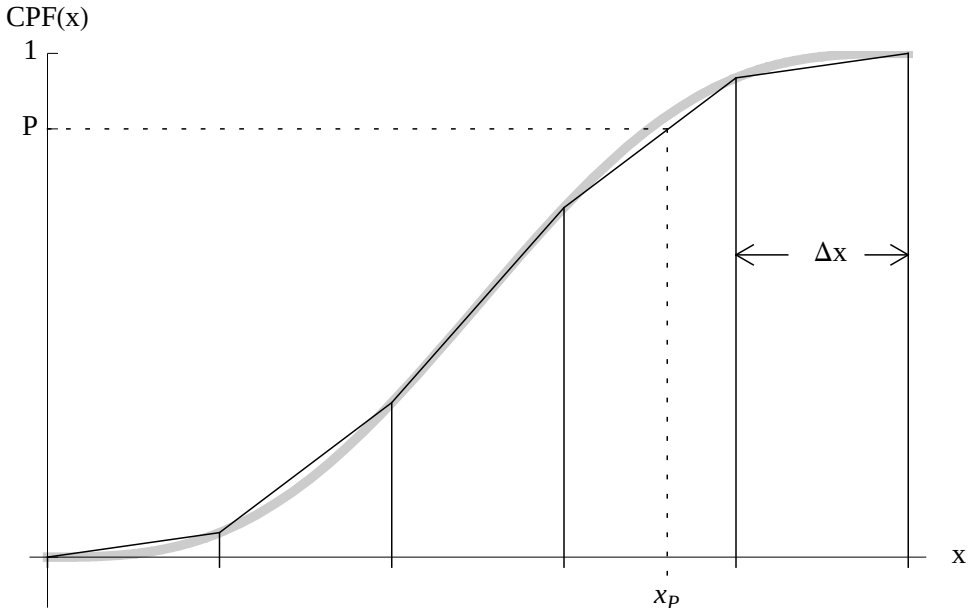


Figure 2.1: At each time step the cumulative probability function (CPF) curve (in grey) is approximated by a series of trapezoids of width Δx. The position x_P, corresponding to the constant quantile P value, is found by solving the linear equation of the top line of the particular trapezoid.

To compute the trajectories, at each time $t = n\Delta t$ ($n = 1, 2, 3 \ldots$) with time step Δt, the positions of the ensemble of particles is found by the steps described above. The positions from each time step for each particle are then linked together to form the trajectories. The size of the time step Δt can be adjusted to have smoother trajectories.

2.3 Examples

Several examples are shown that compare the quantile trajectories to the Bohm trajectories. Using the trapezoid method described above it was found that $\Delta x/\Delta t \approx 3$ gave nice results for the examples below. The first three examples are in one dimension, while the last example is for a two-dimensional separable wave function. In each example, the wave function is non-stationary so that $\partial\rho/\partial t \neq 0$. The probability density is computed from first solving Schrödinger's equation for ψ, and then $\rho(x,t) = |\psi(x,t)|^2$. The Bohmian trajectories can be numerically solved using Eq. (2.6) or more conveniently from this alternative form for the velocity field [12, 41],

$$\dot{x} = \frac{\hbar}{2mi}\left(\frac{\psi^*\partial\psi/\partial x - \psi\partial\psi^*/\partial x}{\rho}\right). \tag{2.17}$$

The quantile trajectories are computed numerically by the method described above in Section 2.2. In the figures below, the quantile trajectory points (+) are shown against the Bohm trajectory. In all these cases, quantile motion is able to reproduce the Bohm trajectories.

2.3.1 Infinite Square Well (1D)

A simple wave function in a one-dimensional infinite square well. The wave function is a superposition of the ground and first excited states,

$$\psi(x,t) = \frac{1}{\sqrt{L}}\left[\sin\left(\frac{n_1\pi x}{L}\right)e^{-iE_1t/\hbar} + \sin\left(\frac{n_2\pi x}{L}\right)e^{-iE_2t/\hbar}\right], \tag{2.18}$$

with $n_1 = 1$, $n_2 = 2$, $L = 1$, $\hbar = 1$, $m = \pi^2/2$, and $E_i = n_i^2\pi^2\hbar^2/(2mL^2)$. The number of particles used during the calculation was $N = 5$, with time from $[0, 3]$ with 30 equal steps. For the trapezoid approximation a grid of 50 divisions split up the well. See Appendix A for a listing of the *Mathematica* code

of this example. In Figure 2.2 are shown the resulting probability conserved trajectories (+) as compared to their corresponding Bohm trajectories (solid).

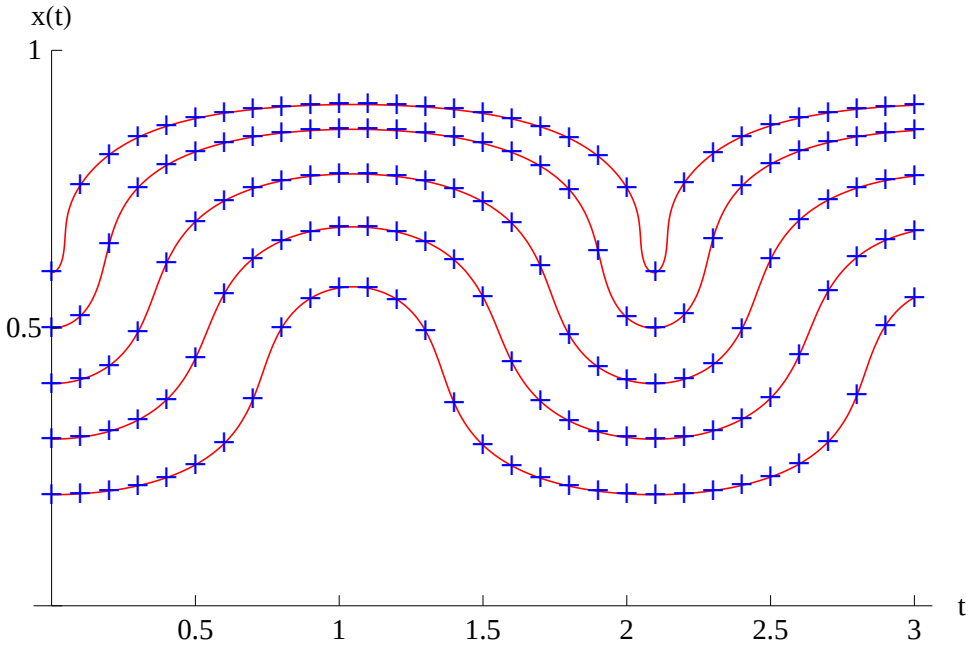


Figure 2.2: (color available). Infinite square well with wave function as the superposition of the ground and first excited states. The quantile trajectories (+) are shown with the Bohm trajectories (solid). The plot is in naturalized units.

2.3.2 Harmonic Oscillator

In this example, the harmonic oscillator wave function is taken to be a superposition of the ground and first excited states,

$$\psi(x,t) = \frac{1}{\sqrt{2}}\left(\sqrt{\frac{1}{a\sqrt{\pi}}}e^{-\frac{x^2}{2a^2}}e^{-iE_0t/\hbar} + \sqrt{\frac{1}{2a\sqrt{\pi}}}e^{-\frac{x^2}{2a^2}}\frac{2x}{a}e^{-iE_1t/\hbar}\right), \qquad (2.19)$$

where $a = \sqrt{\hbar/m\omega}$, and $E_j = \hbar\omega(j+1/2)$. Naturalized units were used so that $\hbar = 1$, $\omega = 3$, and $m = 1$. The range of the time was $t = n\Delta t \in [0,3]$ for

$n = 1, 2, 3, \ldots$, and the size of each time step was $\Delta t = 0.1$. The cumulative probability function [the right hand side of Eq. (2.2)] was approximated by a series of trapezoids (see Section 2.2) each with a width of $\Delta x = 0.2$, and the position range was $x \in [-5, 5]$ (an area where the density was essentially non-zero). In Figure (2.3), the quantile trajectory points (+) are plotted superposed on top of the corresponding Bohm trajectories. We see that the quantile trajectories are, in fact, the Bohm trajectories. Smaller trapezoid widths and time steps will produce more accurate and continuous trajectories.

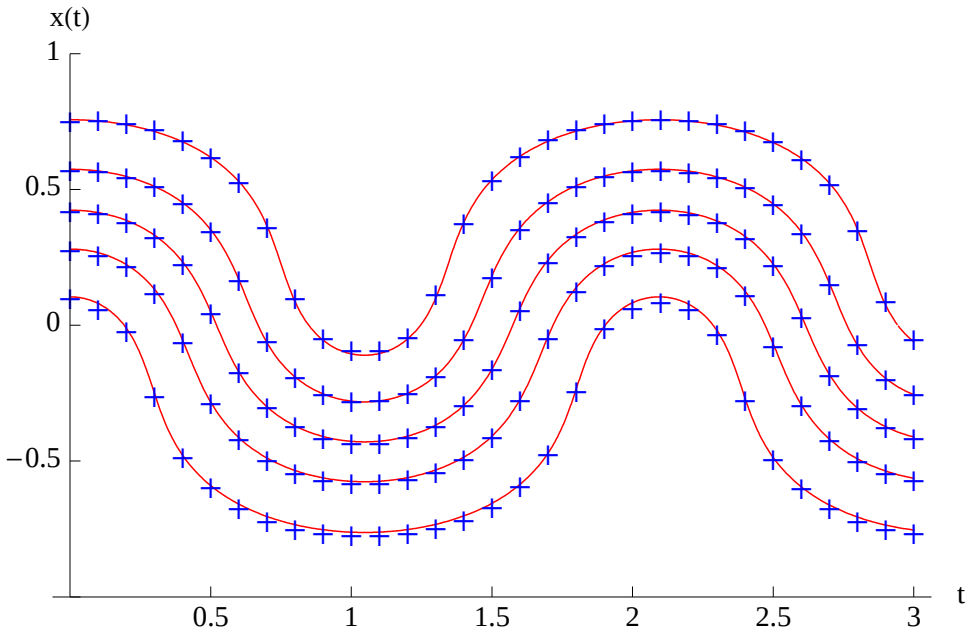


Figure 2.3: (color available). Harmonic Oscillator with wave function as a superposition of the ground and first excited states. The quantile trajectory points (+) are shown superposed on the Bohmian trajectories. The graph is in naturalized units.

2.3.3 Free Particle

The wave function for the free particle (assumed to be Gaussian initially with width a) is taken to be,

$$\psi(x,t) = \left(\frac{2a}{\pi}\right)^{1/4} \frac{e^{-ax^2/[1+(2i\hbar at/m)]}}{\sqrt{1+(2i\hbar at/m)}}. \tag{2.20}$$

Naturalized units were used so that $\hbar = 1$, $m = 1$, and $a = \pi/2$. Again, $t = n\Delta t \in [0, 3]$ ($n = 1, 2, 3, \dots$) with time steps of $\Delta t = 0.1$. The trapezoid widths (see Section 2.2) were $\Delta x = 0.2$, while the range was $x \in [-5, 5]$. In Figure 2.4, the quantile trajectory points (+) are shown against the Bohm trajectories. Notice that the ensemble of trajectories depict the familiar spreading of the wave function.

2.3.4 Two-Slit Experiment

This two-slit example is from §5.1.2 in Holland [41]. At first this problem seems to be two dimensional. However, the motion along the coordinate from the slits to the screen [x in Figure 2.5] is assumed uniform, thus the probability density is one dimensional and is only a function of y and t. To allow for easier calculation the experimental numbers were rescaled so that $\hbar = 1$, $m = 1$, and $t_{\max} = 100$ (the time between the slits and the screen), and then the results were rescaled back to the actual numbers. Using the trapezoid method as described in Section 2.2, the time $t = n\Delta t \in [0, t_{\max}]$ ($n = 1, 2, 3, \dots$) with a time step of $\Delta t = 2.5$ (1/40-th the total time). At each time, the cumulative probability function (this time a function of y) was approximated by a series of trapezoids of width $\Delta y = 3.24169$ (1/80-th the range of y). The range of y was restricted to a width between ± 129.668 where the probability density was essentially non-zero. In Figure 2.5, the quantile trajectory points (+) are

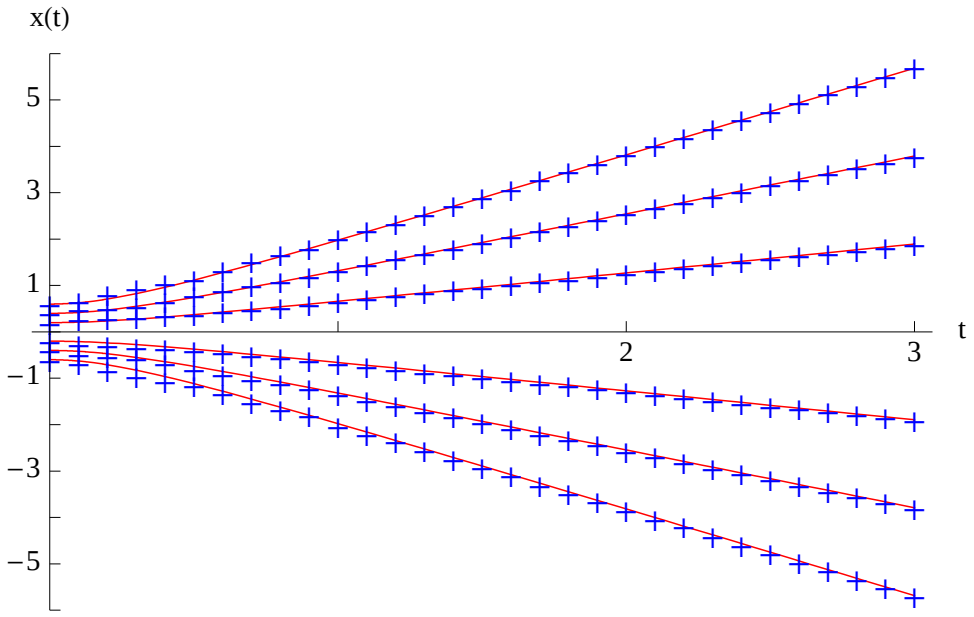


Figure 2.4: (color available). Free particle with an initial wave function of a Gaussian centered around zero. The quantile trajectory points (+) are shown superposed on the Bohmian trajectories. Even with only six trajectories shown the spreading of the wave packet is evident. The plot is using naturalized units.

plotted along with the Bohm trajectories. The ensemble of trajectories makes the familiar two-slit intensity pattern on the screen (located on the right hand side of the figure). The quantile trajectories match the Bohm trajectories quite well, even in those regions where the probability density is very close to zero (between the high intensity bands).

2.3.5 Infinite Square Well (2D-Separable)

In Figure 2.6 is a comparison of the quantile trajectories and their Bohm counterpart for the two-dimensional infinite square well. The separable

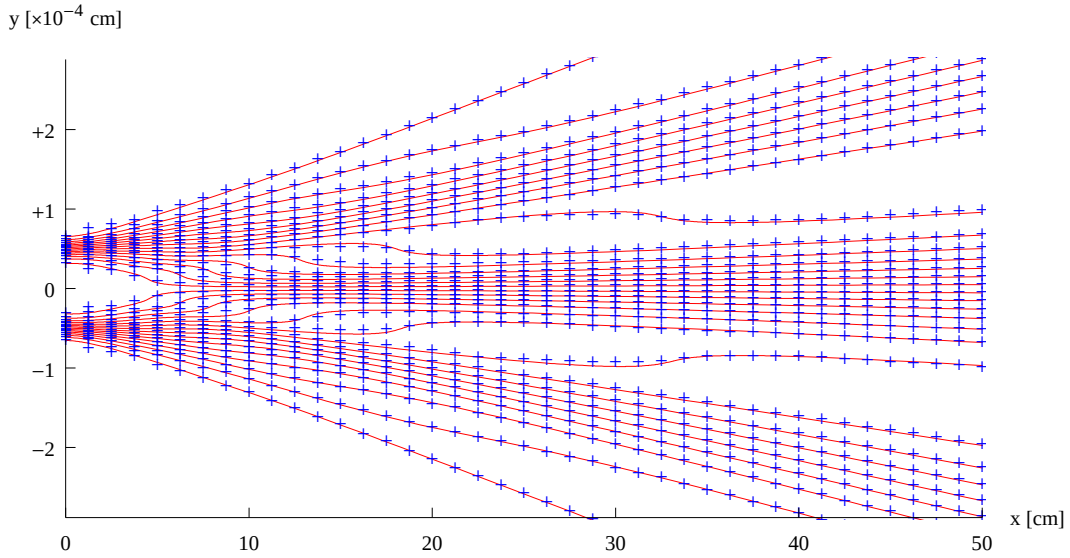


Figure 2.5: (color available). Two-Slit Experiment as described in §5.1.2 of P.R. Holland, *The Quantum Theory of Motion: An Account of the de Broglie-Bohm Causal Interpretation of Quantum Mechanics*, (Cambridge University Press, New York, 1993). The quantile trajectory points (+) are shown superposed on the Bohmian trajectories. The initial positions in each slit (left side of figure) are assumed to be Gaussian, and the ensemble of trajectories makes the familiar bands of bright and dark on the screen (right side of the figure).

wave function is assumed to be $\psi(x, y, t) = \psi_x(x, t)\psi_y(y, t)$, where,

$$\psi_x(x,t) = \sqrt{\frac{1}{L}}\left(\sin\left(\frac{\pi x}{L}\right)e^{-iE_1 t/\hbar} + \sin\left(\frac{2\pi x}{L}\right)e^{-i4E_1 t/\hbar}\right), \tag{2.21}$$

and a similar expression for $\psi_y(y, t)$. The energy $E_1 = \pi^2\hbar^2/2mL^2$, and naturalized units were used so that $m = 1$, $\hbar = 1$, and the width of the well in each direction taken to be $L = 1$. The time was $t = n\Delta t \in [0, 1]$ for $n = 1, 2, 3\ldots$, and the size of each time step was $\Delta t = 0.05$. The $(x(t), y(t))$ position of each particle was computed by approximating the cumulative probability function for each coordinate's marginal distribution by a series of trapezoids (see Sec-

tion 2.2) of width $\Delta x = \Delta y = L/30$. In Figure 2.6, the quantile trajectory points (+) are plotted superposed on top of the corresponding Bohm trajectories. For the separable wave function, the quantile trajectories are again identical to the Bohm trajectories.

2.4 Conclusion

To require that a quantum trajectory conserve total left (or right) probability leads to this expression,

$$\int_{-\infty}^{x_P(t)} \rho(x,t)\, dx = P_0, \tag{2.22}$$

where P_0 is a constant, which it some sense is an equation of motion for the particle's trajectory. The expression, however, is not a dynamic equation of motion since it does not concern itself with masses, forces, or potentials, so the equation above needs to be interpreted as non-dynamical or *kinematic*. This approach works in one dimension to reproduce the Bohm trajectories, and can be extended into higher dimensions if the wave function underlying the probability density is separable.

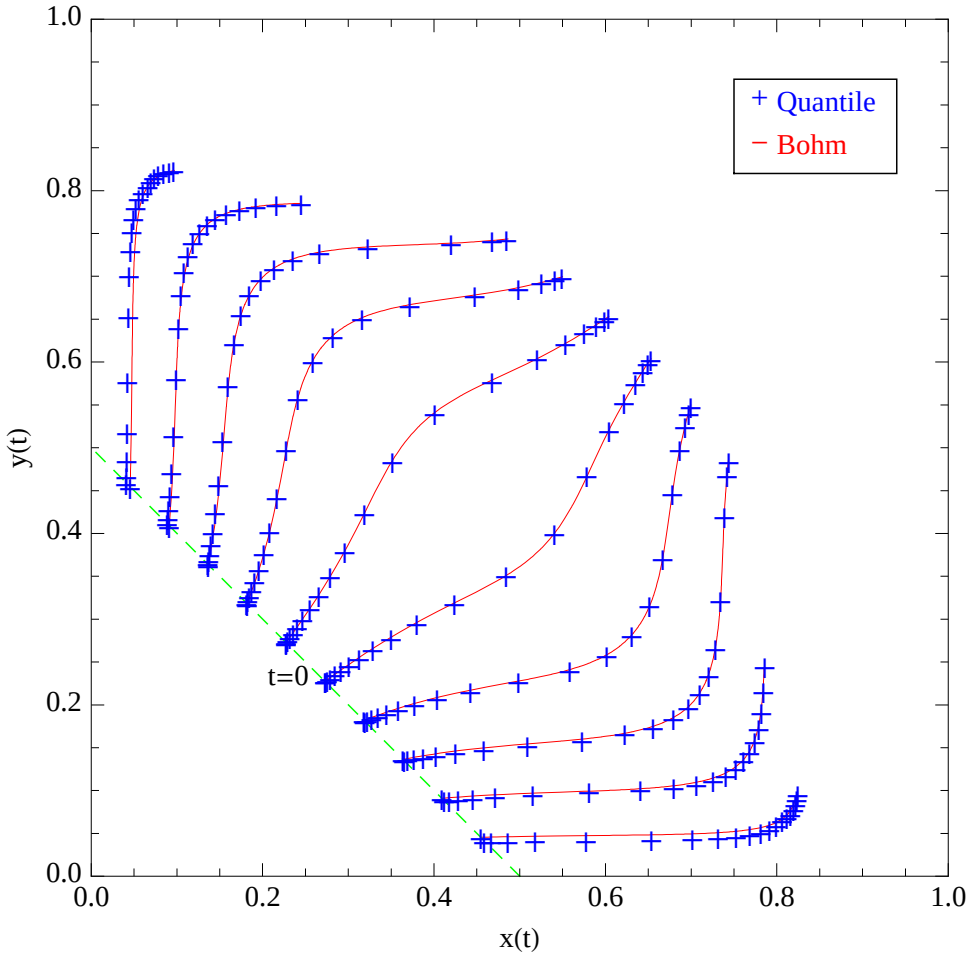


Figure 2.6: (color available). Comparison of the quantile trajectories (+) and the Bohm trajectories for a separable wave function in the two-dimensional infinite square well of size 1×1 in naturalized units. The initial position of each trajectory lies on the line from $(0.5, 0)$ to $(0, 0.5)$.

Chapter 3

Density Sampling Trajectories

The probability conservation trajectories of the last chapter still utilized an equation for the motion of each particle even though the expression was not a typical dynamical equation. Here a Monte Carlo method is described that generates one-dimensional trajectories for Bohm's formulation of quantum mechanics that does not involve differentiation or integration of any equations of motion[1]. At each time, N particle positions are randomly sampled from the quantum probability density. The positions are then sorted in order, and finally chained together with the positions at other times to form trajectories. The resulting trajectories are shown to be the Bohm trajectories in the limit that $N \to \infty$ and $\delta t \to 0$, where δt is the step between successive times. Like the probability conservation method in the previous chapter, the density sampling method works for one dimension, and in higher dimensions for separable wave functions.

3.1 Density Sampling

As with the probability conservation method, this new method assumes that the probability density, $\rho(x,t)$, is known and given. The evolution of

[1]Adapted from T.M. Coffey, R.E. Wyatt, and Wm.C. Schieve, *Monte Carlo generation of Bohmian trajectories*, J. Phys. A: Math. Theor. **41** (2008) 335304.

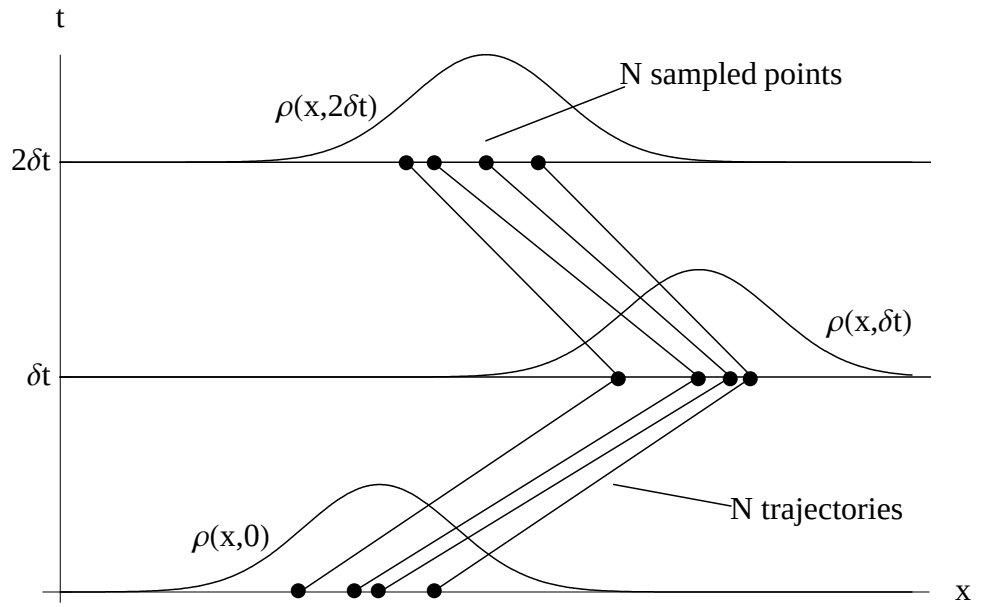


Figure 3.1: The Density Sampling Method. At each time $t = n\delta t$ ($n = 1, 2, 3, \dots$), N points are sampled from the probability density $\rho(x, t)$ and sorted. Trajectories are constructed by joining the i-th sorted point from each time step.

the density is depicted by an ensemble of N particles, see Figure 3.1. The trajectories of the N particles are constructed from locations at times $t = n\delta t$ ($n = 1, 2, 3, \dots$) with step size δt. At each time, the probability density is *sampled* to generate a set of N possible x-points. The x-points are sorted numerically. The i-th trajectory in the ensemble is built from the i-th x-point of the sorted N points at each time step. Though there are many ways to generate a set of N points sampled from a given distribution [35, 47], in the examples below (Section 3.3), we use the von Neumann acceptance-rejection method [69], see Figure 3.2, with a uniform proposal distribution ρ_U.

The quality of the trajectories generated relies on how one chooses the

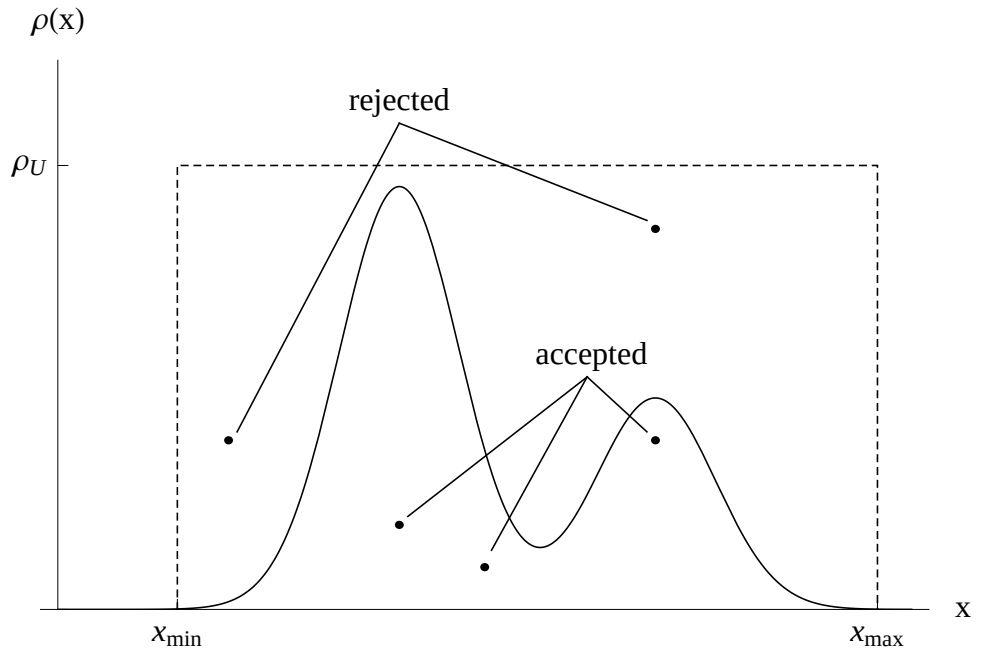


Figure 3.2: The von Neumann Acceptance-Rejection Method. For a given density $\rho(x)$, a set of x-points is generated by uniformly placing random dots on the graph. If a dot is under the density curve, that dot's x value is placed in the set.

number of particles in the ensemble N and the size of each time step δt. The following restrictions can be placed for these two parameters,

$$N \gg 2L\rho_{\max} \qquad \text{and} \qquad N\delta t \approx \frac{L}{\epsilon}, \tag{3.1}$$

where L is the width of the domain of the x coordinate (generally this is limited to a range of values where the density is essentially non-zero), ϵ is a small number $0 < \epsilon \ll 1$ with the dimensions of a speed, and $\rho_{\max}$ is the maximum value of the density for all positions $x \in L$ and for all times between the initial and final times.

3.2 Connection to Bohmian Mechanics

The method constructs the i-th trajectory from the i-th N sampled and sorted points at each time step. The particle trajectories, therefore, do not intersect by design (a familiar property of Bohmian trajectories). Hence, between successive time steps the approximate size of the maximum change in position δx is of the order $\delta x \approx 2L/N$. Identifying the density sampled trajectory as $x_{DS}(t)$, and the Bohm trajectory as $x_B(t)$, we assume at $t = 0$ that $x_{DS}(0) = x_B(0)$. We now describe the density sampled trajectory as $x_{DS}(t) = x_B(t) + \delta x(t)$ for some function $\delta x(t) \sim 2L/N$ such that $\delta x(0) = 0$. Computing the cumulative probability function (CPF) value Eq. (2.2) for the density sampled trajectory to first order in δx,

$$P_{DS} = \int_{-\infty}^{x_{DS}(t)} \rho(x,t)\, dx \approx P_B + \rho(x_B(t),t)\delta x(t). \tag{3.2}$$

Recall, the CPF value for the Bohm trajectory P_B is constant. From the restrictions and assumptions above, $\rho_{\max}\delta x \ll 1$, therefore, $\rho\delta x \leq \rho_{\max}\delta x \ll 1$, so the density sampled trajectory fluctuates about the Bohm trajectory.

While the parameter N places the location of the particle close to the actual Bohmian location at each time, the other parameter δt (the size of each time step) fixes the density sampled speed approximately equal to the Bohm speed. The difference in the two speeds is on the order of $\delta x/(2\delta t)$, where again δx is the size of the fluctuation about the Bohmian trajectory. From Eq. (3.1), we find that $\delta x/(2\delta t) \approx \epsilon \ll 1$, or that the difference in the speeds in quite small. Thus the density sampled trajectory will become the Bohm trajectory for $N \to \infty$ and $\delta t \to 0$.

3.3 Examples

In the examples below, the probability density is determined in the usual way $\rho(x,t) = |\psi(x,t)|^2$. The wave function was chosen to be non-stationary so that $\partial\rho/\partial t \neq 0$. For comparison, the Bohmian trajectories are computed from Eq. (1.8) or its equivalent,

$$\dot{x} = \frac{\hbar}{2mi}\left(\frac{\psi^*\partial\psi/\partial x - \psi\partial\psi^*/\partial x}{\rho}\right). \tag{3.3}$$

In all cases, the range L of possible position values x was limited to an area where the probability density was essentially non-zero. From Eq. (3.1), the number of particles in the ensemble (or the number of sampled points) N was approximately equal to $2L\rho_{\max} \times 10^3$, while ϵ was taken to have a numerical value of the order 10^{-3}.

3.3.1 Infinite Square Well

Again a simple wave function that is a superposition of the ground and first excited state in the infinite square well,

$$\psi(x,t) = \frac{1}{\sqrt{L}}\left[\sin\left(\frac{n_1\pi x}{L}\right)e^{-iE_1t/\hbar} + \sin\left(\frac{n_2\pi x}{L}\right)e^{-iE_2t/\hbar}\right], \tag{3.4}$$

with $L = 1$, $n_1 = 1$, $n_2 = 2$, and $E_i = n_i^2\pi^2\hbar^2/(2mL^2)$. Naturalized units were used so that the mass $m = \pi^2/2$ and $\hbar = 1$. During each time the probability density ρ was sampled $N = 10^4$ times. The time range for the calculation was $t \in [0,3]$ with 60 equal steps. In Figure 3.3 are shown five of the resulting density sampling trajectories (+) compared to their Bohm counterparts (solid). Even with the wave packet oscillating in the well, the density sampling trajectories exactly match the Bohm trajectories.

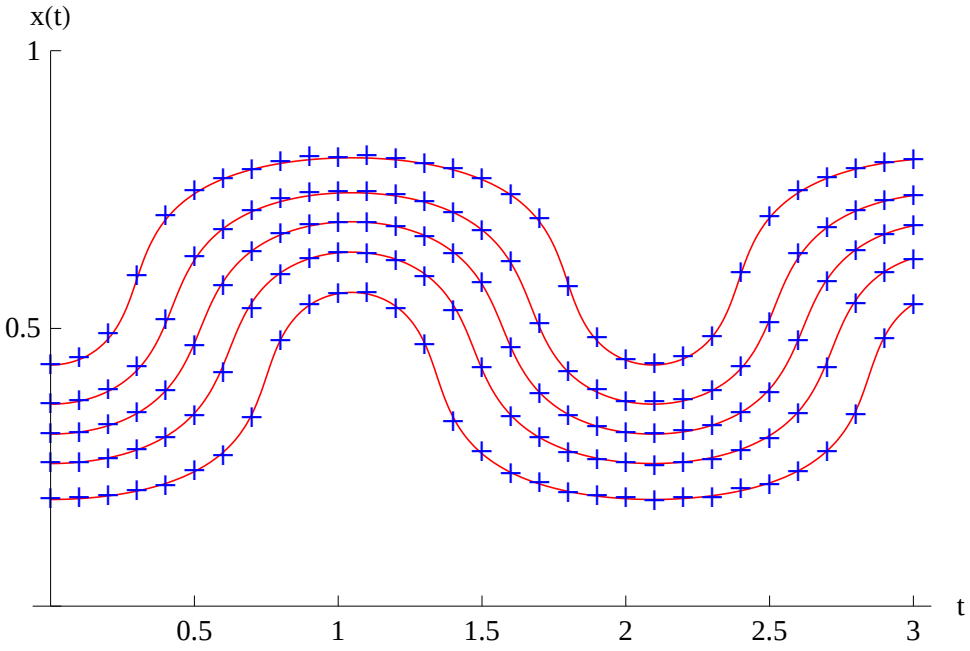


Figure 3.3: (color available). Infinite square well with wave function as a superposition of ground and the first excited states. The Bohmian trajectories are solid lines while the density sampled trajectories are plotted as plus (+) signs. The plot is in naturalized units.

3.3.2 Harmonic Oscillator

For this example a more complicated harmonic oscillator wave function was used which was a superposition of the ground state and the first three odd excited states,

$$\psi(x,t) = \frac{1}{2\sqrt{a\sqrt{\pi}}} e^{-\frac{x^2}{2a^2}} \sum_{n=0,1,3,5} \frac{H_n(x/a)e^{-iE_n t/\hbar}}{\sqrt{n!\ 2^n}} \tag{3.5}$$

where $a = \sqrt{\hbar/m\omega}$, $E_n = \hbar\omega(n+1/2)$, and H_n are the Hermite polynomials. Naturalized units were used so that $\hbar = 1$, $\omega = 3$, and $m = 1$. The range of time was $t \in [0,3]$, and the size of each time step was $\delta t = 0.1$. The range of the

possible positions was $x \in [-5, 5]$ (the area where the density was essentially non-zero). The number of particles in the ensemble was $N = 10^4$. Five of the resulting density sampled trajectories (+) are shown in Figure 3.4 against the actual Bohm trajectories. Notice that the density sampled trajectories are able to depict the complicated oscillatory behavior of the Bohm trajectories rather well. The *Mathematica* code for a simpler harmonic oscillator example is listed in Appendix B.

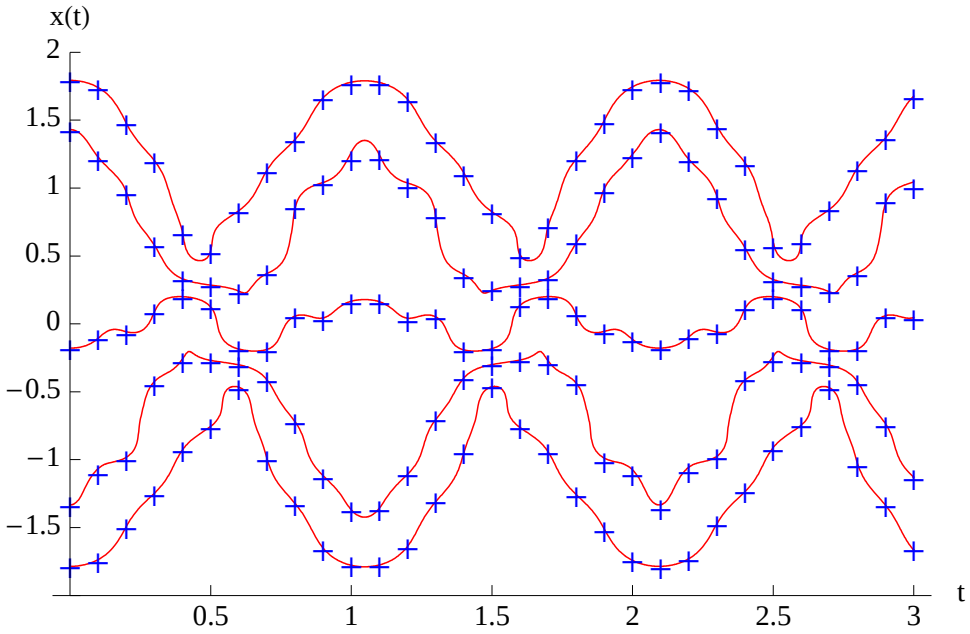


Figure 3.4: (color available). Harmonic oscillator with wave function as a superposition of ground and the first three odd excited states. The Bohmian trajectories are solid lines while the density sampled trajectories are plotted as plus (+) signs. The plot is in naturalized units.

3.3.3 Free Particle

The wave function for the free particle (assumed to be Gaussian initially with width a) was taken to be,

$$\psi(x,t) = \left(\frac{2a}{\pi}\right)^{1/4} \frac{e^{-ax^2/[1+(2i\hbar at/m)]}}{\sqrt{1+(2i\hbar at/m)}} \tag{3.6}$$

Naturalized units were used so that $\hbar = 1$, $m = 1$, and $a = \pi/2$. Again, $t \in [0, 3]$ with time steps of $\delta t = 0.15$. The number of particles in the ensemble was $N = 10^5$. Six of the resulting density sampled trajectories are plotted in Figure 3.5 superposed on top of their corresponding Bohm trajectory. The trajectories depict the familiar spreading of the wave packet.

3.3.4 Two-Slit Experiment

Again the two-slit example is from §5.1.2 in Holland [41]. Recall, that this problem seems to be two dimensional. However, the motion along the coordinate from the slits to the screen [x in Figure 3.6] is assumed uniform, thus the probability density is effectively in one dimension. To allow for easier computation the experimental values given in Holland's book were rescaled so that $\hbar = 1$, $m = 1$, and the total time between the slits and screen was $t_{\max} = 100$. The range of possible positions was $y \in [-129.668, +129.668]$. The number of particles in the ensemble was $N = 10^5$, and $\delta t = t_{\max}/30$. The resulting trajectories were then rescaled back to Holland's numbers for plotting. Thirty of the density sampled trajectories (+) are plotted against their Bohm counterpart in Figure 3.6. The trajectories manifest the familiar bright and dark bands of the two-slit intensity pattern on the screen (left side of figure). Also, notice the size of the fluctuations about the Bohm trajectory in the different regions. In high density regions the method does better since

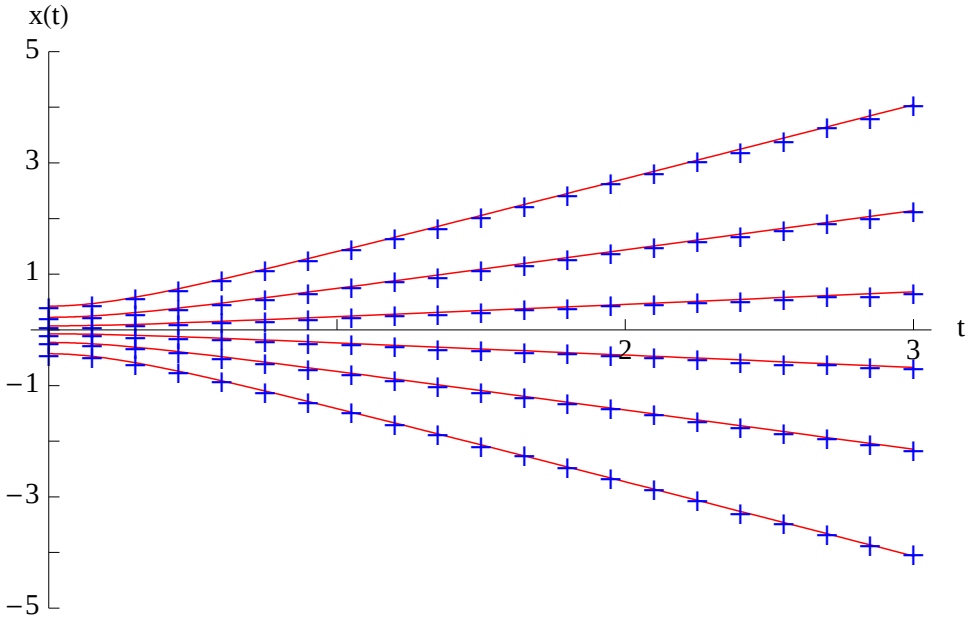


Figure 3.5: (color available). Free particle with an initial wave function of a Gaussian centered around zero. The Bohmian trajectories are solid lines, and the density sampled trajectories are plotted as plus (+) signs. The plot is in naturalized units. The ensemble of trajectories demonstrates the familiar spreading of the wave packet.

$\delta x \propto 1/N$ is smaller. But in low density regions (between the bright bands) δx is larger due to less particles being there.

3.4 Extension to Higher Dimensions

In general, the one-dimensional density sampling method described above can not be extended into higher dimensions. In higher dimensions there is no natural ordering to sort the N sampled points, and therefore, no way to consistently identify the i-th position in the ensemble at each time step like in the one-dimensional case. However, the quantile motion concept Eq. (2.2)

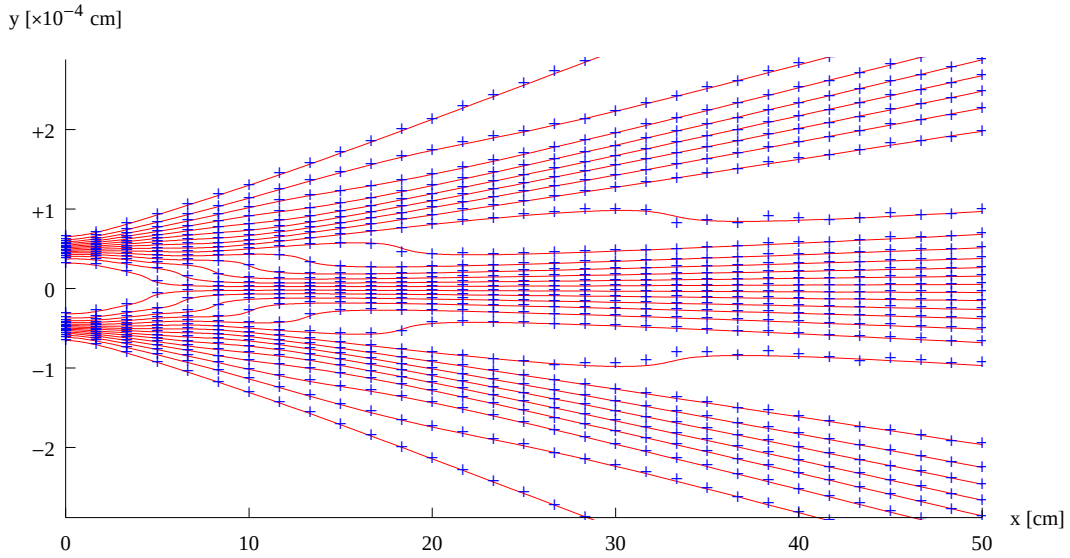


Figure 3.6: (color available). Two-Slit Experiment as described in §5.1.2 of P.R. Holland, *The Quantum Theory of Motion: An Account of the de Broglie-Bohm Causal Interpretation of Quantum Mechanics*, (Cambridge University Press, New York, 1993). The density sampled trajectories (+) are plotted superposed on the Bohmian trajectories (solid). The initial positions are assumed Gaussian in the slits (left side of figure), and the ensemble of trajectories makes the familiar bands of bright and dark on the screen (right side of the figure).

of the last chapter can be used independently on each coordinate in higher dimensions when the wave function is separable as shown in Section 2.1. Since the one-dimensional density sampled trajectory fluctuates about the Bohm trajectory defined by Eq. (2.2), the one-dimensional density sampling method can be used independently on each coordinate to generate higher-dimensional Bohm trajectories for those cases of a separable wave function. The method will also generate higher-dimensional Bohm trajectories for wave functions that are nearly separable [57], by applying the method independently on each

coordinate of the separable part of the wave function.

3.4.1 2D Example

In Figure 3.7 is a comparison of the density sampled trajectories and their Bohm counterpart for the two-dimensional infinite square well. The separable wave function was assumed to be $\psi(x, y, t) = \psi_x(x, t)\psi_y(y, t)$, where,

$$\psi_x(x,t) = \sqrt{\frac{1}{L}}\left(\sin\left(\frac{\pi x}{L}\right)e^{-iE_1 t/\hbar} + \sin\left(\frac{2\pi x}{L}\right)e^{-i4E_1 t/\hbar}\right), \tag{3.7}$$

and a similar expression for $\psi_y(y, t)$. The energy $E_1 = \pi^2\hbar^2/2mL^2$, and naturalized units were used so that $m = 1$, $\hbar = 1$, and the width of the well in each direction taken to be $L = 1$. The one-dimensional density sampling method was used independently for each coordinate. The time was $t = n\delta t \in [0, 1]$ for $n = 1, 2, 3\ldots$, and the size of each time step was $\delta t = 0.05$. The number of particles in each coordinate's ensemble was $N = 10^4$. Note, in general, a particle's place in the each coordinate's ensemble is not the same. The density sampled trajectory points (+) are plotted superposed on top of the corresponding Bohm trajectories. For the separable wave function, the density sampled trajectories are again identical to the Bohm trajectories.

3.5 Conclusion

Like the probability conservation method of the last chapter, the density sampling method reproduces Bohm's trajectories in one dimension and higher dimensions if the wave function is separable. The sampling method is very easy to implement and requires only three steps. First the probability density is sampled, then the sample points are sorted, and finally the points are chained together to form the trajectories. During an experiment the data

points are themselves a sampling of the true density, thus with sufficient data points at each time the quantum trajectories can be developed directly from experimental data. A fact that will be exploited in the last chapter.

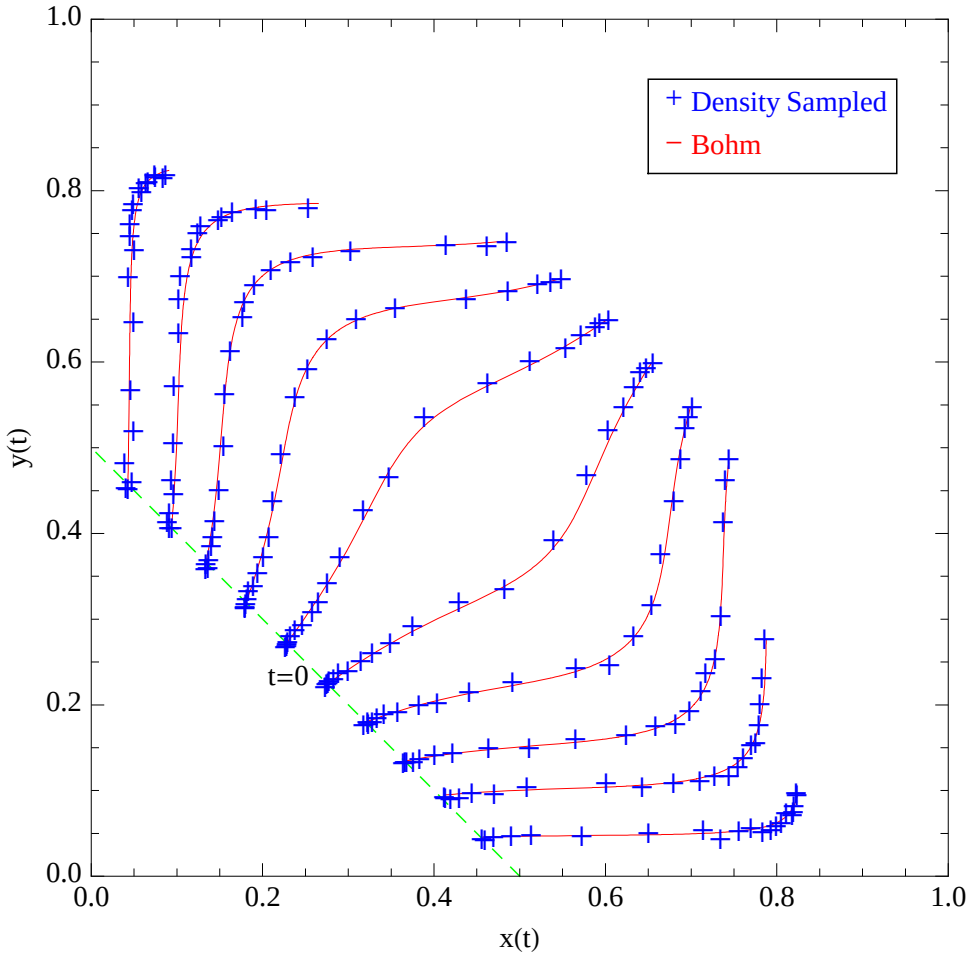


Figure 3.7: (color available). Comparison of the density sampled (+) and the Bohm trajectories for a separable wave function in the two-dimensional infinite square well of size 1×1 in naturalized units. The initial position of each trajectory lies on the line from $(0.5, 0)$ to $(0, 0.5)$.

Chapter 4

Centroidal Voronoi Tessellation Trajectories

In the last chapter, the density sampling method was able to reproduce the Bohm trajectories *without any equations of motion!* The only drawback of the density sampling method is it only works one dimension and for separable higher-dimensional wave functions. In this chapter a new method is introduced that overcomes this shortcoming[1]. Like the density sampling method, again a finite sample of the probability density is obtained at each time. The sample, however, is rearranged to minimize a novel error or distortion functional. The minimum arrangement of the particle positions form a *centroidal Voronoi tessellation*(CVT) of the configuration space. A particular minimizing process is used so that the identity of each particle is maintained during the calculation. Thus the trajectories again can be formed by chaining together the positions at different times. The last two examples in this chapter are for a non-separable wave function in a two-dimensional infinite square well. In each example, the CVT trajectories match the Bohm trajectories.

[1]Adapted from T.M. Coffey, R.E. Wyatt, and Wm.C. Schieve, *Quantum Trajectories from Kinematic Considerations*, J. Phys. A: Math. Theor. **43** (2010) 335301.

4.1 Density Representation

We begin by asking a more basic question: how does one represent a given probability density $\rho(\mathbf{x})$ by a finite set of N particles? In Figure 4.1(a) we show ten particles plotted along the x-axis that are one possible representation of the distribution $\rho(x)$ plotted above them. The representation, in this case, seems poor since there are no particles in the higher probability region. In part (b), however, the particles represent the distribution better and more uniformly. We quantify the goodness of the N particle representation by introducing a novel *error* or *distortion functional*,

$$D = \sum_{i=1}^{N} \int_{C_i} (\mathbf{x} - \mathbf{x}_i)^2 \rho(\mathbf{x})^\gamma \, d\mathbf{x}, \tag{4.1}$$

where $\gamma = (k+2)/k$, and k is the number of dimensions (the length of each position vector $\mathbf{x}_i$). The distortion functional introduced in Eq. (4.1) is similar to the distortion measure in the field of *vector quantization* or signal compression [36]. [We must note that word 'quantization' in this field has nothing to do with quantum mechanics.] For vector quantization the distortion functional is the same as in Eq. (4.1) except with $\gamma = 1$. In addition to the probability distribution, a particle density can be defined $\lambda(\mathbf{x}) \equiv \lim_{N\to\infty} N(\mathbf{x})/N$, where $N(\mathbf{x})d\mathbf{x}$ is the number of particles that are located in a small volume $d\mathbf{x}$ around $\mathbf{x}$. Our use of $\gamma = (k+2)/k$ in the distortion functional of Eq. (4.1) is necessary so that the particle density $\lambda(\mathbf{x})$ becomes the probability distribution $\rho(\mathbf{x})$ in the high-resolution or large N limit. Otherwise, for $\gamma = 1$ the particle density only becomes proportional to $\rho(\mathbf{x})^{k/(k+2)}$ as shown in Gersho and Gray [36].

The *best representation* of the probability density, the set of particle $\mathbf{x}_i$'s, is defined to be the one which minimizes the distortion. Each integration

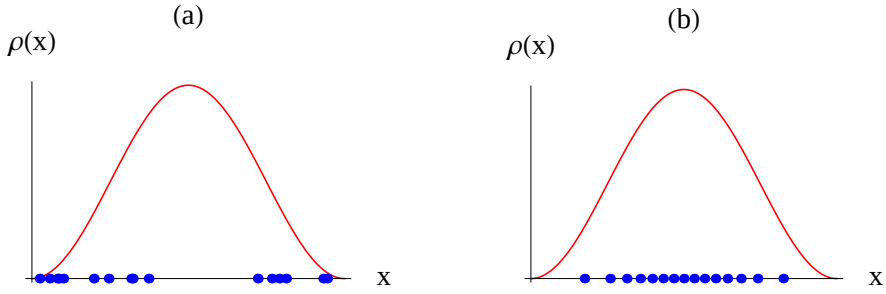


Figure 4.1: (color available) Two possible particle representations (the black dots on the x-axis) of a probability distribution $\rho(x)$. The representation in (a) seems poor since there are no particles in the higher probability region, while in (b) the particles seem to depict the probability distribution much better.

in Eq. (4.1) is taken over an exclusive volume or area C_i that surrounds each particle at $\mathbf{x}_i$. The C_i's are determined solely by the entire set of $\mathbf{x}_i$'s and the boundary conditions. A necessary condition for the particle positions to minimize the distortion is that they form a *centroidal Voronoi tessellation*(CVT) [36, 24]. This means that each particle location $\mathbf{x}_i$ is at the center of mass or centroid of its particular Voronoi volume or cell C_i, where [52],

$$C_i = \{\mathbf{x} \mid \ \| \mathbf{x} - \mathbf{x}_i \| \leq \| \mathbf{x} - \mathbf{x}_j \| \ \text{ for all } j \neq i\}. \tag{4.2}$$

Notice that minimizing the distortion functional to get the particle positions introduces *non-locality* since each particle's position depends on the positions of all the other particles in the ensemble.

The non-parametric method used to compute the CVT is the Lloyd-Max iterative deterministic algorithm (also known in the literature simply as the Lloyd algorithm) [48, 50]. In Figure 4.2(a) is shown a two-dimensional probability density at some particular time. The Lloyd-Max algorithm typically begins with a random sampling of the density as shown in part (b).

During each iteration the algorithm computes the Voronoi tessellation of the particle positions, then each particle is moved to the center of mass or centroid of its particular Voronoi cell C_i. The algorithm continues until some stopping criteria is satisfied; typically either a fixed number of iterations, or the maximum distance any one particle moves during the iteration is less than some small predetermined value. Shown in Figure 4.2(c) is the resulting CVT after 200 such iterations. Notice the uniformity of the structure in part (c) as opposed to the tessellation in part (b). The Lloyd-Max algorithm is beneficial since it is easy to implement, and has several non-degeneracy and global minimum or fixed-point convergence proofs in one and many dimensions [22, 23, 27]. More importantly, the algorithm keeps track of each particle's position during the entire computation, which is necessary for identifying each particle to build its trajectory.

4.2 Centroidal Voronoi Tessellation Trajectory Method

The CVT trajectory method begins by sampling the probability distribution at $t = 0$ to get N particle positions. One could begin with a predetermined array of particle positions, then the entire calculation would be deterministic, and not just in what follows. The initial sampling, then, is used to construct an initial CVT at $t = 0$. The positions of the initial CVT become the launch points for the particle trajectories. Time is then advanced a small amount δt. Rather than resampling the probability density at $t = \delta t$, the initial CVT's particle configuration is used as the input for the CVT computation at the new time. The Voronoi tessellation can be computed many ways (see Chapter 4 of Okabe *et al.* [52]), but in the examples below, Fortune's sweep line algorithm [30] was used. The calculation of the center of mass of each

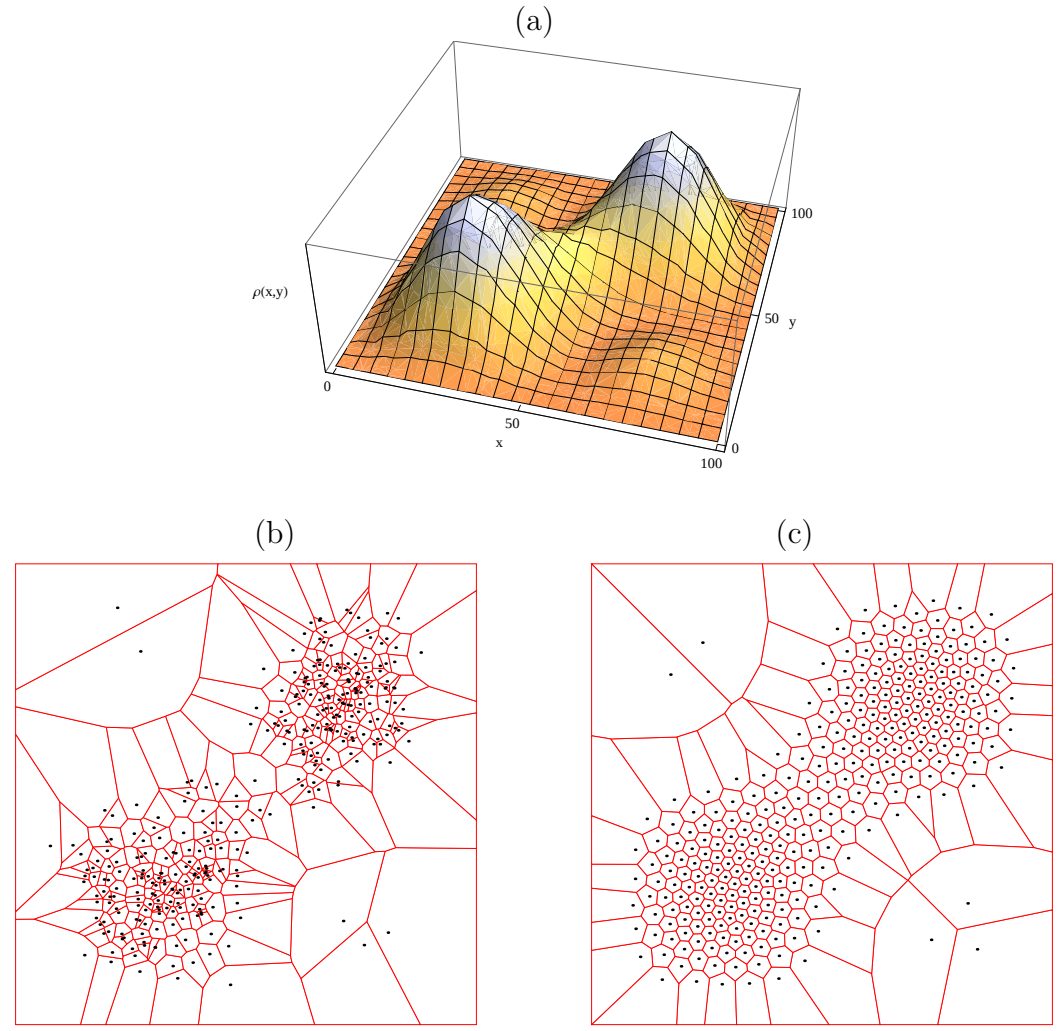


Figure 4.2: (color available) (a) An example of a two-dimensional probability density. (b) A Monte-Carlo sampling of the probability density. The single dots represent the possible particle positions. The straight lines are the Voronoi tessellation of these particle positions. (c) The centroidal Voronoi tessellation after 200 iterations of the Lloyd-Max algorithm that began from the initial sampling.

Voronoi cell C_i during the Lloyd-Max algorithm is done not with the common $\rho(\mathbf{x};t)$, but instead with $\rho(\mathbf{x};t)^{(k+2)/k}$ in keeping with the distortion functional in Eq. (4.1). The time steps keep advancing by δt until some predetermined time is reached. The computation of the CVT at each time begins with the particle positions of the previous time's CVT. Recall that the Lloyd-Max algorithm keeps track of each particle's position during the CVT computation. Therefore, one can chain the i-th particle's positions at the various times to form a trajectory. In Figure 4.3 is shown an example of a two-dimensional CVT at three times, and the construction of a particular trajectory. Notice that the CVT method will never have trajectory intersections, which is a familiar behavior of the Bohm trajectories as well.

At each time step, the method above relies on the minimization of the distortion functional of Eq. (4.1). Certainly, the minimum particle configuration (or fixed-point) is, in general, not unique. A simple example would be if the probability density exhibited any rotational symmetry. The CVT method, however, uses the previous time's fixed-point as the starting configuration for the minimization process at the new time. By making small time steps the fixed-point at the new time will be in a small neighborhood of the old fixed-point. Therefore, the evolution of the fixed-point (or minimum particle configuration) of the distortion functional will yield smooth trajectories. This feature is best illustrated in the two-dimensional free gaussian example below.

4.3 One-Dimensional Infinite Square Well

The CVT method simplifies greatly in one-dimensional calculations since the Voronoi tessellation just amounts to finding the midpoint between successive neighbors of the particle ensemble. To demonstrate the CVT and

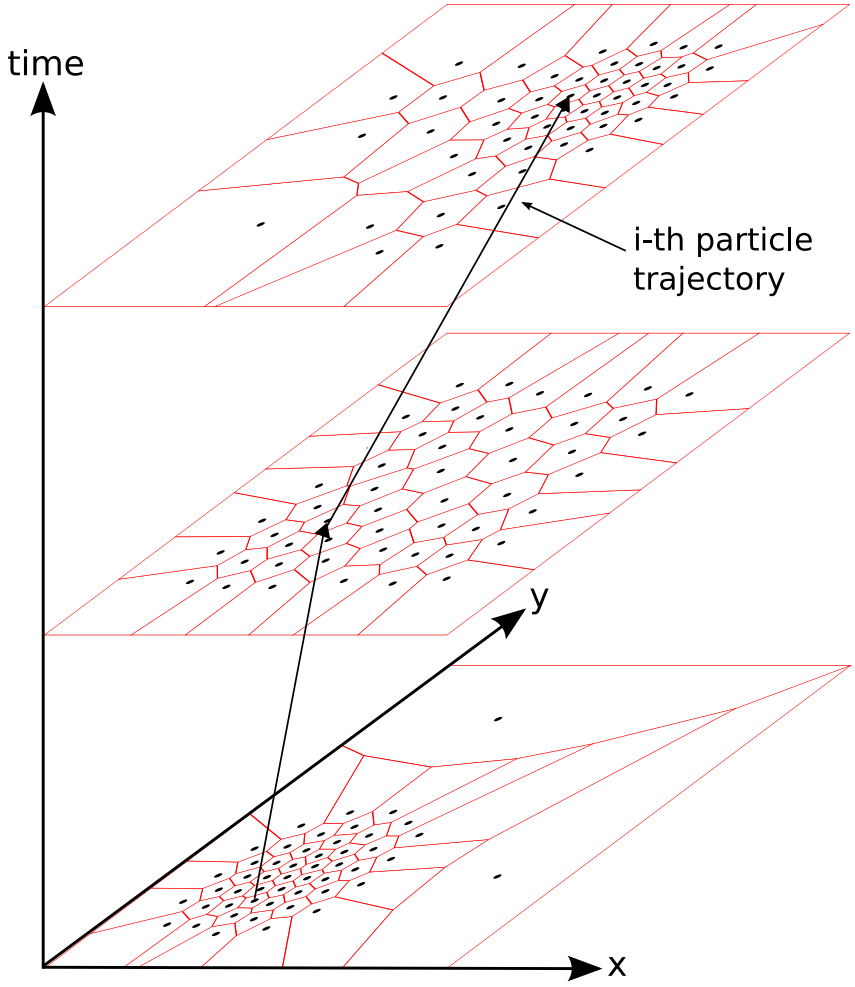


Figure 4.3: (color available) At each time a centroidal Voronoi tessellation(CVT) is computed using the particle positions from the previous time as input. The Lloyd-Max algorithm begins with the old positions, but uses the probability distribution at the new time. Each particle's position is tracked during each iteration of the Lloyd-Max algorithm. After the algorithm stops the trajectories are constructed by mapping a particle's old position to the new position as shown in the figure.

Bohm equivalence let's take for example a *non-stationary* state in a one-dimensional infinite square well,

$$\psi(x,t) = \sqrt{\frac{1}{L}}\left[\sin\left(\frac{\pi x}{L}\right)e^{-iE_1 t/\hbar} + \sin\left(\frac{2\pi x}{L}\right)e^{-iE_2 t/\hbar}\right], \tag{4.3}$$

where $E_n = n^2\pi^2\hbar^2/(2mL^2)$. For this example the following units were used: $\hbar = 1$, $m = \pi^2/2$, $L = 1$, and time was $t \in [0, 3]$ with 100 equal uniform time steps. The number of particles in the ensemble was $N = 256$, and the Lloyd-Max algorithm ran for a fixed number of 30,000 iterations. A selection of the resulting CVT trajectories (+) is shown in Figure 4.4 with their corresponding Bohm trajectories (solid line). The CVT trajectories match the Bohm trajectories even during the reversal of the wave packet's direction. The correlation coefficient for all $N = 256$ particles between the CVT trajectory positions and the Bohm positions was $r_x = 0.999$.

4.4 Two-Dimensional Examples

Presented here are three examples of the CVT method in two dimensions. The first example is the free gaussian wave packet. The probability density in this example has no unique fixed-point at any time because of rotational symmetry. Yet with small time steps the fixed-point does evolve smoothly, and hence so do the resulting CVT trajectories. The second example is for a *separable* wave function in a two-dimensional square well. Even though in one dimension the CVT trajectories are identical to the Bohm trajectories, it is not assured that a higher-dimensional separable example will yield the correct trajectories since the Voronoi tessellation has a completely different character in one and higher dimensions. A *non-separable* example in the same square well is done third. Each component of the two-dimensional CVT trajectories

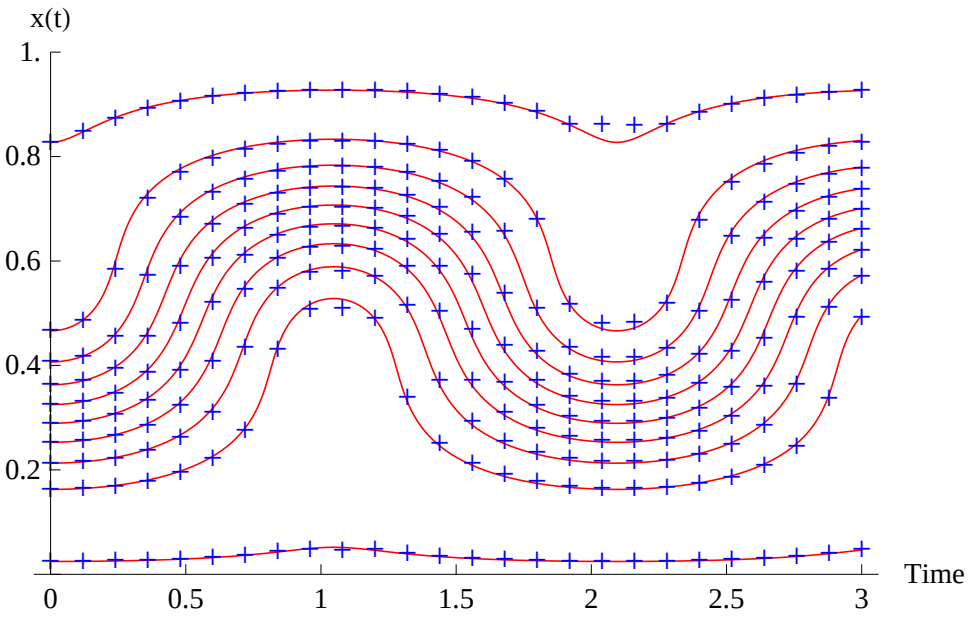


Figure 4.4: (color available) A comparison of the CVT trajectories (+) and the Bohm trajectories (solid line) for a wave packet in a one-dimensional infinite square well. The plot employs naturalized units ($\hbar = 1$).

are compared to the corresponding Bohm, $x(t)$ and $y(t)$, components. For all examples, the resulting CVT trajectories are highly correlated with the Bohm trajectories.

4.4.1 Free Gaussian Wave Packet

First, we begin with the free gaussian wave packet,

$$\psi(x, y; t) = \left(\frac{2a}{\pi}\right)^{1/4} \frac{e^{-a(x^2+y^2)/g(t)}}{\sqrt{g(t)}}, \tag{4.4}$$

where $g(t) = 1 + 2i\hbar at/m$. For the calculation, naturalized units were used such that $\hbar = 1$, $m = 1$, and $a = \pi/2$. The gaussian packet was placed in the exact center of a two-dimensional box with a width 100 on each side. The

time duration was $t \in [0, 10]$ with 20 equal time steps, and for each time step a fixed 400 iterations were done of the Lloyd-Max algorithm. In Figure 4.5 is a random subset of 20 trajectories from the total ensemble of $N = 400$ trajectories. The packet begins concentrated at the center of the box, and then spreads in time. For each CVT trajectory (+) the corresponding Bohm trajectory (solid line) is calculated. In the figure, we can see that the CVT trajectories match the Bohm trajectories quite well. For the whole ensemble the correlation coefficients between the components of the CVT and Bohm trajectories were $r_x = 0.996$ and $r_y = 0.997$.

4.4.2 Separable Wave Function in an Infinite Square Well

Next, shown is a non-stationary separable wave function $\psi = \psi_x \psi_y$ in a two-dimensional infinite square well with length L on both sides,

$$\psi_x(x;t) = \sqrt{\frac{1}{L}} \left[\sin\left(\frac{\pi x}{L}\right) e^{-iE_1 t/\hbar} + \sin\left(\frac{2\pi x}{L}\right) e^{-iE_2 t/\hbar} \right], \qquad (4.5)$$

with a similar expression for $\psi_y(y;t)$, and with energy $E_n = n^2\pi^2\hbar^2/(2mL^2)$. Again $\hbar = 1$, $m = 1$, units were used, and the box width was set to $L = 100$. The time interval was $t \in [0, 10,000\pi]$ with 48 equal time steps, and for each time step 300 iterations of the Lloyd-Max algorithm were performed.

The correlation coefficients between the one-dimensional components were $r_x = 0.967$ and $r_y = 0.962$. To better show correlation data concentrations, in Figure 4.6(a) the x positions were counted in 4×4 bins (the y positions show a similar plot); the diagonal dominance is evident. In part (b) a comparison of the trajectories for the particle with the worst correlation of the highest quartile—25% of the ensemble correlate better than this result. In part (c) is the trajectory with the best correlation of the lowest quartile—75%

of the trajectories correlate better. Despite the low number of Lloyd-Max iterations and particles in the ensemble, the CVT trajectory positions correlate quite well with the Bohm positions.

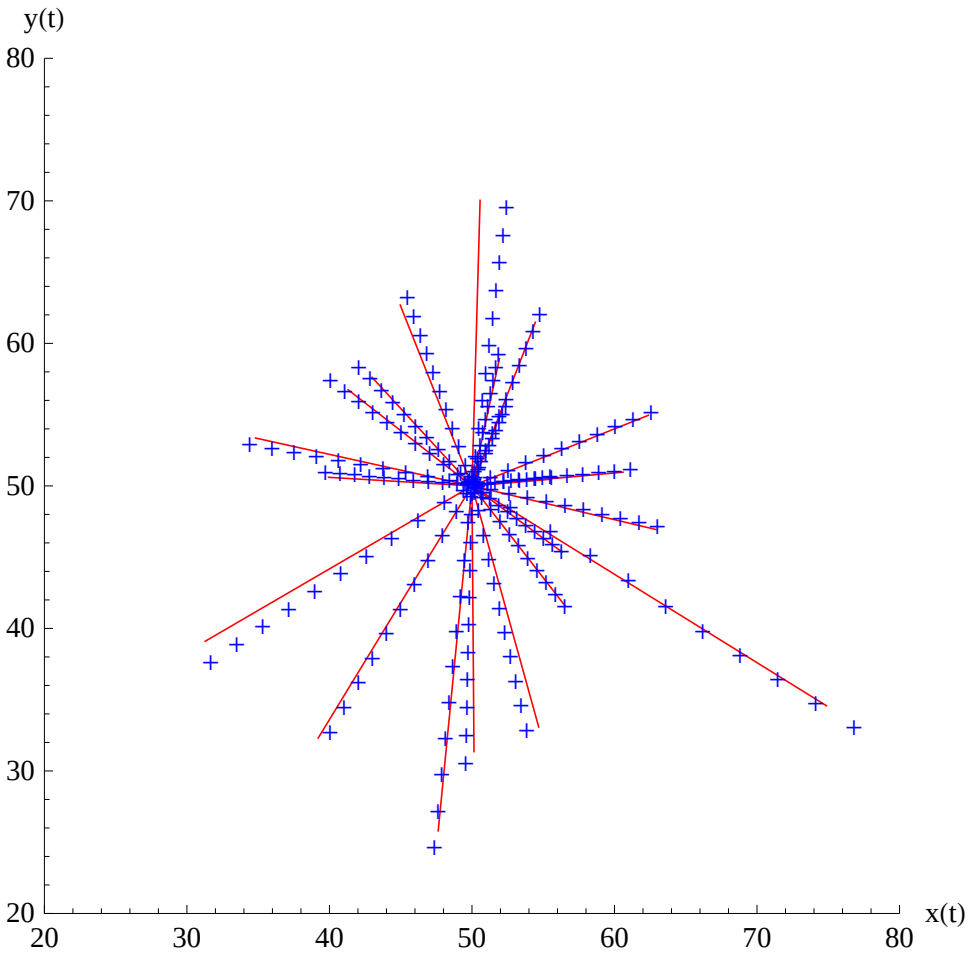


Figure 4.5: (color available) A comparison of the Bohm trajectories (solid line) and the CVT trajectories (+) for a two-dimensional gaussian wave packet. The gaussian begins concentrated at the middle of the figure, and as time progresses the gaussian spreads. The figure shows a random subset of 20 trajectories from the total of 400 particles used in the calculation.

4.4.3 Non-Separable Wave Function in an Infinite Square Well

Lastly, the non-stationary non-separable wave function in a two-dimensional infinite square well. The parameters were $\hbar = 1$, $m = \pi^2/2$, $L = 100$, and

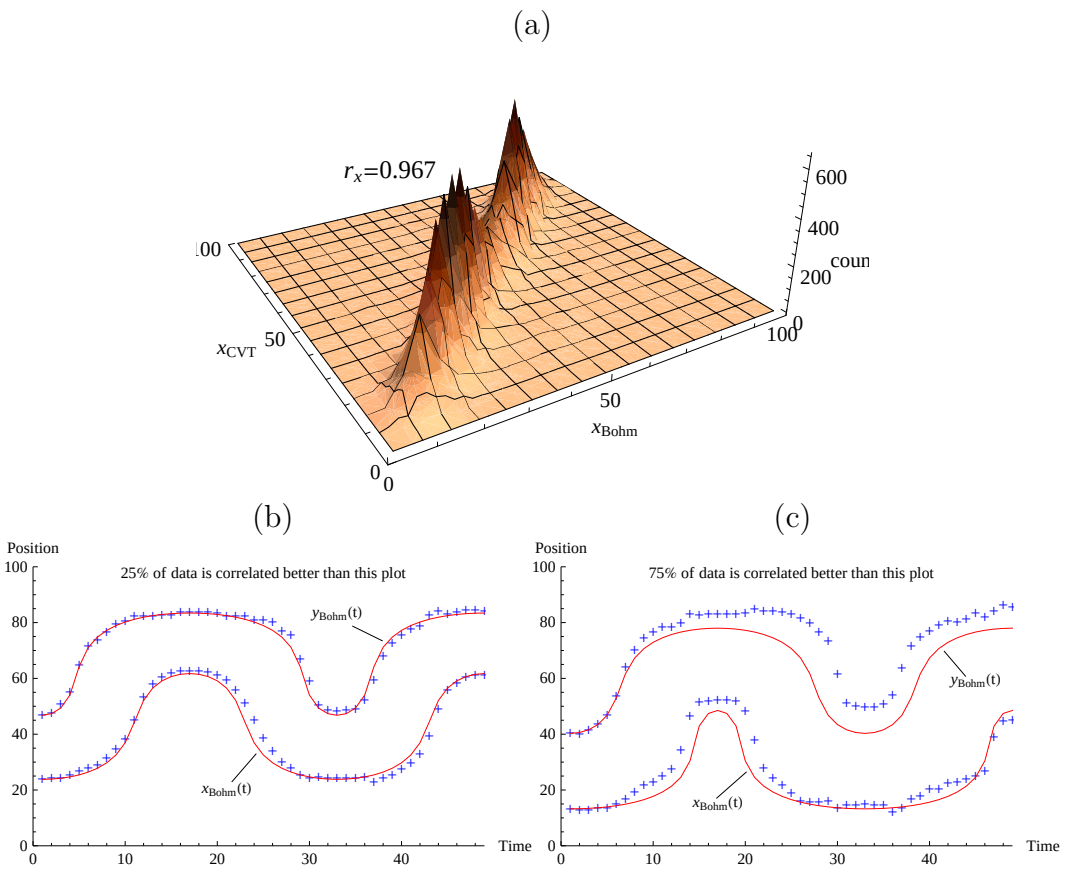


Figure 4.6: (color available) Results for the separable wave function in the two-dimensional well example. (a) The correlation of the whole ensemble's x positions for the CVT and Bohm trajectories. The correlation data was counted in bins of width 4 on each side. (b) Comparison of the CVT trajectory (+) and the Bohm trajectory (solid) for the worst correlation of the highest quartile (i.e. 25% of the data is better than this result), where the two-dimensional trajectory has been decomposed into its corresponding one-dimensional coordinates. (c) The best correlated result of the lowest quartile. All units have been naturalized ($\hbar = 1$).

the number of particles in the ensemble was $N = 400$. Time was restricted to $t \in [0, 10^4\pi]$ with 50 equal time steps. A fixed number of 400 iterations were performed of the Lloyd-Max algorithm for each time step. The non-separable wave function was,

$$\begin{aligned} \psi(x,y;t) &= \frac{\sqrt{2}}{L}\Bigg[\sin\left(\frac{\pi x}{L}\right)\sin\left(\frac{\pi y}{L}\right)e^{-iE_{11}t/\hbar} \\ &\quad + \sin\left(\frac{2\pi x}{L}\right)\sin\left(\frac{2\pi y}{L}\right)e^{-iE_{22}t/\hbar}\Bigg], \end{aligned} \tag{4.6}$$

where $E_{nm} = (n^2 + m^2)\pi^2\hbar^2/(2mL^2)$. The correlation coefficients between the CVT and Bohm trajectories' components were $r_x = 0.957$ and $r_y = 0.964$. In Figure 4.7(a) the x positions were again counted in 4×4 bins (and again, the y positions show a similar plot). Similar to the separable case above, the CVT trajectories for this non-separable example correlate well with the Bohm trajectories; in parts (b) and (c) again we show the worst and best results of the highest and lowest quartiles respectively.

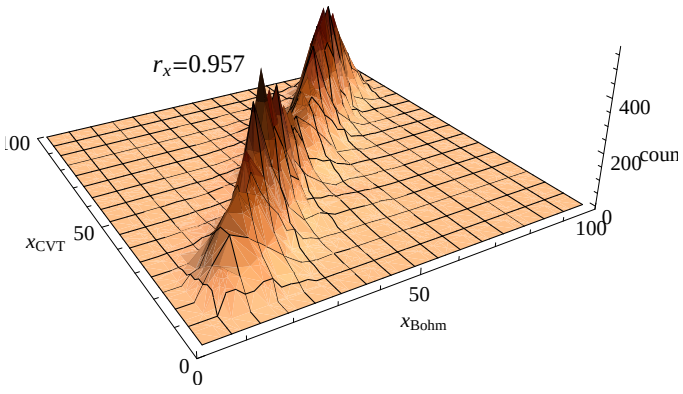
4.5 CVT Method and Quantum Nodes

It is difficult to achieve any desired accuracy with the CVT method in low probability regions since the algorithm encourages the particles to be in the higher probability regions. This behavior is especially true around zeros in the density (or wave function nodes). Here we present an example in which the wave function has six quasi-nodes (low probability regions), of which three periodically become actual nodes every quarter period. The wave function

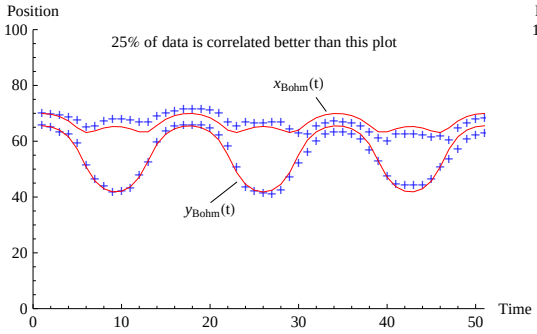
was,

$$\psi(x,y;t) = \sqrt{\frac{4}{3L^2}}\left[\sin\left(\frac{\pi x}{L}\right)\sin\left(\frac{\pi y}{L}\right)e^{-iE_{11}t/\hbar} + e^{-iE_{41}t/\hbar}\left\{\sin\left(\frac{4\pi x}{L}\right)\sin\left(\frac{\pi y}{L}\right) + i\sin\left(\frac{\pi x}{L}\right)\sin\left(\frac{4\pi y}{L}\right)\right\}\right], \tag{4.7}$$

(a)



(b)



(c)

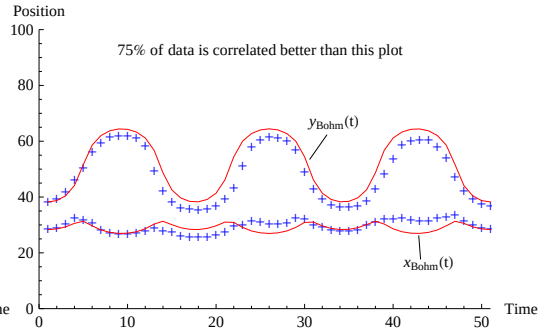


Figure 4.7: (color available) Results for the non-separable wave function in the two-dimensional well example. (a) The correlation of the whole ensemble's x positions for the CVT and Bohm trajectories. The correlation data was counted in bins of width 4 on each side. (b) Comparison of the CVT trajectory (+) and the Bohm trajectory (solid) for the worst correlation of the highest quartile, where 25% of the data is better than this plot. (c) The best correlated trajectory of the lowest quartile. Naturalized units with $\hbar = 1$ are used.

with $\hbar = 1$, $m = \pi^2/2$, $E_{nm} = (n^2 + m^2)\pi^2\hbar^2/(2mL^2)$, and the width on each side of the well was $L = 100$. The time for the calculation was $t \in [0, 8000\pi/3]$ (two periods) with 80 equal time steps, and the number of particles in the ensemble was $N = 1200$. The correlations between the CVT and Bohm trajectories were $r_x = 0.84$ and $r_y = 0.82$ with similar looking correlation plots as in Figure 4.6(a) and 4.7(a). The component correlations in this example are smaller that the examples above, since some particles during the CVT calculation were pushed to the opposite side of a nodal region than the corresponding Bohm trajectory. In Figure 4.8 five trajectories, each with a minimum x and y average correlation of 0.9, are shown for various time steps superposed on the corresponding CVT diagram. Again, the CVT trajectories (solid) resemble the Bohm trajectories (dashed) quite well even though the six nodal regions are moving quite a bit around the box. Between time steps 20 and 60 (and 40/80) the density evolves for one full period. Notice that after this full period none of the trajectories shown return to their original positions. Even after two periods (the total run of this calculation) the particles do not return to their initial launch points. This again demonstrates that the history or evolution of the fixed-point of the distortion functional is important for the CVT method to reproduce Bohm trajectories in these cases.

In this example, the trajectories do exhibit some helical behavior. The CVT trajectory method, however, might not able to produce perfect circulatory motion around prolonged or persistent *quantized vortices* [39, 72] that have non-zero vortex excitation, $\oint_L \nabla S \cdot d\mathbf{l} \neq 0$ for any closed loop L around a wave function node. For example, a stationary state for an electron in an hydrogen atom is $\psi_n(r, \phi, \theta, t) = e^{im\phi} f(r) e^{-iE_n t/\hbar}$. The behavior of the Bohm trajectories for the electron extensively depends on the value of the quantum

number m. When $m \neq 0$, the Bohm trajectories are circular orbits with constant angular speed centered around the persistent node at $r = 0$; for $m = 0$, however, the Bohm trajectories are at rest [12, 41]. The circular Bohm orbits for the $m \neq 0$ cases *do not* provide any additional information to the experimentally verifiable probability density ρ, and in fact, the orbits could have any orbital speed whatsoever and still make the same predictions. This contradictory behavior is not present with the CVT trajectories since for all values of m the CVT trajectories are at rest, which is a more consistent description.

4.6 Conclusion

The *centroidal Voronoi tessellation*(CVT) trajectory method uses no equations of motion for the particle trajectories themselves. Instead at each time a particle's position is determined by the global minimization of a distortional functional, Eq. 4.1. The method overcomes the lack of a natural ordering needed in the density sampling method to sort and identify the particles. This is achieved by recycling the minimum configuration of the old time as the starting configuration of the Lloyd-Max algorithm for the new time. Unlike the previous two methods, the resulting CVT trajectories match the Bohm trajectories even in the non-separable wave function examples.

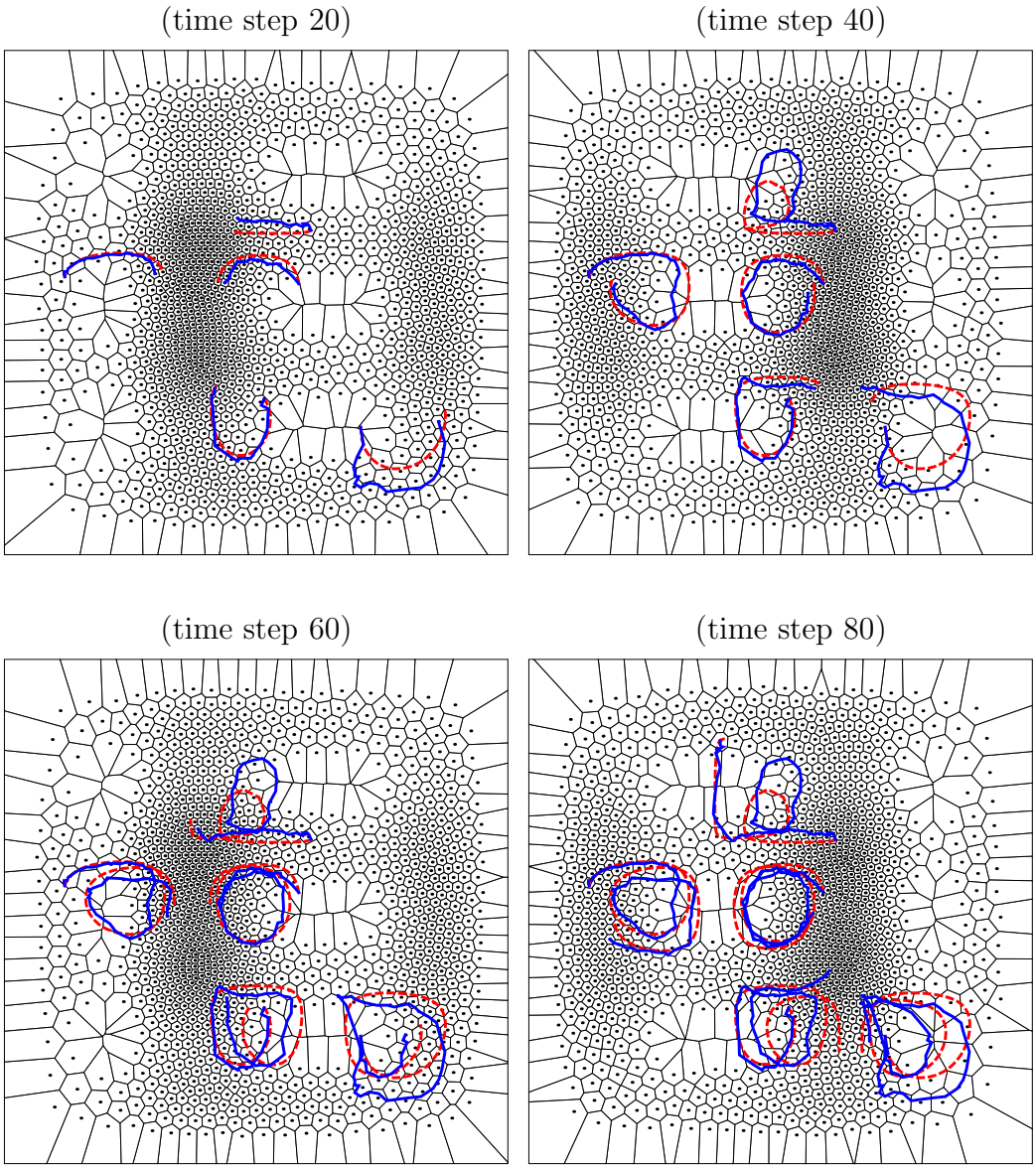


Figure 4.8: (color available)A complicated two-dimensional infinite square well that contains six quasi-nodes. Periodically three of the quasi-nodes become actual nodes every quarter period. The CVT trajectories (solid) and the Bohm trajectories (dashed) are shown for five particles superposed on the CVT diagram at various time steps. The five particles shown have at least an average x and y correlation of 0.9.

Chapter 5

Applications

To be consistent with the predictions of standard non-relativistic quantum mechanics a trajectory model need only satisfy three conditions:

1. The particles trajectories do not cross in configuration space.

2. The density of particle trajectories is equal to the probability density of quantum mechanics, $\psi^*\psi$.

3. The number of particle trajectories is constant due to the lack of creation and annihilation.

Numerous models can be created to satisfy these requirements. For one-dimensional systems, however, all methods will yield the same trajectories, which will be identical with the Bohm trajectories. In higher-dimensional problems this identity might not hold.

In addition to the requirements above, all three methods of Chapters 2, 3, and 4 were constructed so that 1) they didn't utilize any of Bohm's equations of motion, or any dynamical equations of motion, 2) the method's trajectories were identical to the Bohm trajectories, and 3) only used the quantum probability density $\rho = \psi^*\psi$ in their formulation. These constraints were further applied to each model with the aim of gleaning insight about the true nature of the Bohm trajectories. Beyond this understanding, the methods also have

practical applications in the measurement of the wave function and Planck's constant.

5.1 Interpretation of Bohm Trajectories

The Bohm trajectories by design never contradict the predictions of standard non-relativistic quantum mechanics. Bohm interprets the particle trajectories as *physically real*,

> The electron actually *is* a particle with a well-defined position $x(t)$ which varies continuously and is causally determined. [12]

Since the trajectories never contradict quantum mechanical experiments, and any measurement significantly alters the particle, the Bohm trajectories can never be experimentally verified one way or the other.

One challenge to the realist interpretation of the Bohm trajectories came from the under-determination of the quantum probability current (see §1.2.3.2). Holland [40] and others [66], however, working from the non-relativistic limit of the Dirac and Kemmer equations, showed that the Bohm guidance law was unique, though it might include a spin dependent term. Another challenge came by constructing situations for which the trajectories were argued to be surrealistic (see §1.2.3.3), but Hiley [38] countered that the surrealistic conclusions were based upon classical notions about the trajectories, and that one should not *a priori* judge the behavior of the Bohm trajectories; they simply do what they need to do, in order to satisfy the predictions of quantum mechanics.

The three non-dynamical methods described in Chapters 2–4 were designed to reproduce the Bohm trajectories without recourse to any of the

equations of motion of Bohm's theory. Instead, only the quantum probability density $\rho = \psi^*\psi$ was used to extract the quantum trajectories. In one dimension all three methods produced trajectories identical with Bohm's trajectories. Typically, the higher dimensional non-dynamical quantum trajectories remain identical to Bohm's trajectories with the noted exception around stationary persistent quantum nodes (see §4.5).

Beyond reproducing the Bohm trajectories for quantum probability densities, all three methods can compute trajectories for *any probability density!* In this respect the non-dynamical methods are more general that Bohm's theory. For a time-independent density the resulting non-dynamical trajectories are at rest, which is exactly the behavior of Bohm's trajectories as well. Since trajectories for classical objects can be experimentally verified the non-dynamical methods can be used to generate *Bohm-like* trajectories for classical probability densities. For example, consider the motion of a pendulum described by the angle of deflection, $\theta(t) = \Theta\cos(\omega t + \gamma)$. The time-independent probability distribution for the position of the pendulum is $\rho(\theta) = 1/(\pi\sqrt{1-(\theta/\Theta)^2})$ [20]. The resulting non-dynamical trajectories are at rest, which *does not* reflect the swinging motion of the pendulum. Again, the at rest trajectories for a time-independent probability density are a familiar behavior of Bohm's trajectories for stationary quantum states.

Now suppose that the pendulum is lightly damped but still oscillates with frequency ω, then the pendulum's motion is $\theta(t) = \Theta(t)\cos(\omega t + \gamma)$, where $\Theta(t)$ is a function that decays in time. The time-dependent probability distribution is $\rho(\theta;t) = 1/(\pi\sqrt{1-(\theta/\Theta(t))^2})$ [20]. The distribution is a concave-up bowl centered at zero with vertical asymptotes at $\pm\Theta(t)$, which becomes narrower as $\Theta(t)$ decays. In this case, the trajectories (see Figure 5.1)

begin bunched at the edges of the bowl, and then as time progresses move toward, but never cross, zero. As before, the trajectories *do not* resemble the swinging pendulum.

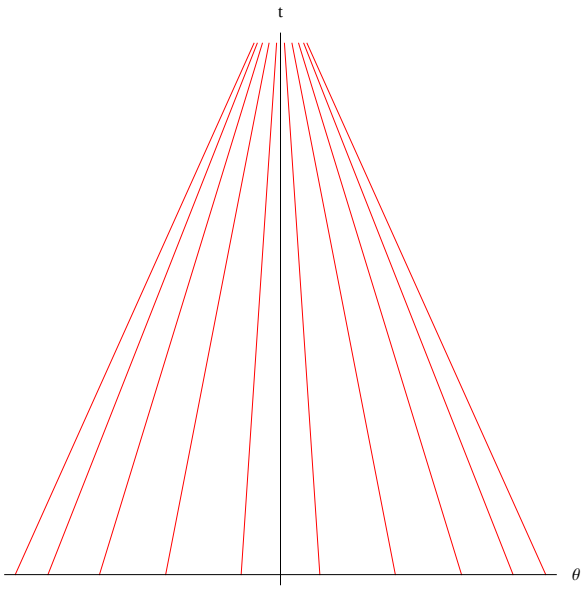


Figure 5.1: (color available). The resulting non-dynamical Bohm-like trajectories for a slightly damped classical pendulum. The trajectories *do not* reflect the actual swinging motion of the pendulum.

The CVT method produced Bohm trajectories for separable and non-separable wave functions. Recall the method generated the particle positions at each time by minimizing a distortion functional

$$D = \sum_{i=1}^{N} \int_{C_i} (\mathbf{x} - \mathbf{x}_i)^2 \rho(\mathbf{x})^\gamma \, d\mathbf{x}, \tag{5.1}$$

where $\gamma = (k+2)/k$, and k is the number of dimensions or length of the position vector $\mathbf{x}_i$. Of course the square measure is arbitrary and was only chosen so that the resulting trajectories would match the Bohm trajectories.

For classical systems the resulting Bohm-like trajectories *are not*, in general, the physically real trajectory. Instead, the non-dynamical methods produce hydrodynamic trajectories [16], in which the particles are simply discrete fluid elements that follow the probability density current. Therefore, the Bohm trajectories, like the non-dynamical trajectories, are *not physically real, but are just kinematically portraying the evolution of the probability density.*

5.2 Quantum Trajectories for Experiments

The connection between the non-dynamical methods described in the previous chapters and actual physical experiments is pretty straightforward. Recall that the quantum probability density $\rho(\mathbf{x}, t)$ is the only quantity that can be measured in quantum mechanics,

> ... in physics the only observations we must consider are position observations, if only the positions of instrument pointers. It is a great merit of the de Broglie-Bohm picture to force us to consider this fact. [8]

After the probability density has been measured at various times, the non-dynamical methods can be used to generate the quantum trajectories between those times. Then the quantum trajectories *permit one to infer* Schrödinger's wave function and Planck's constant.

To obtain the quantum trajectories for experiments follow these steps:

1. **Measure position data at various times.** At each time the position is measured for each particle of an ensemble of similarly prepared particles. A number of different times are necessary in order to build

trajectories during the time range. The examples below have on the order of ten experimental time steps. The position data can be recorded on a detection screen. Resolutions on the detection screen are around 10^{-6} meters. One such screen for detecting helium atoms was described by Kurtseifer and Mlynek in 1997 [44]. With this screen a two-slit experiment can be performed (see Figure 5.2) by moving the detection screen various distances from the slits. Each distance corresponds to a time since the motion from the slits to the screen is assumed uniform. At each distance a number of particle detections are recorded.

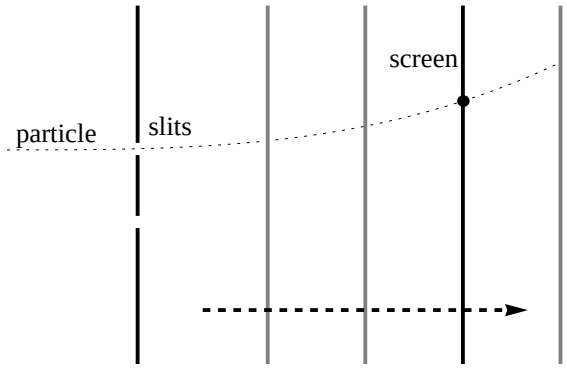


Figure 5.2: (color available). Experimental setup for a two-slit experiment using a movable detection screen. At each position the screen records a large number of particle detections.

2. **Estimate probability density at each time.** Using the measured position data, the probability density is estimated at each time. The field of density estimation is extensive, but some of the more popular techniques include: histograms, kernel estimators, or Fourier series estimation [67, 58]. The kernel estimators assume a kernel function $K(u)$ at

each data point X_j, and then adds the functions from the n data points,

$$\hat{\rho}(x) = \frac{1}{nh} \sum_{j=1}^{n} K\left(\frac{x - X_j}{h}\right), \tag{5.2}$$

where h is a bandwidth or smoothing parameter. Typical kernel functions include the Gaussian $K(u) = (1/\sqrt{2\pi}) \exp(-u^2/2)$, the triangular $K(u) = 1 - |u|$ for $|u| < 1$, and the Epanechnikov $K(u) = 3(1 - u^2/5)/(4\sqrt{5})$ with $|x| < \sqrt{5}$. Related to kernel estimation is the Fourier series estimator,

$$\hat{\rho}(\mathbf{x}) = \sum_{k} \hat{B}_k e^{2\pi i k \mathbf{x}}, \tag{5.3}$$

where the Fourier coefficients are defined,

$$\hat{B}_k = \frac{1}{n} \sum_{j=1}^{n} e^{-2\pi i k X_j}, \tag{5.4}$$

and $k = 0, \pm 1, \pm 2, \ldots$. The maximum k value of the estimator is determined by first testing if $|\hat{B}_k|^2 > 2/(n+1)$, and then the maximum k is set when typically one or two values in succession fail the inequality. The Fourier series estimator is frequently used due to its easy differentiation and integration properties.

3. **Generate Bohm trajectories from density.** Once the probability density has been estimated at each time the non-dynamical methods described in Chapters 2–4 can be used to approximate the Bohm trajectories between the experimental time range. The probability conservation method would be preferable if the measurements were in one dimension and the number of data points at each time is less than 10^5. If the number of data points at each time is greater than 10^5 then the density sampling technique might be preferred since the previous density estimation step

above can be omitted, and the data itself can be used to generate the sampling trajectories. For higher-dimensional measurements the CVT method needs to be used.

4. **Approximate smooth function for trajectories.** The velocity and acceleration along a each trajectory might need to be known so it is best to approximate each with a smooth function using the positions at the various time steps. Of course the approximation can be accomplished in many ways. In the first example below at each time the trajectory was approximated with a quadratic polynomial using least squares fitting with the twenty nearest neighbors. The second example has a closed form solution for the Bohm trajectories, so this solution's parameters were fit again with least squares across the entire ensemble.

5.2.1 Wave Function Measurements

Since Schrödinger first introduced his wave equation, people have been trying to determine the physical meaning of the wave function. Bohm interprets the wave function as physically real so that it can guide or pilot the particles. However, de Broglie, the initial creator of the pilot wave theory, claims that the wave function cannot be physically real since it propagates, in general, in a higher-dimensional configuration space. Recently, there have been experiments that attempt to actually measure the wave function (or the equivalent Wigner function) [9, 31, 32, 33, 44, 45, 59, 60]. These experiments must somehow mix the measurements of momentum and position together into a position only measurement. The non-dynamical quantum trajectory methods, however, only need position measurements.

To use the non-dynamical quantum trajectory methods to measure the

wave function, one first recalls that the *quantum trajectory method*(QTM) solves the time-dependent Schrödinger equation by simultaneously solving the following expressions [49, 72],

$$\frac{d\rho}{dt} = -\rho\nabla\cdot\mathbf{v} \tag{5.5}$$

$$\frac{dS}{dt} = L(t) = \frac{1}{2}m\mathbf{v}\cdot\mathbf{v} - (V+Q) \tag{5.6}$$

$$\frac{d\mathbf{x}}{dt} = \mathbf{v} = \frac{1}{m}\nabla S. \tag{5.7}$$

The wave function is computed along each trajectory in the ensemble by,

$$\psi(\mathbf{x},t) = \exp\left[-\frac{1}{2}\int_{t_0}^{t}\left(\nabla\cdot\mathbf{v}\right)_{\mathbf{x}(\tau)}\,d\tau\right]\exp\left[\frac{i}{\hbar}\int_{t_0}^{t}L(\tau)\,d\tau\right]\psi(\mathbf{x}_0,t_0). \tag{5.8}$$

The expressions above (and the Bohm equations) claim that the wave function phase S needs to be known before, or at the same time, to compute the particle trajectories. Given the complete density the expressions above cannot be solved for the phase and trajectories since there is no unique inverse to the divergence operator. But with only the density and any of the non-dynamical trajectory methods described in the previous chapters the Bohm trajectories can be generated *without the phase* S. Either way, once the trajectories have been computed, the phase is propagated along each trajectory across time steps Δt by

$$S_i = S_{i-1} + \Delta t\left(\frac{1}{2}mv_i^2 - V_i - Q_i\right), \tag{5.9}$$

where i is the i-th time step. The initial phase on all trajectories is assumed to be zero, since the wave function is only known within a global phase. During actual calculations the quantum potential,

$$Q = -\frac{\hbar^2}{2m}\frac{\nabla^2\sqrt{\rho}}{\sqrt{\rho}}, \tag{5.10}$$

is typically transformed by letting $C = \ln \sqrt{\rho}$ [72], then,

$$Q = -\frac{\hbar^2}{2m}\left(\nabla^2 C + \nabla C \cdot \nabla C\right). \tag{5.11}$$

In summary, to measure the wave function by the non-dynamical quantum trajectory methods, one first follows the procedure outlined in the previous section. Then after the quantum trajectories have been determined, the phase S of the wave function is computed along each trajectory using the expressions above. The amplitude of the wave function is calculated from the estimated probability density, or $R = \sqrt{\rho}$.

5.2.1.1 Two-Slit Example

A two-slit experiment with helium atoms to measure the Wigner function was first done in 1997 [45, 55]. Helium atoms, $m = 6.64632 \times 10^{-27}$ kg, were made to pass through two slits, each having width 10^{-6} m (see Figure 5.3), and a separation of 8×10^{-6} m. An atom detection screen [44] was placed at various distance from the slits. Since the motion from the slits to the screen is uniform it is treated like a time variable, and the quantum interference is only along the perpendicular direction. The detection screen's spatial resolution was 10^{-6} m, and it had a temporal resolution of 10^{-6} s. During the experiment a lensing system was used to project the momentum at different angles onto the detection screen.

The experiment was simulated (see Appendix D for program listings) by substituting the appropriate values into the two-slit wave function described in §5.1.2 of Holland [41]. The experimental data was simulated by sampling this *assumed* probability density 10^5 times in an interval of $\pm 5 \times 10^{-7}$ seconds centered at each time. The time range was $t \in [0, 975\mu s]$ with 10 equal time

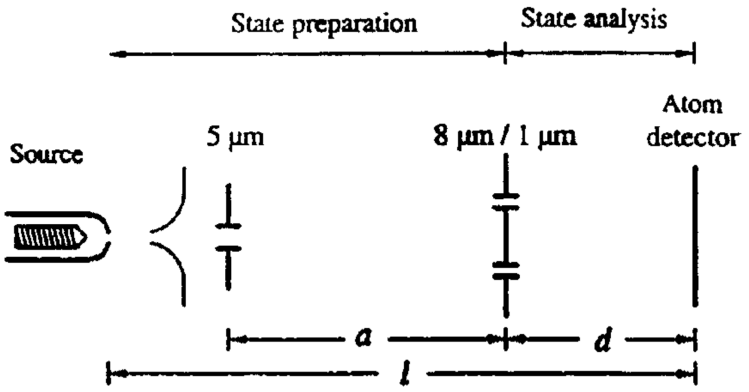


Figure 5.3: Experimental setup for a two-slit experiment performed with helium atoms. The detection screen is moved between various distances beyond the two slits. The figure is from Kurtsiefer *et al*, *Nature*, **386**, 150–153 (1997).

steps. The sampled points were counted in bins of width 10^{-6} meters along the detection screen. Only the bin counts centered at each bin on the screen were used for the rest of the calculation. The density was then estimated using a $k_{\max} = 40$ Fourier series estimator from Eq. 5.3. The quantum trajectories were computed using the probability conservation method of Chapter 2. The resulting trajectories were smoothed using a quadratic moving least squares fitting with the 20 nearest neighbors at each time. Between each of the experimental time steps an additional 10 computational steps were assumed. The density estimation at these in-between steps was a simple linear interpolation of the two densities surrounding the step. In Figures 5.4 and 5.5 is a comparison of the assumed probability density (the density used to generate the sampled data points) versus the *inferred* density (the density computed from the sampled data points) along the quantum trajectories. The two slits are located on the left side of the figures, while on the right side is seen the familiar bright and dark bands of intensity pattern on the screen. Along each

trajectory the average relative error between the inferred and assumed density was calculated. The errors are the least, see Figure 5.6 (the error bars in the figure represent the standard deviation of the relative errors for each trajectory), where the majority of trajectories are concentrated right behind the two slits. The assumed and inferred phase of the two-slit wave function is also compared in Figures 5.7 and 5.8, with the average relative errors in Figure 5.9. Similar to the density errors, the phase errors are also least where the quantum probability density is high right behind the slits.

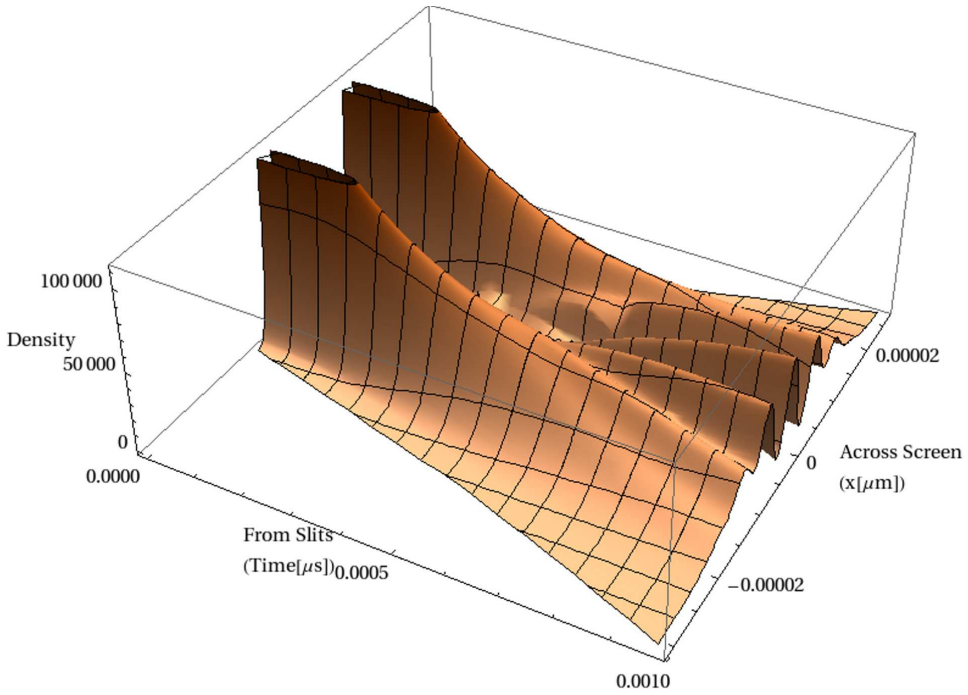


Figure 5.4: (color available). The assumed quantum probability density for the two-slit experiment that was the source of the simulated data used in the measuring the wave function helium example.

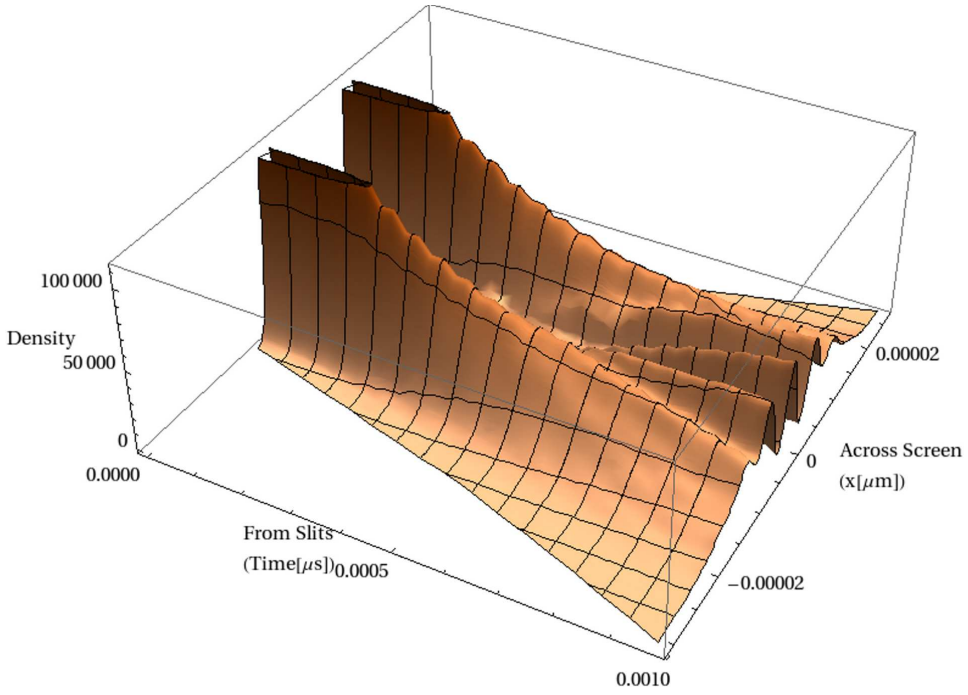


Figure 5.5: (color available). The inferred probability density computed from the simulated sampled data for the measuring the wave function helium example. The assumed or source density is in Figures 5.4.

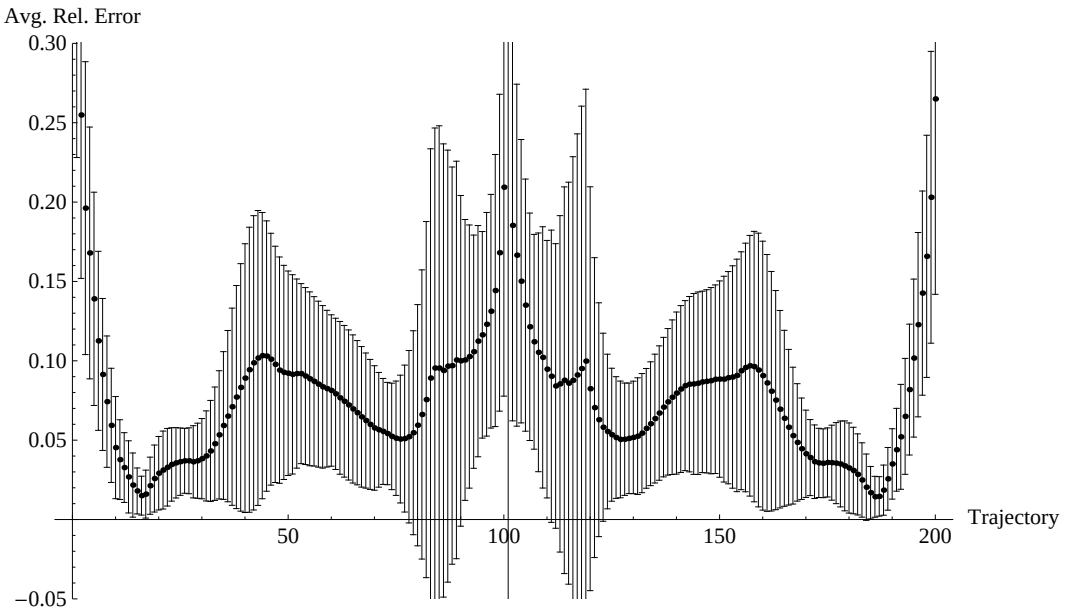


Figure 5.6: The average relative error along each trajectory between the assumed density (Figure 5.4), and the inferred density (Figure 5.5) for the helium two-slit experiment. The error bars represent the standard deviation of the relative errors for each trajectory. The errors are the least where the quantum probability density is high right behind the two slits.

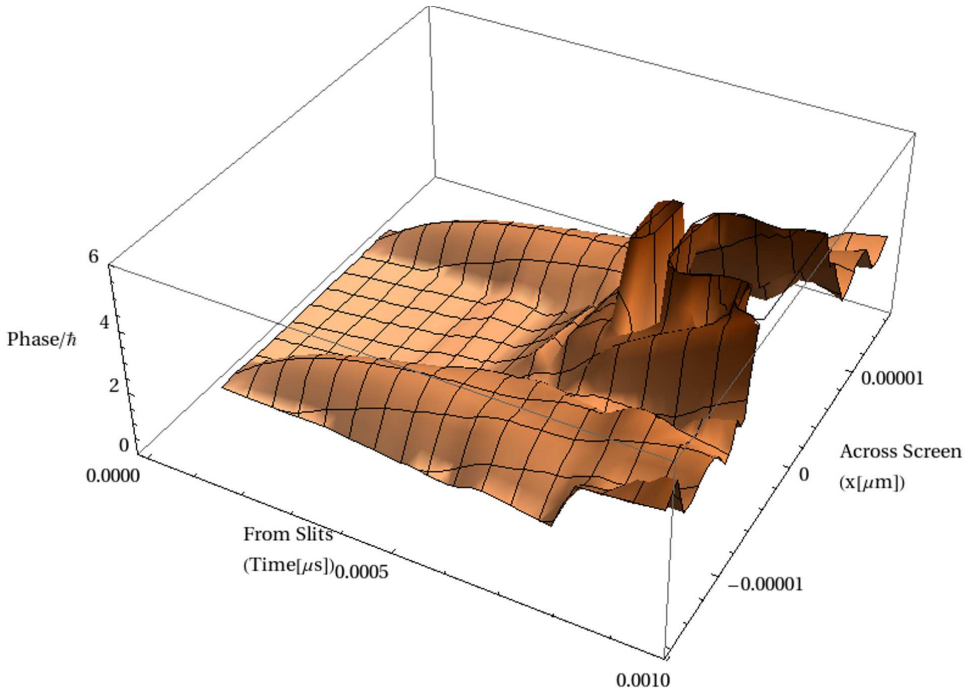


Figure 5.7: (color available). The phase of the source or assumed wave function supplied to the helium two-slit experiment simulation to generate experimental data.

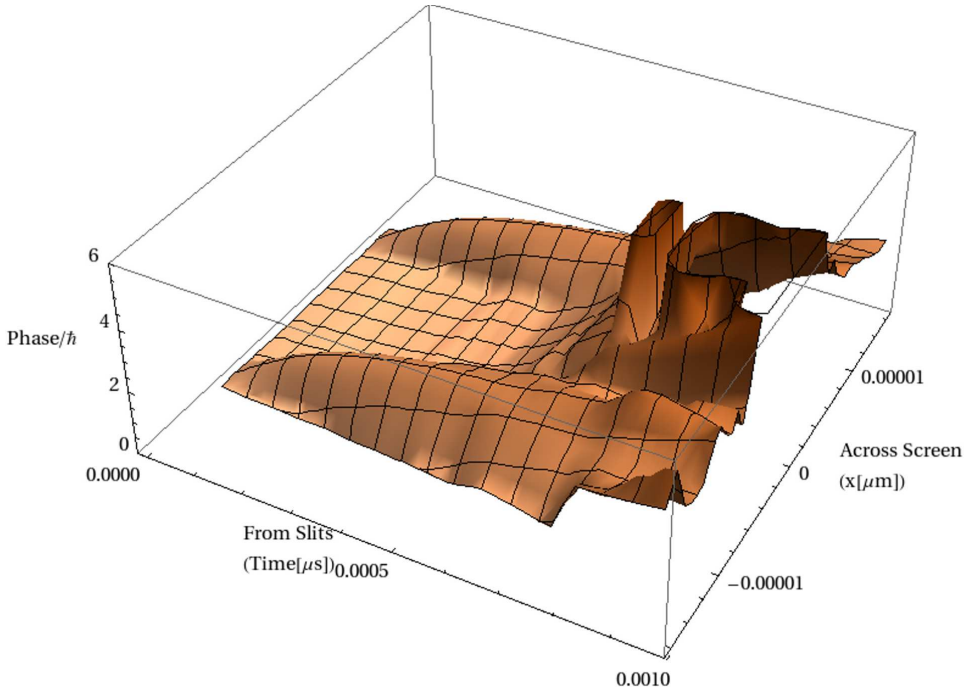


Figure 5.8: (color available). The measured or inferred phase of the wave function of the helium two-slit experiment simulation. The supplied or assumed phase of the wave function is shown in Figure 5.7.

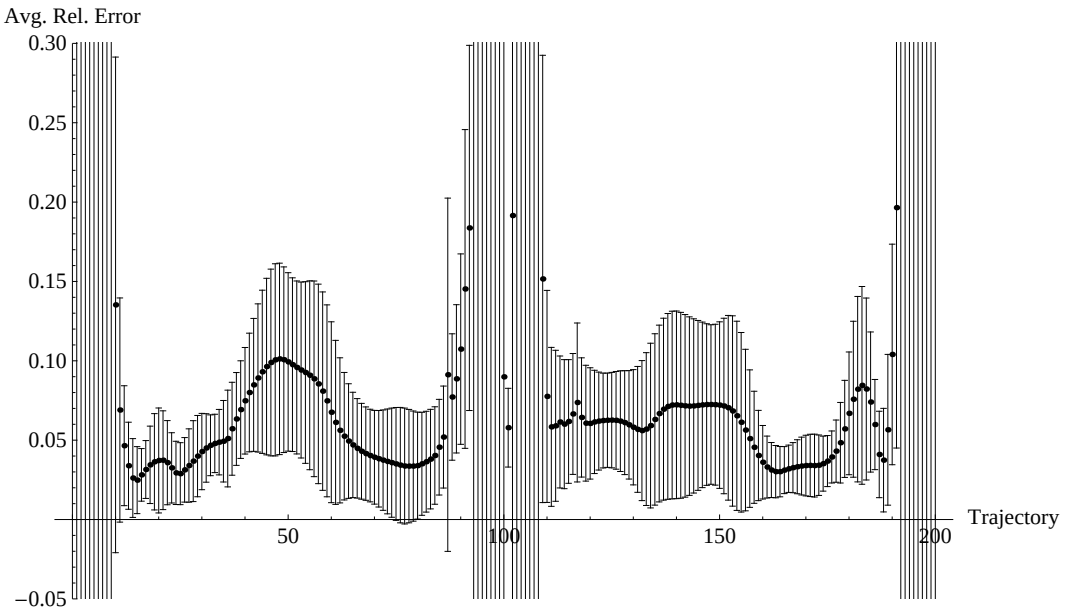


Figure 5.9: The average relative error along each trajectory of the assumed phase (Figure 5.7), and inferred phase (Figure 5.8) for the wave function of the helium two-slit experiment. The error bars are the size of the standard deviation of the relative errors for each trajectory. Again the errors are least for the trajectories right in the middle of the two slits.

5.2.2 Planck's Constant Measurements

Typically a watt balance is used for the measurement of Planck's constant [71, 65]. A watt balance uses an induced current, in a wire loop or coil that is placed in a magnetic field, to balance a solid mass object [26]. The watt balance experiments are complicated and require data be taken over month scales. The non-dynamical quantum trajectory methods might allow for the measurement to be done with a table-top experiment over day scales. Again the procedure above for obtaining quantum trajectories for experiments is preformed. The quantum Newton's second law expression (Eq. 1.9),

$$ma = -\nabla V + \frac{\hbar^2}{2m}\nabla\left(\frac{\nabla^2\sqrt{\rho}}{\sqrt{\rho}}\right), \tag{5.12}$$

can be solved for Planck's constant,

$$\hbar = \left[\frac{2m(ma+\nabla V)}{\nabla\left(\frac{\nabla^2\sqrt{\rho}}{\sqrt{\rho}}\right)}\right]^{1/2}. \tag{5.13}$$

Therefore, from just experimental position data taken at various times, the density ρ can be estimated, then from the density, the quantum trajectories are found by the non-dynamical methods, and then acceleration a along the trajectories is determined, which finally allows a value for Planck's constant to be measured.

5.2.2.1 Gaussian Single Slit Example

Here we simulate a single-slit electron diffraction experiment. Again, we assume position only data has been taken at various times as shown in Figure 5.2 for the two-slit experiment. The single slit was approximated by a Gaussian with width $\frac{1}{2}\sqrt{1+t^2}$, which has naturalized units of $\hbar = 1$ and mass

$m = 1$. The experimental times were between 0 and 3 with 10 equal steps. At each time the actual density was sample 14×10^6 times, with each sample counted in bins between ± 15 of width 0.015. An estimated Gaussian was determined by a least squares process [42] using a histogram of the counted bin data. Five-thousand quantum trajectories were computed by the probability conservation method of Chapter 2 using the estimated Gaussian density. The CPF constant values for the trajectories ranged uniformly from $[0, 0.25]$ and $[0.75, 1.0]$. The Bohm trajectories in this case have a closed form solution [41],

$$x(t) = x_0 \left[1 + \left(\frac{\hbar t}{2m\sigma_0^2} \right)^2 \right]^{1/2}, \tag{5.14}$$

where σ_0 is the initial half-width of the Gaussian slit, and x_0 is the initial position of the trajectory. Solving for Planck's constant gives,

$$\hbar = \frac{2m\sigma_0^2}{t} \sqrt{\frac{x(t)^2}{x_0^2} - 1}. \tag{5.15}$$

At each the 10 positions of the $N = 5000$ trajectories the measured value of $\hbar$ was computed. The estimated values and the histogram are shown in Figure 5.10. The average value of Planck's constant for all vertices was $\hbar = 1.0008 \pm 0.0011$. The non-dynamical quantum trajectories value of Planck's constant is not as precise as the watt balance experiments, but with additional data taken at each time step and improved computational techniques it potentially could be.

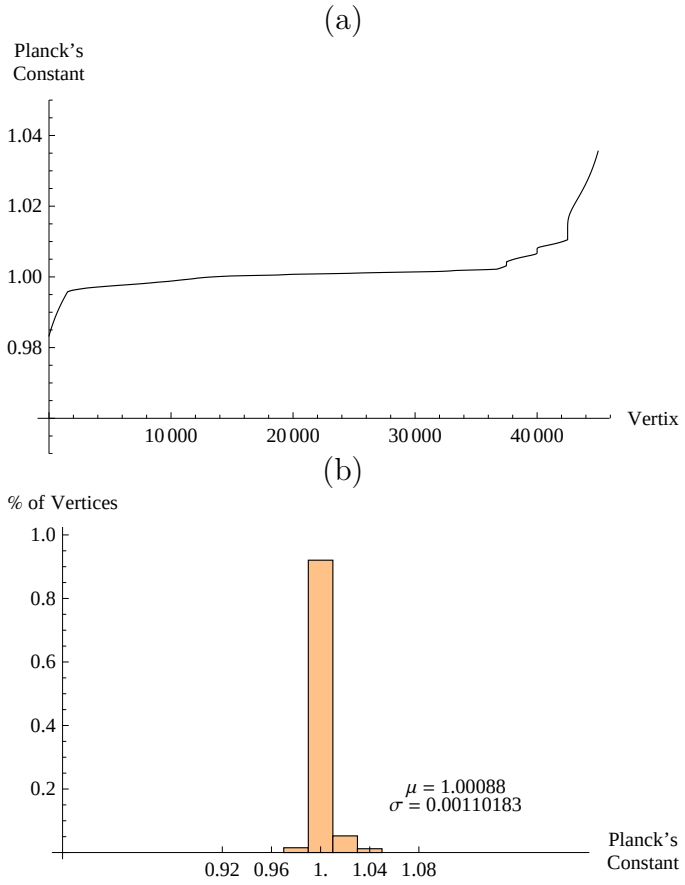


Figure 5.10: (color available). The experimental values for Planck's constant from a simulated Gaussian single-slit calculation with $\hbar = 1$ and $m = 1$. In (a) are the sorted values computed at each vertex on the quantum trajectories, and in (b) is a histogram of these values.

Chapter 6

Conclusion

In Chapters 2–4 three possible models for quantum trajectories were presented. Each of the *non-dynamical* models did not use or solve any equations of motion described by causes of motion: masses, forces, or potentials. In fact, only the probability conservation method in Chapter 2 took advantage of an equation to describe each particle's trajectory. This model proposed that trajectories evolve in such a way as to conserve probability to the left and right of them. In other words, for a particular particle's trajectory $x(t)$,

$$P_L(t) = \int_{-\infty}^{x(t)} \rho(x,t)\, dx = \text{constant}, \tag{6.1}$$

where the quantum probability density $\rho = \psi^*\psi$ for the Schrödinger wave function ψ. In Chapter 3 the next model was described that relied on sampling the quantum probability density. After the density was sampled at each time, the sampled points were numerically sorted, and then each particle's positions at the various times were chained together to form trajectories. Both of these first two methods had the drawback that they only worked in one dimension, or higher dimensions for separable wave functions. The last model in Chapter 4 overcame this deficiency and still utilized no equations of motion at all. It was realized that the goal of sampling the probability density was to find a finite representation of the density. A novel distortion functional was defined,

$$D = \sum_{i=1}^{N} \int_{C_i} (\mathbf{x} - \mathbf{x}_i)^2 \rho(\mathbf{x})^\gamma \, d\mathbf{x}, \tag{6.2}$$

where $\gamma = (k+2)/k$, and k is the number of dimensions (the length of each position vector $\mathbf{x}_i$). The best representation at each time was defined as the particle configuration that minimized the distortion. This configuration was found to be a *centroidal Voronoi tessellation*(CVT), which was computed by an altered Lloyd-Max iterative algorithm; the best representation was recycled from the previous time to begin the search for the new representation at the current time. The CVT trajectories were then similarly constructed by joining the CVT positions of each particle from each time step. Unlike the previous models, the CVT method works in any number of dimensions for *any* density, but is computationally intensive (and not well studied) for dimensions greater than two.

In many situations, all three methods were able to reproduce the known quantum trajectories of Bohm's particle theory. The probability conservation and density sampling methods worked in one-dimension and for higher dimensional separable wave functions. The CVT method was shown to work in one or two dimensions for separable and non-separable wave functions in several cases. The only known case where the CVT and Bohm trajectories disagreed was around a persistent stationary node of the wave function. These methods together, though, provide a new insight into the true nature of the Bohm trajectories. In was argued that the Bohm particle trajectories, instead of being physically real, are *simply kinematically portraying the evolution of the quantum probability density* ρ. In addition, it was shown that the non-dynamical quantum trajectory methods allow one to measure or infer Schrödinger's wave function (amplitude and phase) and Planck's constant from *only* experimental position data taken at various times.

6.1 Future Work

The non-dynamical quantum trajectory methods presented herein are ripe for improvements and new applications. All three methods can be easily converted to run on parallel computers. Perhaps with more computing power the density sampling method could be used for non-separable wave functions in higher dimensions by computing all possible mappings from one time step to the other, and from this large set (on the order of N^2) choose the mapping that satisfies some additional constraint, like least maximum distance moved. Beyond the nodal examples shown in §4.5, the Voronoi method needs to be altered to accommodate stationary persistent wave function nodes, so that the non-dynamical quantum trajectories again reproduce Bohm's trajectories in this highly constrained situation.

The use of the non-dynamical quantum trajectories for measurements of the wave function and Planck's constant could also be improved upon. New numerical techniques will be needed to increase the precision of the Planck's constant experiment to become comparable the other standard experiments. There are other interesting avenues for new uses of the methods as well. As remarked in §5.1 the methods can also be used with classical probability densities. In fact, the methods can generate Bohm-like trajectories for *any* density, or even any positive function for that matter. From the density of some system, the kinematic Bohm-like trajectories are computed using the non-dynamical methods, then the trajectories themselves can be used to infer the dynamical laws for a particle description of the system.

Appendices

Appendix A

Program Listing: Probability Conservation Trajectories for the Infinite Square Well

Below is the *Mathematica* code to generate the probability conserved trajectories and the Bohm trajectories for an infinite square well where the wave function is a superposition of two energy eigenstates.

Declaration of parameters and variables.

```
m = π^2/2;                          (* mass of particle *)
h = 1.0;                            (* Plank's const divided by 2π *)
L = 1.0;                            (* width of the well *)
n1 = 1.0; n2 = 2.0;                 (* energies of wave function *)

particles = 6;                      (* number of particle trajs *)
tmin = 0.0; tmax = 3.0;             (* time range *)
deltaT = tmax/30;                   (* time step *)
xmin = 0.0; xmax = L;               (* left/right well boundaries *)
bins = 50;                          (* number of bins in well *)
deltaX = (xmax - xmin)/bins;        (* bin width *)

(* set up x values for the bins *)
binX = Table[N[xp], {xp, xmin ,xmax , deltaX}];

(* wave function is superposition of two energies *)
psi = Sqrt[1/L] (Sin[n1 π x/L] Exp[-I n1^2 π^2 h^2 t/(2 m L^2 h)] + Sin[n2 π x/L] Exp[-I n2^2 π^2 h^2 t/(2 m L^2 h)])
```

```
(* conjugate of wave function *)
psistar = psi /.Complex[aaa_, bbb_]→ −Complex[aaa, bbb];

(* probability density *)
rho = psistar * psi;
```

Function Definitions

```
(*
 getX[{xvalues}, {rvalues}, r]
 Returns the x value that corresponds to the value r.
 The list rvalues contains the CPF values for the bins
 in addition to 0.0 and 1.0
*)
getX[xvalues_, rvalues_, r_]:=
 Block[{s,rtemp=rvalues},
    rtemp = Sort[AppendTo[rtemp, r]];
    s = First[Position[rtemp, r]-1][[1]];
    xvalues[[s]] +
    (r - rvalues[[s]]) *
    (xvalues[[s+1]]-xvalues[[s]]) / (rvalues[[s+1]]-rvalues[[s]])
 ];


(*
 cpfRho[xp, tp]
 Returns the integration of the density rho at a given
 time from the lower bound xmin to the location xp.
 Better known as the Cumulative Probability Function.
 Assumes a function rho(x,t) and xmin are defined already.
*)
cpfRho = Compile[{{xp,_Real}, {tp, _Real}},
    NIntegrate[rho/.t→tp,{x, xmin, xp}]
];
```

Set up Initial Trajectory Values

```
(* equally distribute particle launch points in the well *)
launchPoints = Take[
   Table[N[xp], {xp, xmin, xmax, (xmax-xmin)/(particles+1)}],
 {2, particles + 1}];

(* the r-value (or CPF value) of each launch point *)
particleRs = Table[ cpfRho[launchPoints[[i]], 0], {i,particles}];
```

Calculate the Bohm Trajectories

```
(* the Bohm velocity field *)
v = (h/(2 m I) * ((psistar * D[psi, x] - psi * D[psistar, x])/rho)) /.x→x[t];

(* the Bohm trajectory for each launch point *)
BohmTrajectories = Table[
 x /.  First[
   NDSolve[{x'[t] == v, x[0] == launchPoints[[i]]},
   x, {t, tmin, tmax}]
   ],
 {i, particles}];


(* plot of Bohm trajectories *)
BohmPlot = Table[
 Plot[ BohmTrajectories[[i]][t], {t, tmin, tmax},
   DisplayFunction→Identity,
   PlotRange→{xmin, xmax},
   AxesOrigin→{tmin, xmin},
   AxesLabel→{``t'', ``x(t)''},
   Ticks→{Range[tmin, tmax, (tmax-tmin)/6],
              Range[xmin, xmax, (xmax-xmin)/4]},
   PlotStyle→{Thin, Black}],
 {i,particles}];
```

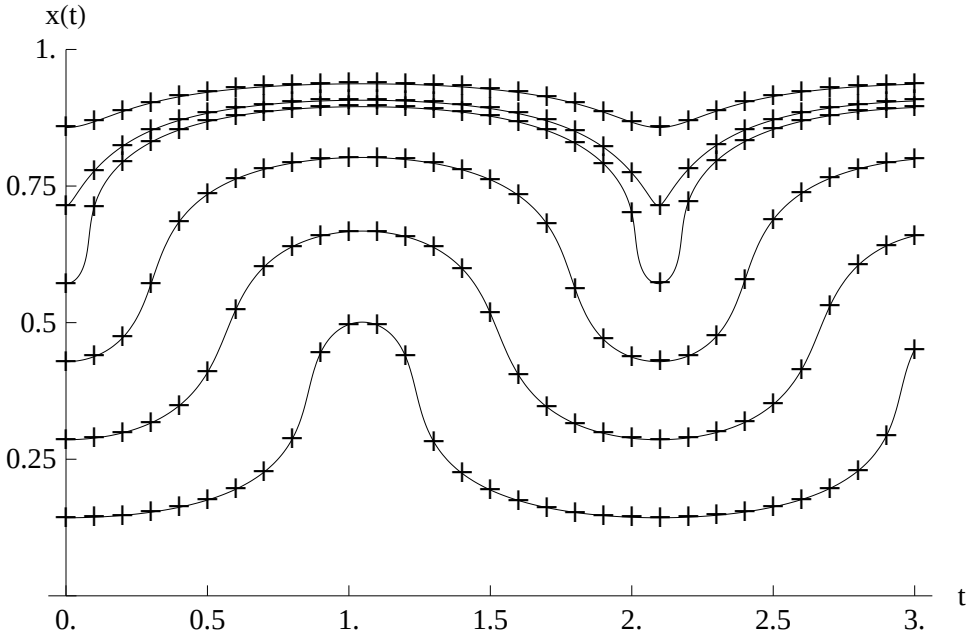
Calculate the Probability Conservation Trajectories

```
(*
 Loop through time steps to build trajectories.  At each
 step the r-values (or CPF values) of the bins are
 calculated.  Then for each particle the new position is
 found, and added to the particle's trajectory
*)
CPFTrajectories = Table[{{0, launchPoints[[i]]}}, {i,particles}];
tcurrent = deltaT;
While[ tcurrent <= tmax,
 binR = Table[ cpfRho[binX[[i]], tcurrent], {i, bins+1}];
 For[j=1, j <= particles, j++,
   xpoint = getX[ binX, binR, particleRs[[j]] ];
   AppendTo[CPFTrajectories[[j]],{tcurrent, xpoint}]
 ];
 tcurrent = tcurrent + deltaT;
]


(* plot of CPF trajectories *)
CPFPlot = Table[
 ListPlot[ CPFTrajectories[[i]],
   Joined→False,
   DisplayFunction→Identity,
   PlotRange→{xmin, xmax},
   AxesOrigin→{tmin, xmin},
   PlotStyle→{Black},
   PlotMarkers→{``+'', 12}],
 {i,particles}];
```

Show CPF vs Bohm Trajectories

```
Show[ BohmPlot, CPFPlot, ImageSize→Large ]
```



Appendix B

Program Listing: Density Sampling Trajectories for Harmonic Oscillator

Below is the *Mathematica* code to generate the density sampling trajectories and the Bohm trajectories for a wave function that is a superposition of the ground and first excited states of a harmonic oscillator.

Declaration of parameters and variables.

```
ensemble = 10^4;                (* points to sample each time *)
particles = 5;                  (* trajectories to compute *)
tmin = 0;                       (* time values *)
tmax = 3;
deltaT = tmax/60;
xmin = -5;                      (* boundaries *)
xmax = +5;


h = 1;                          (* Planck's constant *)
ω = 3;                          (* frequency *)
m = 1;                          (* particle's mass *)
a = (h/(mω))^(1/2);
n1 = 0; n2 = 1;                 (* ground + first excited states *)
```

```
(* wave function is superposition of two energies *)
```

$$\texttt{psi} = \frac{1}{\sqrt{2}}\left(\frac{\texttt{HermiteH[n1, x/a] Exp[-x}^2\texttt{/(2 a}^2\texttt{)]}}{\left(\texttt{n1!}\quad 2^{\texttt{n1}}\ \texttt{a}\ \sqrt{\pi}\right)^{1/2}}\texttt{Exp}\left[-\texttt{I}\left(\texttt{n1}+\tfrac{1}{2}\right)\texttt{h}\omega\frac{\texttt{t}}{\texttt{h}}\right]+ \frac{\texttt{HermiteH[n2, x/a] Exp[-x}^2\texttt{/(2 a}^2\texttt{)]}}{\left(\texttt{n2!}\quad 2^{\texttt{n2}}\ \texttt{a}\ \sqrt{\pi}\right)^{1/2}}\texttt{Exp}\left[-\texttt{I}\left(\texttt{n2}+\tfrac{1}{2}\right)\texttt{h}\omega\frac{\texttt{t}}{\texttt{h}}\right]\right)$$

```
(* conjugate of wave function *)
psistar = psi /.Complex[aaa_, bbb_]→ −Complex[aaa, bbb];
(* probability density *)
rho = psistar * psi;
```

Function Definitions

```
(*
 rhoC[ x, t]
 A compiled function for the density rho.
*)
rhoC = Compile[ {x, t}, Evaluate[Re[rho]] ];


(*
 getMaximumValue[ number, xMin, xMax, tNow ]
 Evaluates density rho number times between xMin and xMax
 at tNow, and returns the maximum value found.
*)
getMaximumValue[ number_, xMin_, xMax_, tNow_ ] :=
   Last[Sort[Table[rhoC[RandomReal[xMin,xMax],tNow],number]]]
```

```
(*
 getAPoint[ xMin, xMax, tNow, rhoMax ]
 Returns one sampled point between xMin and xMax
 for the density with a maximum of rhoMax at time tNow.
*)
getAPoint = Compile[#1, #2, #3, #4,
 Block[ xmin = #1, xmax = #2, rhotry, xtry=0.0,
    While[ True,
    rhotry = RandomReal[#4];
    xtry = RandomReal[xmin,xmax];
    If[ rhoC[xtry, #3] >= rhotry, Break[] ]
    ];
 xtry] ];


(*
 getPoints[ numbGet, xMin, xMax, tNow, rhoMax ]
 Returns numbGet sampled points between xMin and xMax
 for the density with a maximum of rhoMax at time tNow.
*)
getPoints[ numbGet_, xMin_, xMax_, tNow_, rhoMax_ ] :=
    Table[ getAPoint[xMin,xMax,tNow,rhoMax], numbGet ];
```

Set up Initial Trajectory Values

```
(* which particles going to make trajectories for *)
trackingParticles =
    Table[ i*Round[ensemble/(particles+1)], {i, 1, particles} ];

rhoMax = getMaximumValue[10^3, xmin, xmax, 0] * 1.1;

possibleLaunchPoints =
    Sort[ getPoints[ensemble, xmin, xmax, 0, rhoMax] ];
```

```
(* launch points for trajectories *)
launchPoints =
   Table[ possibleLaunchPoints[[trackingParticles[[i]] ]],
   {i, 1, particles} ];
```

Calculate the Bohm Trajectories

```
(* the Bohm velocity field *)
v = (h/(2 m I) * ((psistar * D[psi, x] - psi * D[psistar, x])/rho)) /.x→x[t]);

(* the Bohm trajectory for each launch point *)
BohmTrajectories = Table[
 x /.  First[
   NDSolve[{x'[t] == v, x[0] == launchPoints[[i]]},
   x, {t, tmin, tmax}]
   ],
 {i, particles}];


(* plot of Bohm trajectories *)
BohmPlot = Table[
 Plot[ BohmTrajectories[[i]][t], {t, tmin, tmax},
   DisplayFunction→Identity,
   PlotRange→{-1, +1},
   AxesOrigin→{0, -1},
   AxesLabel→{``t'', ``x(t)''},
   Ticks→{{0.5, 1, 1.5, 2, 2.5, 3},
            {-1, -0.5, 0, 0.5, 1}},
   PlotStyle→{Thin, Black}],
 {i,particles}];
```

Calculate the Density Sampling Trajectories

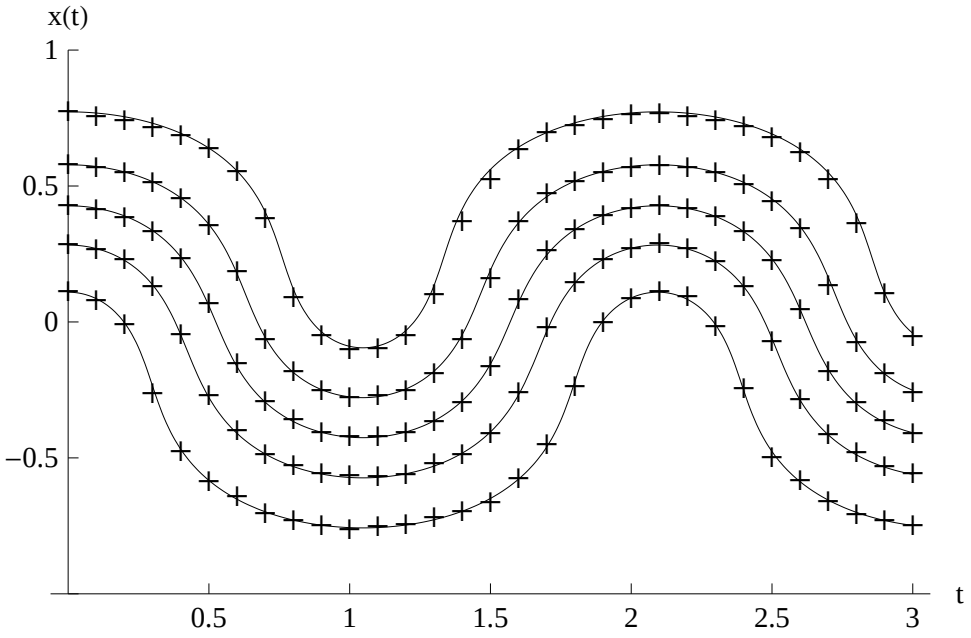
```
(*
 Loop through time steps to build trajectories.  At each
 step the density is sampled ensemble times.  The sampled
 points are sorted and the positions are added to each
 trajectory.
*)
RandomTrajectories =
   Table[{{0, launchPoints[[i]]}}, {i, 1, particles}];
tcurrent = deltaT;
While[ tcurrent <= tmax,
 rhoMax = 1.1 * getMaximumValue[10^3, xmin, xmax, tcurrent];
 xpoints = Sort[getPoints[ensemble, xmin, xmax, tcurrent, rhoMax]];
 For[j=1, j <= particles, j++,
   AppendTo[RandomTrajectories[[j]],
               {tcurrent, xpoints[[trackingParticles[[j]]]]}]
 ]; tcurrent = tcurrent + deltaT;
];


(* only plot every other time step *)
RandomTrajectoriesShort = Table[ RandomTrajectories[[j,i]],
   {j, 1, particles}, {i, 1, Length[RandomTrajectories[[1]]],2} ];


(* plot of density sampled trajectories *)
RandomPlot = Table[
 ListPlot[ RandomTrajectoriesShort[[i]],
   Joined→False,
   DisplayFunction→Identity,
   PlotRange→{-1, +1},
   AxesOrigin→{0, -1},
   PlotStyle→{Black},
   PlotMarkers→{``+'', 12}],
 {i,particles}];
```

Show Density Sampled vs Bohm Trajectories

```
Show[ BohmPlot, RandomPlot, ImageSize→Large ]
```



Appendix C

Program Listing: Centroidal Voronoi Tessellation Trajectories for Infinite Square Well (2D)

The C code for a non-separable wave function in a two-dimensional infinite square well using the centroidal Voronoi tessellation (CVT) trajectory method of Chapter 4. The code is divided into six files: defs.h, main.c, lloyd.c, triangle.c, rho.c, output.c. The Voronoi tessellations are performed by Fortune's program [30] utilizing Derek Bradley's memory fixes contained online at `http://www.derekbradley.ca/voronoi.html`. Fortune's `out_triple` function was replaced by the routine in output.c.

defs.h

```
/*************
* constants *
*************/
#define GAUSSPTS 25

#define ENSEMBLE        400        /* particle # in box */

/* % to reflect at boundaries
   and room for the reflections */
#define BOX_REFLECTION  1.0
#define ENSEMBLE_PLUS   6*ENSEMBLE

#define MAX_LLOYD       500        /* # of iterations */
#define RHO_MAX         6e-2       /* max(rho(t=0)^2) */
```

```
#define TMIN            0.0       /* time settings */
#define TMAX            10.0
#define TSTEPS          10
#define DELTA_T         ((TMAX - TMIN) / TSTEPS)

#define BOX_XMIN        0.0       /* box dimensions */
#define BOX_YMIN        0.0
#define BOX_XMAX        100.0
#define BOX_YMAX        100.0

/* max # vertices per generator's polygon */
#define MAX_NUM_VERTICES 30

#define SRAND_SEED      4266      /* [0, 65535]*/


/*************
* structures *
*************/
struct Point
{
  double x,y;
};

struct Generator
{
  struct Point coord;
  struct Point cell[MAX_NUM_VERTICES];
  int numcell;      /* vertices count*/
  int genid;        /* >0 if inside the box, 0 otherwise */
} generator[ENSEMBLE_PLUS], generatorTemp[ENSEMBLE];

struct myTriangle   /* holds Fortune's output */
{
  int gen1, gen2, gen3;
  struct Point center;
};
```

```
struct vertix_angle /* angle of each vertix */
{
  double angle;
  int vertixID;
};

/************************
* function declarations *
************************/
void compute_voronoi(void);     /* Fortune's main */
void new_generators(void);
void ReflectPoints(void);
void GetPolygons(void);
void sort_vertices(struct Generator *g);
int generator_comp(const void *a, const void *b);
int angle_comp(const void *a, const void *b);
double what_angle(struct Point *p1, struct Point *p2);
void center_generators(void);
void center_generators_uniform(void);
void TriangleCenter(struct Point *p1, struct Point *p2,
                    struct Point *p3,
                    struct Point *result );

double triangle_center( struct Point *p1,
                        struct Point *p2,
                        struct Point *p3,
                        struct Point *result);

double triangle_error(  struct Point *p1,
                        struct Point *p2,
                        struct Point *p3 );

double rho(struct Point *p);

struct myTriangle theTriangle[4*ENSEMBLE_PLUS];
int  myNumTriangles;
double tnow;          /* time now */
```

main.c

```
/*********************************************************
*
* Compute Centroidal Voronoi Tessellation Trajectories for
* a Non-Separable Probability Distribution in a Two-
* Dimensional Infinite Square Well.
*
* Tim Coffey, 2009
*
*********************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "defs.h"


/*********************************************************
*  Main Function
*********************************************************/
int main(void)
{
  int i, j, k, t;
  int numTemp;

  FILE *data_file;

  /* limit filenames to 20 characters */
  char data_name[20] = "time_step_";
  char buf[20];

  /* set time, and get new generators */
  tnow = TMIN;
  new_generators();
```

```
/* loop time and include last step */
for(t=0; t<(TSTEPS + 1); t++)
{
  tnow = TMIN + t * DELTA_T;

  /* Lloyd iterations for this time step */
  for(k=0; k<MAX_LLOYD; k++)
  {
    myNumTriangles = 0;       /* (re)set this counter */

    nsites = ENSEMBLE;        /* Fortune needs site # */

    /* set the box dimensions */
    xmin = BOX_XMIN;  ymin = BOX_YMIN;
    xmax = BOX_XMAX;  ymax = BOX_YMAX;

    /* create boundaries of box by reflection */
    ReflectPoints();

    /* Fortune requires sorting */
    qsort(generator, nsites, sizeof *generator, scomp);

    compute_voronoi();  /* Fortune's main() */
    free_all();         /* Bradley's recover memory */

    GetPolygons();      /* each generator's polygon */

    /* transfer those in the box to temp. storage */
    numTemp = 0;
    for(i=0; i<nsites; i++)
    {
      if( generator[i].genid )     /* >0 inside box */
      {
        generatorTemp[numTemp] = generator[i];
        numTemp++;
      }
    }
```

```
  /* transfer back those in the box */
  for(i=0; i<ENSEMBLE; i++)
  {
    generator[i] = generatorTemp[i];
  }

  /* move each to center of mass of polygon
     for all steps except last one */
  if( k<(MAX_LLOYD-1) )
    center_generators();

} /* k (Lloyd iterations) */

/* sort generators to maintain identity
   and build trajectories */
qsort( generator, ENSEMBLE,
       sizeof *generator, generator_comp );

/* save generators' positions */
sprintf( buf, "%d", t );
strcat( buf, ".txt" );
strcat( data_name, buf );

data_file = fopen( data_name, "w" );
if( data_file == NULL)
{
  printf("Couldn't␣open␣file.\n");
  return -1;
}

for(i = 0; i<ENSEMBLE; i++)
{
  fprintf(data_file, "%f␣%f␣",
      generator[i].coord.x, generator[i].coord.y );
}
fprintf(data_file,"\n");
```

```
    /* save polygons */
    for(j = 0; j<ENSEMBLE; j++)
    {
      for(i = 0; i<generator[j].numcell; i++)
      {
        fprintf(data_file, "%f␣%f␣",
          generator[j].cell[i].x,
          generator[j].cell[i].y );
      }
      fprintf(data_file, "\n");
    }

    /* clean up and reset */
    fclose( data_file );
    strcpy( data_name, "time_step_" );

  } /* t (time steps) */

  return 0;
} /* end main */



/*********************************************************
*  ReflectPoints()
*
* Reflects a % of the ENSEMBLE points about each box side
* in order to create an artificial boundary
*********************************************************/
void ReflectPoints( void )
{
  int i;

  /* compute BOX_REFLECTION markers */
  double xLower = BOX_XMIN +
                  (BOX_XMAX - BOX_XMIN) * BOX_REFLECTION;

  double yLower = BOX_YMIN +
                  (BOX_YMAX - BOX_YMIN) * BOX_REFLECTION;
```

```
double xUpper = BOX_XMAX -
                (BOX_XMAX - BOX_XMIN) * BOX_REFLECTION;

double yUpper = BOX_YMAX -
                (BOX_YMAX - BOX_YMIN) * BOX_REFLECTION;

/* assumes nsites is already equal to ENSEMBLE */
for(i=0; i<ENSEMBLE; i++)
{

  /* reflect around the x=BOX_XMIN line */
  if( generator[i].coord.x <= xLower )
  {
    generator[nsites].coord.x =
              2.0 * BOX_XMIN - generator[i].coord.x;
    generator[nsites].coord.y = generator[i].coord.y;
    generator[nsites].numcell = 0;
    generator[nsites].genid = 0;      /* outside box */
    nsites++;
  }

  /* reflect around the x=BOX_XMAX line */
  if( generator[i].coord.x >= xUpper )
  {
    generator[nsites].coord.x =
              2.0 * BOX_XMAX - generator[i].coord.x;
    generator[nsites].coord.y = generator[i].coord.y;
    generator[nsites].numcell = 0;
    generator[nsites].genid = 0;      /* outside box */
    nsites++;
  }

  /* reflect around the y=BOX_YMIN line */
  if( generator[i].coord.y <= yLower )
  {
    generator[nsites].coord.x = generator[i].coord.x;
    generator[nsites].coord.y =
              2.0 * BOX_YMIN - generator[i].coord.y;
```

```
      generator[nsites].numcell = 0;
      generator[nsites].genid = 0;      /* outside box */
      nsites++;
    }

    /* reflect around the y=BOX_YMAX line */
    if( generator[i].coord.y >= yUpper )
    {
      generator[nsites].coord.x = generator[i].coord.x;
      generator[nsites].coord.y =
                2.0 * BOX_YMAX - generator[i].coord.y;
      generator[nsites].numcell = 0;
      generator[nsites].genid = 0;      /* outside box */
      nsites++;
    }
  }

  /* change the box coordinates to account for reflections */
   xmin = BOX_XMIN - BOX_REFLECTION * (BOX_XMAX - BOX_XMIN);
   ymin = BOX_YMIN - BOX_REFLECTION * (BOX_YMAX - BOX_YMIN);

   xmax = BOX_XMAX + BOX_REFLECTION * (BOX_XMAX - BOX_XMIN);
   ymax = BOX_YMAX + BOX_REFLECTION * (BOX_YMAX - BOX_YMIN);

} /* ReflectPoints */



/*******************************************************
*  new_generators()
*
*  samples density using von Neumann acceptance-rejection
*  method to produce a set of generators for Lloyd
*  algorithm
*******************************************************/
void new_generators( void )
{
  int i;
  struct Point pttry;
```

```
  double rhotry;

  srand( SRAND_SEED );

  /* put generators in Fortunes sites[] array */
  for(i=0; i<ENSEMBLE; i++)
  {
    /* repeat till acceptable point */
    while(1)
    {
      pttry.x = ( (double)rand() /
            ((double)RAND_MAX + (double)1.0) )
            * (BOX_XMAX - BOX_XMIN)
            + BOX_XMIN;

      pttry.y = ( (double)rand() /
            ((double)RAND_MAX + (double)1.0) )
            * (BOX_YMAX - BOX_YMIN)
            + BOX_YMIN;

      rhotry = ( (double)rand() /
            ((double)RAND_MAX + (double)1.0) )
            * RHO_MAX;

      if( rho(&pttry) >= rhotry ) break;   /* got one! */
    }

    /* use this point */
    generator[i].coord.x = pttry.x;
    generator[i].coord.y = pttry.y;
    generator[i].numcell = 0;      /* init. cell count */
    generator[i].genid = i+1;      /* inside box */
  }

} /* new_generators */
```

```
/*********************************************************
* scomp()
*
* Fortune's function modified.
*
* Sorts sites on y, then x coord.
* Uses Generator structure instead of Point.
*********************************************************/
int scomp(const void *a, const void *b)
{
  struct Generator *s1 = (struct Generator *)a;
  struct Generator *s2 = (struct Generator *)b;

  if(s1 -> coord.y < s2 -> coord.y) return -1;
  if(s1 -> coord.y > s2 -> coord.y) return 1;
  if(s1 -> coord.x < s2 -> coord.x) return -1;
  if(s1 -> coord.x > s2 -> coord.x) return 1;

  return 0;
} /* scomp */
```

lloyd.c

```
/*********************************************************
*  Routines needed to perform iterations of the Lloyd
*  algorithm to compute centroidal Voronoi tessellations.
*********************************************************/

#include <math.h>
#include <stdlib.h>
#include "defs.h"

/*********************************************************
*  center_generators()
*
*  moves the current positions of the generators to the
*  centroid of the generators Voronoi polygon (cell).
*  Assumes the density is planar over each sub-triangle
*  of the polygon.
*********************************************************/
```

```
void center_generators(void)
{
  int i, j;

  double tot_mass, mass;

  struct Point com, poly_com;

  for(j=0; j<ENSEMBLE; j++)
  {
    mass = tot_mass = 0.0;
    com.x = com.y = 0.0;
    poly_com.x = poly_com.y = 0.0;

    /* find centroid of generator's polygon */
    for(i=0; i<(generator[j].numcell-1); i++)
    {
      /* mass of the i-th triangle of the polygon */
      mass = triangle_center( &(generator[j].coord),
            &(generator[j].cell[i]),
            &(generator[j].cell[i+1]),
            &com
            );
      tot_mass += mass;

      /* centroid is weighted sum over triangles */
      poly_com.x += mass * com.x;
      poly_com.y += mass * com.y;
    }

    /* assign generator the new coordinates */
    generator[j].coord.x = poly_com.x / tot_mass;
    generator[j].coord.y = poly_com.y / tot_mass;
    generator[j].numcell = 0;
  }

} /* center_generators */
```

```
/*******************************************************
*  GetPolygons()
*
*  Computes ordered polygon for each generator
*******************************************************/
void GetPolygons(void)
{
  int i;

  /* find center of each triangle's circle */
  for(i=0; i<myNumTriangles; i++)
  {
    TriangleCenter(
      &(generator[theTriangle[i].gen1].coord),
      &(generator[theTriangle[i].gen2].coord),
      &(generator[theTriangle[i].gen3].coord),
      &(theTriangle[i].center)
    );
  }

  /* add vertices (the centers) to polygon list */
  for(i=0; i<myNumTriangles; i++)
  {
    /* only for generators inside the box */
    if(generator[theTriangle[i].gen1].genid)
    {
      generator[theTriangle[i].gen1]
        .cell[generator[theTriangle[i].gen1].numcell]
         = theTriangle[i].center;

      generator[theTriangle[i].gen1].numcell++;
    }

    if(generator[theTriangle[i].gen2].genid)
    {
      generator[theTriangle[i].gen2]
        .cell[generator[theTriangle[i].gen2].numcell]
         = theTriangle[i].center;
```

```
      generator[theTriangle[i].gen2].numcell++;
    }

    if(generator[theTriangle[i].gen3].genid)
    {
      generator[theTriangle[i].gen3]
        .cell[generator[theTriangle[i].gen3].numcell]
         = theTriangle[i].center;

      generator[theTriangle[i].gen3].numcell++;
    }
  }

  /* order the vertices */
  for(i=0; i<nsites; i++)
  {
    if(generator[i].genid)  /* inside the box */
      sort_vertices( &generator[i] );
  }

} /* GetPolygons */



/*********************************************************
*  sort_vertices()
*
*  orders generator's polygon vertices by angle up from
*  right horizontal ray. Also removes any duplicates.
*********************************************************/
void sort_vertices( struct Generator *g )
{
  int i;
  int numTemp = 0;
  double lastAngle = -1.0;

  struct Point temp[MAX_NUM_VERTICES];
```

```
  struct vertix_angle vAngles[MAX_NUM_VERTICES];

  /* initialize vertix_angle structure */
  for(i = 0; i<g->numcell; i++)
  {
    vAngles[i].angle =
                what_angle( &(g->coord), &(g->cell[i]) );
    vAngles[i].vertixID = i;
  }

  /* sort the vertix_angles */
  qsort(  vAngles, g->numcell,
          sizeof *vAngles, angle_comp );

  /* build sorted list of vertices removing duplicates */
  for(i=0; i<g->numcell; i++)
  {
    if(vAngles[i].angle != lastAngle)
    {
      temp[numTemp] = g->cell[vAngles[i].vertixID];
      numTemp++;
      lastAngle = vAngles[i].angle;
    }
  }

  /* transfer temp list back to the generator */
  for(i=0; i<numTemp; i++)
  {
    g->cell[i] = temp[i];
  }
  g->cell[numTemp]= temp[0]; /* close the polygon */
  g->numcell = ++numTemp;

} /* sort_vertices */
```

```
/*******************************************************
* generator_comp()
*
* compare generators based on their id's
*******************************************************/
int generator_comp(const void *a, const void *b)
{
  struct Generator *g1 = (struct Generator *)a;
  struct Generator *g2 = (struct Generator *)b;

  if( g1->genid > g2->genid ) return 1;
  if( g1->genid < g2->genid ) return -1;

  return 0;
} /* generator_comp */



/*******************************************************
* angle_comp()
*
* decides which two angles is bigger
*******************************************************/
int angle_comp(const void *a, const void *b)
{
  struct vertix_angle *s1 = (struct vertix_angle *)a;
  struct vertix_angle *s2 = (struct vertix_angle *)b;

  if( s1->angle > s2->angle ) return 1;
  if( s1->angle < s2->angle ) return -1;

  return 0;
} /* angle_comp */
```

```
/*********************************************************
*  what_angle()
*
*  returns the angle a line segment makes with the
*  horizontal right going ray
*********************************************************/
double what_angle(struct Point *p1, struct Point *p2)
{
  double a, b;
  double dx = p2->x - p1->x;
  double dy = p2->y - p1->y;

  /* deal with indeterminate cases */
  if( dx == 0.0 )
  {
    if( dy > 0.0 )
      return M_PI / 2.0;
    else
      return 3.0 * M_PI / 2.0;
  }

  /* figure out which quadrant and return angle a */
  b = (double) atan( (double) dy / dx );
  if( dx > 0.0 && dy >= 0.0)
    a = b;
  else if( dx > 0.0 && dy < 0.0 )
    a = 2.0 * M_PI + b;
  else /* dx < 0.0 */
    a = M_PI + b;

  return a;
} /* what_angle */
```

```
/*********************************************************
*  TriangleCenter()
*
*  finds center of circle that intersects the 3 points
*  of the triangle
*********************************************************/
void TriangleCenter(struct Point *p1, struct Point *p2,
                    struct Point *p3,
                    struct Point *result )
{
  double x1 = p1->x;  double y1 = p1->y;
  double x2 = p2->x;  double y2 = p2->y;
  double x3 = p3->x;  double y3 = p3->y;

  result->x = (x3*x3*(y1 - y2) +
               (x1*x1 + (y1 - y2)*(y1 - y3))*
               (y2 - y3) + x2*x2*(y3 - y1))/
               (2.0*(x3*(y1 - y2) +
               x1*(y2 - y3) + x2*(y3 - y1)));

  result->y = (-(x2*x2*x3) + x1*x1*(x3 - x2) +
               x3*(y1*y1 - y2*y2) +
               x1*(x2*x2 - x3*x3 + y2*y2 - y3*y3) +
               x2*(x3*x3 - y1*y1 + y3*y3))/
               (2.0*(x3*(y1 - y2) +
               x1*(y2 - y3) + x2*(y3 - y1)));

} /* TriangleCenter */
```

triangle.c

```
#include <math.h>
#include <stdlib.h>
#include "defs.h"

/*********************************************************
*  triangle_center()
*
*  returns the mass of a triangle defined by Points p1,
*  p2, and p3. Also, set the variable Point result to the
*  center of mass of the triangle. Mass is probability.
*  The numerical integration on triangle performed as
*  described by D.A. Dunavant, Int. J. Num. Meth. Eng.,
*  Vol. 21, 1129--1148 (1985).
*********************************************************/
double triangle_center( struct Point *p1,
                        struct Point *p2,
                        struct Point *p3,
                        struct Point *result)
{
  static double weight[25] = {
   0.09081799038275, 0.03672595775647, 0.03672595775647,
   0.03672595775647, 0.04532105943553, 0.04532105943553,
   0.04532105943553, 0.07275791684542, 0.07275791684542,
   0.07275791684542, 0.07275791684542, 0.07275791684542,
   0.02832724253106, 0.02832724253106, 0.02832724253106,
   0.02832724253106, 0.00942166696373, 0.00942166696373,
   0.00942166696373, 0.00942166696373, 0.00942166696373,
   0.00942166696373  };

  static double coord[25][3] = {
    {0.33333333333333,0.33333333333333,0.33333333333333},
    {0.02884473323269,0.48557763338366,0.48557763338366},
    {0.48557763338366,0.02884473323269,0.48557763338366},
    {0.48557763338366,0.48557763338366,0.02884473323269},
    {0.78103684902993,0.10948157548504,0.10948157548504},
    {0.10948157548504,0.78103684902993,0.10948157548504},
    {0.10948157548504,0.10948157548504,0.78103684902993},
```

```
  {0.14170721941488,0.30793983876412,0.55035294182100},
  {0.14170721941488,0.55035294182100,0.30793983876412},
  {0.30793983876412,0.14170721941488,0.55035294182100},
  {0.30793983876412,0.55035294182100,0.14170721941488},
  {0.55035294182100,0.14170721941488,0.30793983876412},
  {0.55035294182100,0.30793983876412,0.14170721941488},
  {0.02500353476269,0.24667256063990,0.72832390459741},
  {0.02500353476269,0.72832390459741,0.24667256063990},
  {0.24667256063990,0.02500353476269,0.72832390459741},
  {0.24667256063990,0.72832390459741,0.02500353476269},
  {0.72832390459741,0.02500353476269,0.24667256063990},
  {0.72832390459741,0.24667256063990,0.02500353476269},
  {0.00954081540030,0.06680325101220,0.92365593358750},
  {0.00954081540030,0.92365593358750,0.06680325101220},
  {0.06680325101220,0.00954081540030,0.92365593358750},
  {0.06680325101220,0.92365593358750,0.00954081540030},
  {0.92365593358750,0.00954081540030,0.06680325101220},
  {0.92365593358750,0.06680325101220,0.00954081540030}
  };

double area = 0.5 * fabs( (p3->x - p1->x) *
                          (p2->y - p1->y) -
                          (p2->x - p1->x) *
                          (p3->y - p1->y) );
double mass   = 0.0;
double massX  = 0.0;
double massY  = 0.0;
double rhoAtp;
struct Point p;
int i;

for(i=0; i<25; i++)
{
  p.x = coord[i][0] * (p1->x) +
        coord[i][1] * (p2->x) +
        coord[i][2] * (p3->x);
```

```
    p.y = coord[i][0] * (p1->y) +
          coord[i][1] * (p2->y) +
          coord[i][2] * (p3->y);

    rhoAtp = rho( &p );
    massX += weight[i] * (p.x) * rhoAtp;
    massY += weight[i] * (p.y) * rhoAtp;
    mass += weight[i] * rhoAtp;
  }

  result->x = massX / mass;
  result->y = massY / mass;
  return area * mass;

} /* triangle_center */
```

rho.c

```
#include "defs.h"
#include <math.h>

/*********************************************************
* rho()
*
* returns the value of the density at Point p given the
* time tnow. Time runs from 0 to 1.25*2000*M_PI before
* the density almost repeats.
*
* Box width is 100 on both sides.
*********************************************************/
double rho(struct Point *p)
{
  /* the terms with tnow in them */
  double t1 = cos( tnow / 2000.0 );
  double t2 = sin( tnow / 2000.0 );
  double t3 = cos( 13.0 * tnow / 10000.0 );
  double t4 = sin( 13.0 * tnow / 10000.0 );
```

```
  /* the terms with x in them */
  double x1 = sin( M_PI * (p->x) / 100.0 );
  double x2 = sin( M_PI * (p->x) / 50.0 );

  /* the terms with y in them */
  double y1 = sin( M_PI * (p->y) / 50.0 );
  double y2 = sin( 3.0 * M_PI * (p->y) / 100.0 );

  double r = (  (t1 * t1 + t2 * t2) *
                (x1 * x1 * y1 * y1) +
                (t1 * t3 + t2 * t4) *
                (2.0 * x1 * x2 * y1 * y2) +
                (x2 * x2 * y2 * y2)
             ) / 50.0;

  return r*r;   /* must square for 2D CVT trajectories */
} /* rho */
```

output.c

```
/*********************************************************
* out_triple()
*
* Replacement of Fortune's out_triple() function. Instead
* of saving to drive, the information is collected in
* the theTriangle[] array.
*********************************************************/
void out_triple(struct Site *s1, struct Site *s2,
                struct Site *s3)
{
  theTriangle[myNumTriangles].gen1 = s1->sitenbr;
  theTriangle[myNumTriangles].gen2 = s2->sitenbr;
  theTriangle[myNumTriangles].gen3 = s3->sitenbr;
  myNumTriangles++;
}
```

Appendix D

Program Listing: Measuring Wave Function for Two-Slit Experiment

The C code for the simulation to measure the wave function of the two-slit experiment as described in §5.2.1.1. The calculation is comprised of six programs: get_data.c, get_coeffs.c, get_trajs.c, get_traj_fits.c, get_R.c, get_S.c. All programs share defs.h.

defs.h

```
/*************************
* parameter declarations *
*************************/
#define SEED    101             /* random number */
#define ESTEPS  21              /* exp. time steps + 1 */
#define EPOINTS 1e5             /* exp. points per step */
#define TMIN    0.0             /* time boundaries */
#define TMAX    975.0e-6
#define XMIN    -5.25e-5        /* space boundaries */
#define XMAX    +4.75e-5

#define DT      5e-7            /* half time resolution */
#define DX      5e-7            /* half space resolution */
#define DRES    1e3             /* find density max res. */

#define FK      40              /* Fourier coeffs order */

#define NTRAJS  10000           /* # trajectories */
#define TSTEPS  10              /* steps btwn ESTEPS */

#define FITNUM  20              /* neighbors for fitting */
```

```
#define HBAR    1.05457148e-34
#define MASS    6.64632e-27
#define HBARMASS (HBAR*HBAR)/(2.0*MASS)

#define DN      5               /* deriv. order = 2*DN */
#define SX      1e-6            /* sampling distance */
```

get_data.c

```
/*******************************************************
! Purpose:
! Using a prescribed probability density, sample
! the density a number of times at various time steps.
! Save the points at each time step in separate files.
*******************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "defs.h"

/* function declarations */
double density( double x, double t );

int main(int argc, char *argv[])
{
  /* variable declarations */
  int ets;      /* exp. time step */
  int i;

  double point; /* new point */
  double dmax;  /* maximum of density at step */
  double dtry;  /* density value try */
  double tnow;  /* current time */
  double terr;  /* error around time */

  FILE *data_file;
  char data_name[40];
```

```
/* begin */
srand(SEED);

/* main time loop */
printf( "Getting␣data␣for␣experimental␣time␣step␣\n" );

for( ets = 0; ets < ESTEPS; ets++ )
{
  printf("\t%d\n", ets);

  /* open step's file */
  sprintf( data_name, "data/data_%d.txt", ets );
  data_file = fopen( data_name, "w" );
  if( data_file == NULL )
  {
    printf( "Couldn't␣open␣necessary␣data␣file␣%d\n",
             ets );
    return -1;
  }

  /* calculate density maximum for this step */
  /* assume no time error to find maximum */
  dmax = 0.0;
  tnow = (TMAX-TMIN) * ets / ESTEPS + TMIN;
  for( i = 0; i <= DRES; i++)
  {
    dtry = density( (XMAX-XMIN) * i / DRES + XMIN,
                    tnow );
    if( dtry > dmax ) dmax = dtry;
  }
  dmax = 1.05 * dmax;   /* make a little bigger */
```

```
    /* get sampled points */
    for( i = 0; i < EPOINTS; i++ )
    {
      /* get one point */
      do{
        point = ( (double)rand() /
                  ((double)RAND_MAX + (double)1.0) ) *
                  (XMAX-XMIN) + XMIN;
        dtry = dmax * ( (double)rand() /
                        ((double)RAND_MAX + (double)1.0) );
        terr = DT * (
                ( (double)rand() /
                  ((double)RAND_MAX + (double)1.0) ) *
                  2.0 - 1.0 );
        if( density( point, tnow + terr )
            >= dtry )
          break; /* keep point */
      } while (1);

      /* put point into bin */
      point = floor( (point - XMIN)/(2.0*DX) ) *
              2.0 * DX + XMIN + DX;

      /* save this point */
      fprintf( data_file, "%lf\n", point );
    }

    /* close file */
    fclose( data_file );
  }

  return 0;
} /* main */
```

```
/*********************************************************
* density()
*
* returns value of probability density at (x,t)
*********************************************************/
double density( double x, double t )
{
  double r = 0.0;
  double ts = 1.0 + 3.03532e7 * t * t;

  r = 332452.0 *
      exp(  (-22.2222 - 3.47222e11 * x * x) / ts ) *
      ( cos( 3.06076e10 * t * x / ts ) +
        cosh( 5.55556e6 * x / ts ) ) /
      sqrt(ts);

  return r;
} /* density */
```

get_coeffs.c

```
/*********************************************************
! Purpose:
! Reads in data at experimental time steps, and computes
! the Fourier coefficients, which are then saved.
*********************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "defs.h"

/* function declarations */
void rescaledata( double x[], double xmin,
                              double xmax );
```

```
int main(int argc, char *argv[])
{
  /* variable declarations */
  int ets;          /* experimental time step */
  int i,k;

  double data[(int)EPOINTS];
  double xmin, xmax;
  double b_re[2*FK + 1];  /* Fourier coeffs (real) */
  double b_im[2*FK + 1];  /* (imaginary)           */

  FILE *data_file;
  FILE *coeffs_file;
  char file_name[40];
  int ferr;

  printf("Computing␣coefficients␣for␣time␣step␣...␣\n");

  /* time loop */
  for( ets = 0; ets < ESTEPS; ets++)
  {
    /* what's going on */
    printf( "\t%d\n", ets );

    /* open data file */
    sprintf( file_name, "data/data_%d.txt", ets );
    data_file = fopen( file_name, "r" );
    if( data_file == NULL )
    {
      printf( "Couldn't␣open␣necessary␣data␣file␣%d\n",
              ets );
      return -1;
    }
```

```
/* open coeffs file */
sprintf( file_name, "data/coeffs_%d.txt", ets );
coeffs_file = fopen( file_name, "wb" );
if( coeffs_file == NULL )
{
  printf( "Couldn't open necessary coeffs file %d\n",
          ets );
  return -1;
}

/* load data */
for(i=0; i<(int)EPOINTS; i++)
{
  fscanf( data_file, "%lf", &data[i] );
}

/* set boundaries */
xmin = XMIN;
xmax = XMAX;

/* rescale data */
rescaledata( data, xmin, xmax );

/* loop over k */
for( k=-FK; k<=FK; k++)
{
  b_re[k+FK] = 0.0;
  b_im[k+FK] = 0.0;

  /* loop over data */
  for(i=0; i<EPOINTS; i++)
  {
    /* calculate coeffs */
    b_re[k+FK] = b_re[k+FK] +
                   cos( 2.0 * M_PI * k * data[i] );
    b_im[k+FK] = b_im[k+FK] +
                   sin( 2.0 * M_PI * k * data[i] );
  }
```

```
      /* fix b_im */
      b_im[k+FK] = - b_im[k+FK];

    } /* k */

    /* save boundaries and coefficients */
    fwrite( &xmin, sizeof(double), 1, coeffs_file );
    fwrite( &xmax, sizeof(double), 1, coeffs_file );
    fwrite( b_re, sizeof(double), 2*FK+1, coeffs_file );
    fwrite( b_im, sizeof(double), 2*FK+1, coeffs_file );

    fclose( data_file );
    fclose( coeffs_file );

  } /* time */

  return 0;
} /* main */


/*********************************************************
* rescaledata()
*
* rescale data in x[] using xmin and xmax
*********************************************************/
void rescaledata( double x[], double xmin, double xmax )
{
  int i;

  double m = 1.0 / (xmax - xmin);

  for(i=0; i<EPOINTS; i++)
  {
    x[i] = m * ( x[i] - xmin );
  }

} /* rescaledata */
```

get_trajs.c

```
/*********************************************************
! Purpose:
! Reads in Fourier coefficients and computes the Bohm
! trajectories using probability conservation and the
! cumulative probability function. The coefficients from
! two successive experimental time steps are combined
! with a weighted average to estimate densities between
! the experimental data time steps.
*********************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "defs.h"

/* global variables */
double br[2*FK+1];   /* avg. Fourier coeffs. */
double bi[2*FK+1];

/* function declarations */
double findx( double rvalue );
double cpf( double x );

int main(int argc, char *argv[])
{
  /* variable declarations */
  int ets;          /* experimental time step */
  int tts;          /* traj time step */
  int i,j;

  double rvals[NTRAJS], cpfvals[NTRAJS];
  double xtemp;
  double *trajs;
  double br1[2*FK+1], bi1[2*FK+1];
  double br2[2*FK+1], bi2[2*FK+1];

  FILE *coeffs_file;
```

```
FILE *trajs_file;
char file_name[40];
int ferr;

/* allocate trajs space */
trajs = (double *)malloc(
     (NTRAJS*((ESTEPS-1)*TSTEPS + 1))*sizeof(double) );
if( trajs == NULL )
{
  printf( "Could␣not␣allocate␣trajectory␣space.\n" );
  return -1;
}

/* set up r values between (0,1) */
for( i=0; i<NTRAJS; i++)
{
  rvals[i] = (double)(i+1) / (NTRAJS+1.0);
}

/* time loop */
printf( "Computing␣trajectories␣for␣time␣step␣..." );
for( ets = 0; ets < ESTEPS-1; ets++)
{
  /* status */
  printf( "\n\t%d\t", ets );

  /* load fourier coefficients for 2 densities */
  sprintf( file_name, "data/coeffs_%d.txt", ets );
  coeffs_file = fopen( file_name, "rb" );
  if( coeffs_file == NULL )
  {
    printf( "Couldn't␣open␣necessary␣coeffs␣file␣%d\n",
            ets );
    return -1;
  }

  /* don't need first 2 values in coeffs_file */
  fread( &xtemp, sizeof(double), 1, coeffs_file );
  fread( &xtemp, sizeof(double), 1, coeffs_file );
```

```
fread( br1, sizeof(double), 2*FK+1, coeffs_file );
fread( bi1, sizeof(double), 2*FK+1, coeffs_file );
fclose( coeffs_file );

sprintf( file_name, "data/coeffs_%d.txt", ets + 1 );
coeffs_file = fopen( file_name, "rb" );
if( coeffs_file == NULL )
{
  printf( "Couldn't␣open␣necessary␣coeffs␣file␣%d\n",
          ets );
  return -1;
}

/* don't need first 2 values in coeffs_file */
fread( &xtemp, sizeof(double), 1, coeffs_file );
fread( &xtemp, sizeof(double), 1, coeffs_file );
fread( br2, sizeof(double), 2*FK+1, coeffs_file );
fread( bi2, sizeof(double), 2*FK+1, coeffs_file );
fclose( coeffs_file );

/* trajectory time loop */
for( tts = 0; tts < TSTEPS; tts++)
{
  /* status */
  printf( "%d␣", tts );

  /* weighted average of coefficients */
  for( i=0; i<=2*FK; i++)
  {
    br[i] = (1.0 - (double)tts/TSTEPS) * br1[i] +
            ((double)tts/TSTEPS) * br2[i];
    bi[i] = (1.0 - (double)tts/TSTEPS) * bi1[i] +
            ((double)tts/TSTEPS) * bi2[i];
  }

  for( i=0; i<NTRAJS; i++)
  {
    /* x value [0,1] for r value */
    xtemp = findx( rvals[i] );
```

```
      /* rescale x, and add to trajs */
      trajs[i * ((ESTEPS-1)*TSTEPS + 1) +
                (ets*TSTEPS + tts)] =
                (XMAX-XMIN)*xtemp + XMIN;
    }

  } /* trajectory time */

} /* time */

printf("fts\n");
/* final time step */
sprintf( file_name, "data/coeffs_%d.txt", ESTEPS-1 );
coeffs_file = fopen( file_name, "rb" );
if( coeffs_file == NULL )
{
  printf( "Couldn't open necessary coeffs file %d\n",
          ets );
  return -1;
}

/* don't need first 2 values in coeffs_file */
fread( &xtemp, sizeof(double), 1, coeffs_file );
fread( &xtemp, sizeof(double), 1, coeffs_file );
fread( br, sizeof(double), 2*FK+1, coeffs_file );
fread( bi, sizeof(double), 2*FK+1, coeffs_file );
fclose( coeffs_file );

for( i=0; i<NTRAJS; i++)
{
  /* x values [0,1] for final time step */
  xtemp = findx( rvals[i] );

  /* rescale x, and add to trajs */
  trajs[i * ((ESTEPS-1)*TSTEPS + 1) +
            ((ESTEPS-1)*TSTEPS) ] =
              (XMAX-XMIN)*xtemp + XMIN;
}
```

```
  /* output trajectories */
  sprintf( file_name, "data/trajs.txt");
  trajs_file = fopen( file_name, "wb" );
  if( trajs_file == NULL )
  {
    printf( "Couldn't open necessary trajs file\n" );
    return -1;
  }

  fwrite( trajs, sizeof(double),
          NTRAJS*((ESTEPS-1)*TSTEPS+1), trajs_file );

  /* clean up */
  fclose( trajs_file );
  free(trajs);

  return 0;
} /* main */

/*********************************************************
* findx()
*
* finds x that corresponds to r by, x = CPF-1(r)
*********************************************************/
double findx( double rvalue )
{
  double a = 0.0, b = 1.0;
  double ya, yb;
  double xm, ym;

  int i;

  ya = cpf(a) - rvalue;
  yb = cpf(b) - rvalue;

  if( floor(ya*yb) > 0 ) return 0.0;
```

```
  /* fixed number of iterations of 50 */
  for( i=0; i<50; i++)
  {
    xm = (a + b) / 2.0;
    ym = cpf(xm) - rvalue;

    if( floor(ya*ym) < 0 )
    {
      b = xm;
      yb = ym;
    }
    else
    {
      a = xm;
      ya = ym;
    }
  }

  return xm;
} /* findx */


/*********************************************************
* cpf()
*
* computes CPF(x) using global Fourier coeffs
*********************************************************/
double cpf( double x )
{
  int i;
  const double TWOPI = 2.0 * M_PI;
  double sum;
  double norm = 1.0 / ( 1.0 + 2.0 * br[FK] );

  sum = x / norm;
```

```
  /* Fourier coefficients less than zero */
  for(i=0; i<FK; i++)
  {
    sum = sum +
          ( br[i] * sin( TWOPI * (i-FK) * x ) -
            2.0 * bi[i] *
            pow( sin( M_PI * (i-FK) * x ), 2.0 )
          ) / ( (i-FK) * M_PI );
  }

  /* Fourier coefficients greater than zero */
  for(i=FK+1; i<=2*FK; i++)
  {
    sum = sum +
          ( br[i] * sin( TWOPI * (i-FK) * x ) -
            2.0 * bi[i] *
            pow( sin( M_PI * (i-FK) * x ), 2.0 )
          ) / ( (i-FK) * M_PI );
  }

  /* normalize */
  sum = sum * norm;

  /* create limits */
  if( sum < 1e-4 )
    sum = 0.0;
  else if( sum > 0.9999 )
    sum = 1.0;

  return sum;
} /* cpf */
```

get_traj_fits.c

```
/*************************************************************
! Purpose:
! Reads in estimated particle trajectories. For each
! position uses a local least squares to find the
! smoothed out trajectory using a quadratic fit. Saves
! the computed fit parameters.
*************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "defs.h"

/* macros */
#define TRAJ_MAX ((ESTEPS-1)*TSTEPS+1)
#ifndef max
  #define max( a, b ) ( ((a) > (b)) ? (a) : (b) )
#endif

#ifndef min
  #define min( a, b ) ( ((a) < (b)) ? (a) : (b) )
#endif

/* function declarations */
void localfit(  double x[], double t[], int cnt,
                double *a, double *b, double *c );

int main(int argc, char *argv[])
{
  /* variable declarations */
  int trj;
  int i,j,fnum;

  double traj[TRAJ_MAX];
  double times[TRAJ_MAX];

  double a,b,c;         /* quadratic parameters */
```

```
FILE *trajs_file;
FILE *fits_file;

/* open files */
trajs_file = fopen( "data/trajs.txt", "rb" );
if( trajs_file == NULL )
{
  printf( "Couldn't␣open␣data/trajs.txt\n" );
  return -1;
}

fits_file = fopen( "data/traj_fits.txt", "wb" );
if( fits_file == NULL )
{
  printf( "Couldn't␣open␣data/traj_fits.txt\n" );
  return -1;
}

/* set up time steps */
for( i=0; i<TRAJ_MAX; i++ )
{
  times[i] = TMIN + (double)i *
                    (TMAX-TMIN) / (TRAJ_MAX-1.0);
}

/* loop over trajs */
printf( "Fitting␣trajectory␣...\n" );
for( trj=0; trj<NTRAJS; trj++)
{
  printf( "␣%d", trj );

  /* load in next traj */
  fread( traj, sizeof(double), TRAJ_MAX, trajs_file );
```

```
    /* locally fit each position */
    for(i=0; i<TRAJ_MAX; i++)
    {
      /* send fitting index */
      j = max( i-FITNUM/2, 0 );
      if( i > FITNUM/2 )
      {
        fnum = min( FITNUM, TRAJ_MAX-j );
      }
      else
      {
        fnum = FITNUM/2 + i;
      }

      localfit( &traj[j], &times[j], fnum, &a, &b, &c );

      /* save quadratic fits for each position */
      fwrite( &a, sizeof(double), 1, fits_file );
      fwrite( &b, sizeof(double), 1, fits_file );
      fwrite( &c, sizeof(double), 1, fits_file );
    }

  } /* traj loop */

  /* clean up */
  printf( "\n" );
  fclose( trajs_file );
  fclose( fits_file );

  return 0;
} /* main */
```

```
/*********************************************************
* localfit()
*
* finds local quadratic fit a t^2 + b t + c using data
* x[] and t[] where cnt is the array size
*********************************************************/
void localfit(  double x[], double t[], int cnt,
                double *a, double *b, double *c )
{
  int i;

  double St = 0.0;
  double Sx = 0.0;
  double St2 = 0.0;
  double Sxt = 0.0;
  double St3 = 0.0;
  double St4 = 0.0;
  double Sxt2 = 0.0;

  double denom;

  for(i=0; i<cnt; i++)
  {
    St += t[i];
    Sx += x[i];
    St2 += t[i]*t[i];
    St3 += t[i]*t[i]*t[i];
    St4 += t[i]*t[i]*t[i]*t[i];
    Sxt += x[i]*t[i];
    Sxt2 += x[i]*t[i]*t[i];
  }

  denom = St2*St2*St2 + cnt*St3*St3 +
          St*St*St4 - St2*(2.0*St*St3 + cnt*St4);

  *a = ( St2*St2*Sx - St*St3*Sx + cnt*St3*Sxt +
          St*St*Sxt2 - St2*(St*Sxt + cnt*Sxt2) )/denom;
```

```
  *b = ( St*St4*Sx + St2*St2*Sxt - cnt*St4*Sxt +
          cnt*St3*Sxt2 - St2*(St3*Sx + St*Sxt2) )/denom;

  *c = ( St3*St3*Sx - St2*St4*Sx + St*St4*Sxt +
          St2*St2*Sxt2 - St3*(St2*Sxt + St*Sxt2) )/denom;

} /* localfit */
```

get_R.c

```
/*********************************************************
! Purpose:
! Reads in coeffs fit for density estimator, and the
! local least square fits for the trajectories. Computes
! and saves the particle location, estimated density, and
! actually density.
*********************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "defs.h"

/* macros */
#define TRAJ_MAX ((ESTEPS-1)*TSTEPS+1)

/* global variables */
double br[ESTEPS][2*FK+1];          /* Fourier coeffs. */
double bi[ESTEPS][2*FK+1];

/* function declarations */
double rhoE( double x, int ts );
double rhoA( double x, double t );
```

```
int main(int argc, char *argv[])
{
  /* variable declarations */
  int i,j;

  double fits[TRAJ_MAX][3];
  double time[TRAJ_MAX];
  double xtemp;
  double xnow;
  double rho_est, rho_act;

  FILE *coeffs_file;
  FILE *fits_file;
  FILE *r_file;
  char file_name[40];

  /* define times */
  for(i=0; i<TRAJ_MAX; i++)
  {
    time[i] = TMIN + (double)i *
                      (TMAX-TMIN) / (TRAJ_MAX-1.0);
  }

  /* open fits and r files */
  fits_file = fopen( "data/traj_fits.txt", "rb" );
  r_file = fopen( "data/traj_R.txt", "wb" );
  if( fits_file == NULL || r_file == NULL )
  {
    printf( "Couldn't open fits or R file\n" );
    return -1;
  }

  /* load in coeffs for all times */
  for(i=0; i<ESTEPS; i++)
  {
    sprintf( file_name, "data/coeffs_%d.txt", i );
    coeffs_file = fopen( file_name, "rb" );
```

```
  if( coeffs_file == NULL )
  {
    printf( "Couldn't␣open␣%s\n", file_name );
    return -1;
  }

  /* don't need first 2 values in coeffs_file */
  fread( &xtemp, sizeof(double), 1, coeffs_file );
  fread( &xtemp, sizeof(double), 1, coeffs_file );
  fread( br[i], sizeof(double), 2*FK+1, coeffs_file );
  fread( bi[i], sizeof(double), 2*FK+1, coeffs_file );
  fclose( coeffs_file );
}

printf( "Calculating␣R␣for␣trajectory␣...\n" );

/* loop on trajectories */
for( i=0; i<NTRAJS; i++ )
{
  /* notify world */
  printf( "%d␣", i );

  /* load in current traj_fits */
  fread( &fits[0], sizeof(double),
         3*TRAJ_MAX, fits_file );

  /* loop time */
  for( j=0; j<TRAJ_MAX; j++ )
  {
    /* compute x at this time */
    xnow =  fits[j][0] * time[j] * time[j] +
            fits[j][1] * time[j] +
            fits[j][2];

    /* rescale x [0,1] ?? */
    xtemp = (xnow - XMIN) / (XMAX-XMIN);

    /* compute rhoE(x), rhoA(x) */
    rho_est = rhoE( xtemp, j );
```

```
      rho_act = rhoA( xnow, time[j] );

      /* rescale rhoE(x) ?? */
      rho_est = rho_est / (XMAX-XMIN);

      /* save (x, rhoE, and rhoA) */
      fwrite( &xnow, sizeof(double), 1, r_file );
      fwrite( &rho_est, sizeof(double), 1, r_file );
      fwrite( &rho_act, sizeof(double), 1, r_file );

    } /* end time loop */

  } /* end traj loop */

  /* clean up */
  printf( "\n" );
  fclose( fits_file );
  fclose( r_file );

  return 0;
} /* main */


/*********************************************************
* rhoE()
*
* Estimated density rho at x and time step t. Needs
* global Fourier coeffs br[ESTEPS][2*FK+1] and
* bi[ESTEPS][2*FK+1]
*********************************************************/
double rhoE( double x, int ts )
{
  int i;

  const double TWOPI = 2.0 * M_PI;

  double brA[2*FK+1];
  double biA[2*FK+1];
```

```
  double rho = 0.0;


  /* current ESTEP and ESTEP+1 */
  int et = (ts / TSTEPS);
  int etplus = ts - TSTEPS * et;

  /* find weight averages for br and bi */
  for( i=0; i<2*FK+1; i++ )
  {
    brA[i] = (1.0 - (double)etplus/TSTEPS) * br[et][i] +
             ((double)etplus/TSTEPS) * br[et+1][i];
    biA[i] = (1.0 - (double)etplus/TSTEPS) * bi[et][i] +
             ((double)etplus/TSTEPS) * bi[et+1][i];
  }

  for( i=0; i<2*FK+1; i++ )
  {
    rho += br[et][i] * cos( TWOPI * (i-FK) * x );
    rho -= bi[et][i] * sin( TWOPI * (i-FK) * x );
  }
  rho *= 2.0;
  rho += 1.0;

  rho /= (1.0 + 2.0*br[et][FK]);

  return rho;
} /* rhoE */


/*********************************************************
* rhoA()
*
* Actual density rho at x and time t
*********************************************************/
double rhoA( double x, double t )
{
  double r = 0.0;
  double ts = 1.0 + 3.03532e7 * t * t;
```

```
  r = 332452.0 *
      exp(  (-22.2222 - 3.47222e11 * x * x) / ts ) *
      ( cos( 3.06076e10 * t * x / ts ) +
        cosh( 5.55556e6 * x / ts ) ) /
      sqrt(ts);

  return r;
} /* rhoA */
```

get_S.c

```
/*********************************************************
! Purpose:
! Reads in coeffs fit for density estimator, and the
! local least square fits for the trajectories. Computes
! and saves the particle location, estimated phase, and
! actually phase.
*********************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "defs.h"

/* macros */
#define TRAJ_MAX ((ESTEPS-1)*TSTEPS+1)
#define dt (TMAX / TRAJ_MAX)

/* global variables */
double br[ESTEPS][2*FK+1];      /* Fourier coeffs. */
double bi[ESTEPS][2*FK+1];

double d1[2*DN+1], d2[2*DN+1];  /* derivative coeffs. */

/* function declarations */
double phaseE( double ph, double x, int ts, double v );
double phaseA( double ph, double x, double t, double v);
```

```
double rhoE( double x, int ts );
double rhoA( double x, double t );

int fact( int n );
void setup_d_coeffs( void );

int main(int argc, char *argv[])
{
  /* variable declarations */
  int i,j;

  double fits[TRAJ_MAX][3];   /* fits for current traj */
  double time[TRAJ_MAX];
  double xtemp;
  double xnow;
  double phase_est, phase_act;

  FILE *coeffs_file;
  FILE *fits_file;
  FILE *s_file;
  char file_name[40];

  /* define times */
  for(i=0; i<TRAJ_MAX; i++)
  {
    time[i] = TMIN + (double)i *
                     (TMAX-TMIN) / (TRAJ_MAX-1.0);
  }

  /* setup derivative coeffs */
  setup_d_coeffs();

  /* open fits and r files */
  fits_file = fopen( "data/traj_fits.txt", "rb" );
  s_file = fopen( "data/traj_S.txt", "wb" );
```

```
if( fits_file == NULL || s_file == NULL )
{
  printf( "Couldn't open fits or S file\n" );
  return -1;
}

/* load in coeffs for all times */
for(i=0; i<ESTEPS; i++)
{
  sprintf( file_name, "data/coeffs_%d.txt", i );
  coeffs_file = fopen( file_name, "rb" );
  if( coeffs_file == NULL )
  {
    printf( "Couldn't open %s\n", file_name );
    return -1;
  }

  /* don't need first 2 values in coeffs_file */
  fread( &xtemp, sizeof(double), 1, coeffs_file );
  fread( &xtemp, sizeof(double), 1, coeffs_file );
  fread( br[i], sizeof(double), 2*FK+1, coeffs_file );
  fread( bi[i], sizeof(double), 2*FK+1, coeffs_file );
  fclose( coeffs_file );
}

printf( "Calculating S for trajectory ...\n" );

/* loop on trajectories */
for( i=0; i<NTRAJS; i++ )
{
  /* notify world */
  printf( "%d ", i );

  /* load in current traj_fits */
  fread( &fits[0], sizeof(double),
         3*TRAJ_MAX, fits_file );

  phase_est = 0.0;
  phase_act = 0.0;
```

```
    /* loop time */
    for( j=0; j<TRAJ_MAX; j++ )
    {
      /* compute x at this time */
      xnow =  fits[j][0] * time[j] * time[j] +
              fits[j][1] * time[j] +
              fits[j][2];

      /* compute phases */
      phase_est = phaseE( phase_est, xnow, j,
                    fits[j][0] * time[j] + fits[j][1] );
      phase_act = phaseA( phase_act, xnow, time[j],
                    fits[j][0] * time[j] + fits[j][1] );

      /* save (x, phaseE, and phaseA) */
      fwrite( &xnow, sizeof(double), 1, s_file );
      fwrite( &phase_est, sizeof(double), 1, s_file );
      fwrite( &phase_act, sizeof(double), 1, s_file );

    } /* end time loop */

  } /* end traj loop */

  /* clean up */
  printf( "\n" );
  fclose( fits_file );
  fclose( s_file );

  return 0;
} /* main */
```

```
/*********************************************************
* setup_d_coeffs()
*
* computes derivative coeffs.
*********************************************************/
void setup_d_coeffs( void )
{
  int k;

  d1[DN] = 0.0;

  for(k=1; k<DN+1; k++)
  {
    d1[k+DN] = pow(-1, (double)(k+1)) *
               (double)(fact(DN) * fact(DN)) /
               (double)(fact(DN-k) * fact(DN+k) * k);
    d2[k+DN] = 2.0 * d1[k+DN] / (double)k;
  }

  d2[DN] = 0.0;

  for(k=1; k<DN+1; k++)
  {
    d1[DN-k] = -d1[DN+k];
    d2[DN-k] = d2[DN+k];

    d2[DN] += d2[DN+k];
  }

  d2[DN] *= -2.0;

} /* setup_d_coeffs */
```

```
/*********************************************************
* factorial n!
*********************************************************/
int fact( int n )
{
  int i;
  int f = 1;

  for( i=1; i<=n; i++)
  {
    f *= i;
  }

  return f;
} /* fact */


/*********************************************************
* phaseE()
*
* Estimate of phase based on previous phase ph and the
* position x and time step ts.
*********************************************************/
double phaseE( double ph, double x, int ts, double v )
{
  int i;

  double L = 0.0;
  double T, V, Q, S;
  double C1, C2;

  double a[2*DN+1];
  double r1, r2;

  /* T = 0.5*m*v*v */
  T = 0.5 * MASS * v * v;

  /* V = 0 */
  V = 0.0;
```

```
  /* Q */
  Q = 0.0;

    /* sample rho */
    for(i=0; i<(2*DN+1); i++)
    {
      a[i] = rhoE( x - (i-DN)*SX, ts );
    }

    /* r1 & r2 */
    r1 = 0.0;
    r2 = 0.0;
    for(i=0; i<(2*DN+1); i++)
    {
      r1 += d1[i] * a[i];
      r2 += d2[i] * a[i];
    }
    r1 /= DT;
    r2 /= (DT * DT);

    /* C1 = 0.5 * rho1(x,t) / rho(x,t) */
    C1 = 0.5 * r1 / rhoE(x,ts);

    /* C2 = 0.5 * (rho2(x,t)/rho(x,t) - 4.0*C1*C1 ) */
    C2 = 0.5 * ( r2 / rhoE(x,ts) -
            4.0 * C1 * C1 );

    /* Q = - h*h / (2 * m) * ( C1*C1 + C2 ) */
    Q = - HBARMASS * (
          C1 * C1 + C2 );
  /* L */
  L = T + V + Q;

  /* S = ph + L*dt */
  S = ph + L * dt;

  return S;
} /* phaseE */
```

```
/*********************************************************
* phaseA()
*
* Estimate of phase based on previous phase ph and the
* position x and time step ts.
*********************************************************/
double phaseA( double ph, double x, double t, double v )
{
  int i;

  double L = 0.0;
  double T, V, Q, S;
  double C1, C2;

  double a[2*DN+1];
  double r1, r2;

  /* T = 0.5*m*v*v */
  T = 0.5 * MASS * v * v;

  /* V = 0 */
  V = 0.0;

  /* Q */
  Q = 0.0;

    /* sample rho */
    for(i=0; i<(2*DN+1); i++)
    {
      a[i] = rhoA( x - (i-DN)*SX, t );
    }

    /* r1 & r2 */
    r1 = 0.0;
    r2 = 0.0;
    for(i=0; i<(2*DN+1); i++)
```

```
    {
      r1 += d1[i] * a[i];
      r2 += d2[i] * a[i];
    }
    r1 /= DT;
    r2 /= (DT * DT);

    /* C1 = 0.5 * rho1(x,t) / rho(x,t) */
    C1 = 0.5 * r1 / rhoA(x,t);

    /* C2 = 0.5 * (rho2(x,t)/rho(x,t) - 4.0*C1*C1 ) */
    C2 = 0.5 * ( r2 / rhoA(x,t) -
            4.0 * C1 * C1 );

    /* Q = - h*h / (2 * m) * ( C1*C1 + C2 ) */
    Q = - HBARMASS * (
           C1 * C1 + C2 );
  /* L */
  L = T + V + Q;

  /* S = ph + L*dt */
  S = ph + L * dt;

  return S;
} /* phaseE */


/*********************************************************
* rhoE()
*
* Estimated density rho at x and time step t. Needs
* global Fourier coeffs br[ESTEPS][2*FK+1] and
* bi[ESTEPS][2*FK+1]
*********************************************************/
double rhoE( double x, int ts )
{
  int i;

  const double TWOPI = 2.0 * M_PI;
```

```
  double brA[2*FK+1];
  double biA[2*FK+1];

  double rho = 0.0;
  double x01 = (x - XMIN) / (XMAX-XMIN);

  /* current ESTEP and ESTEP+1 */
  int et = (ts / TSTEPS);
  int etplus = ts - TSTEPS * et;

  /* find weight averages for br and bi */
  for( i=0; i<2*FK+1; i++ )
  {
    brA[i] = (1.0 - (double)etplus/TSTEPS) * br[et][i] +
             ((double)etplus/TSTEPS) * br[et+1][i];
    biA[i] = (1.0 - (double)etplus/TSTEPS) * bi[et][i] +
             ((double)etplus/TSTEPS) * bi[et+1][i];
  }

  for( i=0; i<2*FK+1; i++ )
  {
    rho += br[et][i] * cos( TWOPI * (i-FK) * x01 );
    rho -= bi[et][i] * sin( TWOPI * (i-FK) * x01 );
  }
  rho *= 2.0;
  rho += 1.0;

  rho /= (1.0 + 2.0*br[et][FK]);

  /* rescale rhoE(x) */
  rho /= (XMAX-XMIN);

  return rho;
} /* rhoE */
```

```
/*********************************************************
* rhoA()
*
* Actual density rho at x and time t
*********************************************************/
double rhoA( double x, double t )
{
  double r = 0.0;
  double ts = 1.0 + 3.03532e7 * t * t;

  r = 332452.0 *
      exp(  (-22.2222 - 3.47222e11 * x * x) / ts ) *
      ( cos( 3.06076e10 * t * x / ts ) +
        cosh( 5.55556e6 * x / ts ) ) /
      sqrt(ts);

  return r;
} /* rhoA */
```

Bibliography

[1] Y. Aharonov, B. G. Englert, and Marlan O. Scully. Protective measurements and Bohm trajectories. *Phys. Lett. A*, 263:137, 1999. 266:216 2000.

[2] Yakir Aharonov and Lev Vaidman. About position measurements which do not show the Bohmian particle position. In James T. Cushing, Arthur Fine, and Sheldon Goldstein, editors, *Bohmian Mechanics and Quantum Theory: An Appraisal*. Kluwer Academic, 1996.

[3] Alain Aspect, Jean Dalibard, and Gerard Roger. Experimental test of Bell's inequalities using time-varying analyzers. *Phys. Rev. Lett.*, 49(25):1804, 1982.

[4] Dmytro Babyuk and Robert E. Wyatt. Multidimensional reactive scattering with quantum trajectories: Dynamics with 50-200 vibrational modes. *J. Chem. Phys.*, 124:214109, 2006.

[5] John R. Barker. On the completeness of quantum hydrodynamics: Vortex formation and the need for both vector and scalar quantum potentials in device simulation. *J. Comp. Elect.*, 1:17–21, 2002.

[6] Frederick J. Belinfante. *A Survey of Hidden-Variables Theories*. Pergamon Press, 1973.

[7] John S. Bell. On the Einstein-Podolsky-Rosen paradox. *Physics*, 1:195, 1964.

[8] John S. Bell. *Speakable and Unspeakable in Quantum Mechanics: Collected Papers in Quantum Mechanics*. Cambridge University Press, 1987.

[9] M. Bienert, F. Haug, W. P. Schleich, and M. G. Raizen. State reconstruction of the kicked rotor. *Phys. Rev. Lett.*, 89(5):050403, 2002.

[10] David Bohm. *Quantum Thoery*. Prentic Hall, Inc., New York, 1951. Dover Publications, Inc. in 1989.

[11] David Bohm. A suggested interpretation of the quantum theory in terms of 'hidden' variables. *Phys. Rev.*, 85:166, 180, 1952.

[12] David Bohm and Basil J. Hiley. *The Undivided Universe*. Routledge, New York, 1993.

[13] Niels Bohr. *Atomic Physics and Human Knowledge*. Science Editions, New York, 1961.

[14] Siegmund Brandt and Hans D. Dahmen. *The Picture Book of Quantum Mechanics*. Springer-Verlag, New York, 3rd edition, 2001.

[15] Siegmund Brandt, Hans D. Dahmen, E. Gjonaj, and T. Stroh. Quantile motion and tunneling. *Phys. Lett. A*, 249:265, 1998.

[16] I. Burghardt and L.S. Cederbaum. Hydrodynamic equations for mixed quantum states. I. General formulation. *J. Chem. Phys.*, 115:10303, 2001.

[17] Timothy M. Coffey, Robert E. Wyatt, and William C. Schieve. Uniqueness of Bohmian Mechanics, and Solutions from Probability Conservation. arXiv: quant-ph: 0710.4099v1, 2007.

[18] Timothy M. Coffey, Robert E. Wyatt, and William C. Schieve. Monte Carlo generation of Bohmian trajectories. *J. Phys. A: Math. Theor.*, 41:335304, 2008.

[19] Timothy M. Coffey, Robert E. Wyatt, and William C. Schieve. Quantum trajectories from kinematic considerations. *J. Phys. A: Math. Theor.*, 43:335301, 2010.

[20] David B. Cook. *Probability and Schrödinger's Mechanics*. World Scientific, New York, 2002.

[21] Louis de Broglie. *Non-Linear Wave Mechanics*. Elsevier Publishing Company, 1960. Translated by A. J. Knodel and J. C. Miller.

[22] Qiang Du and Maria Emelianenko. Acceleration schemes for computing centroidal Voronoi tessellations. *Numer. Linear Algebra Appl.*, 13:173–192, 2006.

[23] Qiang Du and Maria Emelianenko. Uniform convergence of a nonlinear energy-based multilevel quantization scheme. *SIAM J. Num. Anal.*, 46(3):1483–1502, 2008.

[24] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal Voronoi tessellations: applications and algorithms. *SIAM Rev.*, 41(4):637–676, 1999.

[25] Detlef Dürr, Walter Fusseder, Sheldon Goldstein, and Nino Zanghi. Comment on 'Surrealistic Bohm trajectories'. *Z. Naturforsch*, 48a:1261–1262, 1993.

[26] A. Eichenberger, G. Genevés, and P. Gournay. Determination of the planck constant by means of a watt balance. *Eur. Phys. J. Special Topics*, 172:363, 2009.

[27] Maria Emelianenko, Lili Ju, and Alexander Rand. Nondegeneracy and weak global convergence of the Lloyd Algorithm in $\mathbb{R}^D$. *SIAM J. Num. Anal.*, 46(3):1423–1441, 2008.

[28] Berthold-Georg Englert, Marlan O. Scully, Georg Süssmann, and Herbert Walther. Surrealistic Bohm trajectories. *Z. Naturforsch*, 47a:1175–1186, 1992.

[29] Berthold Georg Englert, Marlan O. Scully, Georg Süssmann, and Herbert Walther. Reply to comment on 'Surrealistic Bohm trajectories'. *Z. Naturforsch*, 48a:1263–1264, 1993.

[30] Steven Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987. C program code avaiable at `http://ect.bell-labs.com/who/sjf/`.

[31] Matthias Freyberger, Patrick Bardoff, Clemens Leichtle, Guenter Schrade, and Wolfgang Schleich. The art of measuring quantum states. *Phys. World*, 10(11):41–45, November 1997.

[32] Matthias Freyberger, Stefan H. Kienle, and Valery P. Yakovlev. Interferometric measurement of an atomic wave function. *Phys. Rev. A*, 56(1):195, 1997.

[33] Matthias Freyberger and Wolfgang P. Schleich. True vision of quantum state. *Nature*, 386:121–122, March 1997.

[34] Marco Genovese. Research on hidden variable theories: A review of recent progresses. *Phys. Rep.*, 413:319–396, 2005.

[35] J. E. Gentle. *Random Number Generation and Monte Carlo Methods*. Springer-Verlag, New York, 2nd edition, 2003.

[36] Allen Gersho and Robert M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic, Boston, 1992.

[37] B. J. Hiley, R. E. Callaghan, and O. J. E. Maroney. Quantum trajectories, real, surreal or an approximation to a deeper process? arXiv:quant-ph/0010020v2, November 2000.

[38] Basil J. Hiley and R. E. Callaghan. Delayed-choice experiments and the Bohm approach. *Phys. Scr.*, 74:336–348, 2006.

[39] Joseph O. Hirschfelder, Charles J. Goebel, and Ludwig W. Bruch. Quantized vortices around wavefunction nodes. ii. *J. Chem. Phys.*, 61:5456, 1974.

[40] Peter Holland. Uniqueness of paths in quantum mechanics. *Phys. Rev. A*, 60(6):4326, December 1999.

[41] Peter R. Holland. *The Quantum Theory of Motion*. Cambridge University Press, New York, 1993.

[42] Dragan Jukić and Rudolf Scitovski. Least squares fitting Gaussian type curve. *Appl. Math. Comp.*, 167:286–298, 2005.

[43] Simon Kochen and E. Specker. The problem of hidden variables in quantum mechanics. *J. Math. Mech.*, 17:59–87, 1968.

[44] Ch. Kurtsiefer and J. Mlynek. A 2-dimensional detector with high spatial and temporal resolution for metastable rare gas atoms. *Appl. Phys. B*, 64:85–90, 1997.

[45] Ch. Kurtsiefer, T. Pfau, and J. Mlynek. Measurement of the Wigner function of an ensemble of helium atoms. *Nature*, 386:150–153, March 1997.

[46] L. D. Landau and E. M. Lifshitz. *Quantum Mechanics (Non-relativistic Theory)*. Pergamon Press, New York, 3rd edition, 1977.

[47] T. G. Lewis. *Distribution Sampling for Computer Simulation*. Lexington Books, Massachusetts, 1975.

[48] S. P. Lloyd. Least Squares Quantization in PCM. *IEEE Trans. Inf. Theo.*, IT-28:129–137, 1982. Reprinted in *Quantization*, edited by P. F. Swaszek (Van Nostrand Reinhold, New York, 1985).

[49] Courtney L. Lopreore and Robert E. Wyatt. Quantum wave packet dynamics with trajectories. *Phys. Rev. Lett.*, 82:5190, 1999.

[50] J. Max. Quantizing for Minimum Distortion. *IEEE Trans. Inf. Theo.*, IT-6:7–12, 1960. Reprinted in *Quantization*, edited by P.F. Swaszek (Van Nostrand Reinhold, New York, 1985).

[51] N. David Mermin. Hidden variables and the two theorems of John Bell. *Rev. Mod. Phys.*, 65(3):803, July 1993.

[52] Atsuyuki Okabe, Barry Boots, and Kokichi Sugihara. *Spatial Tessellations*. John Wiley & Sons, New York, 1992.

[53] Oliver Passon. Why isn't every physicist a Bohmian?, 2005. arXiv:quant-ph/0412119v2.

[54] Wolfgang Pauli. In Gauthiers-Villiars et Cie, editor, *Reports on the 1927 Solvay Conference*, Paris, 1928.

[55] T. Pfau and Ch. Kurtsiefer. Partial reconstruction of the motional wigner function of an ensemble of helium atoms. *J. Mod. Optics*, 44(11/12):2551–2564, 1997.

[56] C. Philippidis, C. Dewdney, and B. J. Hiley. Quantum interference and the quantum potential. *Il Nuovo Cimento*, 52B:15–28, 1979.

[57] Bill Poirier. Optimal separable bases and series expansions. *Phys. Rev. A*, 56:120, 1997.

[58] J. O. Ramsay and B. W. Silverman. *Functional Data Analysis*. Springer, New York, 2nd edition, 2005.

[59] M. G. Raymer. Measuring the quantum mechanical wave function. *Cont. Phys.*, 38(5):343–355, 1997.

[60] Th. Richter. Reconstruction of the quantum state via position distribution and its time derivative. *Phys. Rev. A*, 54(3):2499, September 1996.

[61] Ángel S Sanz and F Borondo. A quantum trajectory description of decoherence. *Eur. Phys. J. D*, 44:319–326, 2007.

[62] Ángel S Sanz, F Borondo, and Salvador Miret-Artés. Causal trajectories description of atom diffraction by surfaces. *Phys. Rev. B*, 61(11):7743–7751, 2000.

[63] Ángel S Sanz and Salvador Miret-Artés. A causal look into the quantum talbot effect. *J. Chem. Phys.*, 126:234106, 2007.

[64] Marlan O. Scully. Do Bohm trajectories always provide a trustworthy physical picture of particle motion? *Phys. Scr.*, T76:41–46, 1998.

[65] Richard Steiner, David Newell, and Edwin Williams. Details of the 1998 Watt balance experiment determining the Planck constant. *J. Res. Natl. Stand. Technol.*, 110:1–26, 2005.

[66] W. Struyve, W. De Baere, J. De Neve, and S. De Weirdt. On the uniqueness of paths for spin-0 and spin-1 quantum. *Phys. Lett. A*, 322:84–95, 2004.

[67] Michael E. Tarter and Michael D. Lock. *Model-Free Curve Estimation*. Chapman and Hall, New York, 1993.

[68] John von Neumann. *Mathematische Grundlagen der Quantummechaniek*. Springer, Berlin, 1932. Mathematical Foundations of Quantum Mechanics, Princeton University Press, New Jersey, 1955.

[69] John von Neumann. Various techniques used in connection with random digits. *Nat. Bur. Standards*, 12:36–38, 1951.

[70] S. Weinberg. Unpublished email correspondence to Sheldon Goldstein.

[71] Edwin R. Williams, Richard L. Steiner, David B. Newell, and Paul T. Olsen. Accurate Measurement of the Planck Constant. *Phys. Rev. Lett.*, 81(12):2404, September 1998.

[72] Robert E. Wyatt. *Quantum Dynamics with Trajectories: Introduction to Quantum Hydrodynamics*. Springer, New York, 2005.

Vita

Timothy Michael Coffey was born in Chicago, Illinois on 12 February 1970, the son of Virginia and Timothy Coffey. He began his college studies at Reed College in 1988. Two years later he left school, and began working in Reno, Nevada as a casino dealer. While in Reno he received a US Patent for a video game screen divider. In 1997 he moved back to Portland, Oregon and began working for a national non-profit organization. In 2002, he finally returned to Reed College and received the Bachelor of Arts degree in Physics in 2004.

In August 2004, he entered the doctoral program in the Department of Physics at The University of Texas at Austin, where he was employed as a Teaching Assistant and an Assistant Instructor, winning a Distinguished Teaching Award in 2009. Fall of 2006 he began working with Dr. William C. Schieve, and then in fall 2008 began to work additionally with Dr. Robert E. Wyatt.

Permanent address: 7504 Cayenne Lane
Austin, Texas 78741

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.