Chapter 16

Quantum Systems

6 July 2005

We discuss numerical solutions of the time-independent and time-dependent Schrödinger equation and describe several Monte Carlo methods for estimating the ground state of quantum systems.

16.1 Introduction

So far we have simulated the microscopic behavior of physical systems using Monte Carlo methods and molecular dynamics. In the latter method, the classical trajectory (the position and momentum) of each particle is calculated as a function of time. However, in quantum systems the position and momentum of a particle cannot be specified simultaneously. Because the description of microscopic particles is intrinsically quantum mechanical, we cannot directly *simulate* their trajectories on a computer (see Feynman).

Quantum mechanics does allow us to *analyze* probabilities, although there are difficulties associated with such an analysis. Consider a simple probabilistic system described by the one-dimensional diffusion equation (see Section 7.2)

$$\frac{\partial P(x,t)}{\partial t} = D\frac{\partial^2 P(x,t)}{\partial x^2}, \tag{16.1}$$

where $P(x,t)$ is the probability density of a particle being at position x at time t. One way to convert (16.1) to a difference equation and obtain a numerical solution for $P(x,t)$ is to make x and t discrete variables. Suppose we choose a mesh size for x such that the probability is given at p values of x. If we choose p to be order 10^3, a straightforward calculation of $P(x,t)$ would require approximately 10^3 data points for each value of t. In contrast, the corresponding calculation of the dynamics of a single particle based on Newton's second law would require one data point.

The limitations of the direct computational approach become even more apparent if there are many degrees of freedom. For example, for N particles in one dimension, we would have to

calculate the probability $P(x_1, x_2, \ldots, x_N, t)$, where x_i is the position of particle i. Because we need to choose a mesh of p points for each x_i, we need to specify N^p values at each time t. For the same level of precision, p will be proportional to the length of the system (for particles confined to one dimension). Consequently, the calculation time and memory requirements grow exponentially with the length of the system. For example, for 10 particles on a mesh of 100 points, we would need to store 10^{100} numbers to represent P, which is already much more than any computer today can store. In two and three dimensions the growth is even faster.

Although the direct computational approach is limited to systems with only a few degrees of freedom, the simplicity of this approach will aid our understanding of the behavior of quantum systems. After a summary of the general features of quantum mechanical systems in Section 16.2, we consider this approach to solving the time-independent Schrödinger equation in Section 16.3 and 16.4. In Section 16.5, we use a half-step algorithm to generate wave packet solutions to the time-dependent Schrödinger equation.

Because we have already learned that the diffusion equation (16.1) can be formulated as a random walk problem, it might not surprise you that Schrödinger's equation can be analyzed in a similar way. Monte Carlo methods are introduced in Section 16.7 to obtain variational solutions of the ground state. We introduce quantum Monte Carlo methods in Section 16.8 and discuss more sophisticated quantum Monte Carlo methods in Sections 16.9 and 16.10.

16.2 Review of Quantum Theory

For simplicity, we consider a one-dimensional, nonrelativistic quantum system consisting of one particle. The state of the system is completely characterized by the position space wave function $\Psi(x,t)$, which is interpreted as a probability amplitude. The probability $P(x,t)\,\Delta x$ of the particle being in a "volume" element Δx centered about the position x at time t is equal to

$$P(x,t)\,\Delta x = |\Psi(x,t)|^2 \Delta x, \tag{16.2}$$

where $|\Psi(x,t)|^2 = \Psi(x,t)\Psi^*(x,t)$ and $\Psi^*(x,t)$ is the complex conjugate of $\Psi(x,t)$. This interpretation of $\Psi(x,t)$ requires the use of normalized wave functions such that

$$\int_{-\infty}^{\infty} \Psi^*(x,t)\Psi(x,t)\,dx = 1. \tag{16.3}$$

If the particle is subjected to the influence of a potential energy function $V(x,t)$, the evolution of $\Psi(x,t)$ is given by the time-dependent Schrödinger equation

$$i\hbar\frac{\partial\Psi(x,t)}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2\Psi(x,t)}{\partial x^2} + V(x,t)\Psi(x,t), \tag{16.4}$$

where m is the mass of the particle and $\hbar$ is Planck's constant divided by 2π.

Physically measurable quantities, such as the momentum, have corresponding operators. The expectation or average value of an observable A is given by

$$\langle A\rangle = \int \Psi^*(x,t)\hat{A}\Psi(x,t)\,dx, \tag{16.5}$$

where $\hat{A}$ is the operator corresponding to the measurable quantity A. For example, the momentum operator corresponding to the linear momentum p is $\hat{p} = -i\hbar\partial/\partial x$ in position space.

If the potential energy function is independent of time, we can obtain solutions of (16.4) of the form

$$\Psi(x,t) = \phi(x)\, e^{-iEt/\hbar}. \tag{16.6}$$

A particle in the state (16.6) has a well-defined energy E. If we substitute (16.6) into (16.4), we obtain the time-independent Schrödinger equation

$$-\frac{\hbar^2}{2m}\frac{d^2\phi(x)}{dx^2} + V(x)\,\phi(x) = E\,\phi(x). \tag{16.7}$$

Note that $\phi(x)$ is an *eigenstate* of the Hamiltonian operator,

$$\hat{H} = -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + V(x), \tag{16.8}$$

with the *eigenvalue* E. That is,

$$\hat{H}\,\phi(x) = E\,\phi(x). \tag{16.9}$$

In general, there are many eigenstates ϕ_n, each with eigenvalue, E_n, that satisfy (16.9) and the boundary conditions imposed on the eigenstates by physical considerations.

The general form of $\Psi(x,t)$ can be expressed as a superposition of the eigenstates of the operator corresponding to any physical observable. For example, if $\hat{H}$ is independent of time, we can write

$$\Psi(x,t) = \sum_n c_n\,\phi_n(x)\, e^{-iE_n t/\hbar}, \tag{16.10}$$

where Σ represents a sum over the discrete states and an integral over the continuum states. The coefficients c_n in (16.10) can be determined from the value of $\Psi(x,t)$ at any time t. For example, if we know $\Psi(x,t=0)$, we can use the orthonormality property of the eigenstates of any physical operator to obtain

$$c_n = \int \phi_n^*(x)\Psi(x,0)\,dx. \tag{16.11}$$

The coefficient c_n can be interpreted as the probability amplitude of a measurement of the total energy yielding a particular value E_n.

There are three steps needed to solve (16.7) numerically. The first is to integrate (16.7) for any given value of the energy, E, in a way similar to the approach we have used for numerically solving other ordinary differential equations. This approach will usually not satisfy the boundary conditions. The second step is to find the particular values of E that lead to solutions that satisfy the boundary conditions. Finally, we need to normalize the eigenstate wave function using (16.3) so that we can interpret the eigenstate as a probability amplitude.

We first discuss the solution of (16.7) without imposing any boundary conditions by treating the solution to (16.7) as an initial value problem for the wave function and its derivative at some value of x for a given value of E. We will use these solutions to develop our intuition about the behavior of one-dimensional solutions to the Schrödinger equation.

To use an ODE solver, we express the wave function rate in terms of the independent variable, x,

$$\frac{d\phi}{dx} = \phi' \tag{16.12a}$$

$$\frac{d\phi'}{dx} = -\frac{2m}{\hbar^2}[E - V(x)]\phi \tag{16.12b}$$

$$\frac{dx}{dx} = 1. \tag{16.12c}$$

Because the time-independent Schrödinger equation is a second-order differential equation, two initial conditions must be specified to obtain a solution. For simplicity, we first assume that the wave function is zero at the starting point, `xmin`, and the derivative is nonzero. We also assume that the range of values of x is finite and divide this range into intervals of width Δx. We initially consider potential energy functions $V(x)$ such that $V(x) = 0$ for $x < 0$; $V(x)$ changes abruptly at $x = 0$ to V_0, the value of the `stepHeight` parameter. An implementation of the numerical solution of (16.12) is shown in Listing 16.1.

Listing 16.1: The `Schroedinger` class models the one-dimensional time independent Schrödinger equation.

```
package org.opensourcephysics.sip.ch16;
import org.opensourcephysics.numerics.*;

public class Schroedinger implements ODE {
   double energy = 0;
   double[] phi;
   double[] x;
   double xmin, xmax;                  // range of values of x
   double[] state = new double[3]; // state = phi, dphi/dx, x
   ODESolver solver = new RK45MultiStep(this);
   double stepHeight = 0;
   int numberOfPoints;

   public void initialize() {
      phi = new double[numberOfPoints];
      x = new double[numberOfPoints];
      double dx = (xmax-xmin)/(numberOfPoints-1);
      solver.setStepSize(dx);
   }

   void solve() {
      for(int i = 0;i<numberOfPoints;i++) { // zeros wavefunction
         phi[i] = 0;
      }
      state[0] = 0;     // initial phi
      state[1] = 1.0;  // nonzero initial dphi/dx
      state[2] = xmin; // initial value of x
      for(int i = 0;i<numberOfPoints;i++) {
         phi[i] = state[0];              // stores wavefunction
```

```
            x[i] = state[2];
            solver.step();                    // steps Schroedinger equation
            if(Math.abs(state[0])>1.0e9) {    // checks for diverging solution
               break;                         // leave the loop
            }
         }
      }

      public double[] getState() {
         return state;
      }

      public void getRate(double[] state, double[] rate) {
         rate[0] = state[1];
         rate[1] = 2.0*(-energy+evaluatePotential(state[2]))*state[0];
         rate[2] = 1.0;
      }

      public double evaluatePotential(double x) { // potential is nonzero for x > 0
         if(x<0) {
            return 0;
         } else {
            return stepHeight;
         }
      }
   }
```

The `solve` method initializes the wave function and position arrays and sets the initial value of $d\phi/dx$ to an arbitrary nonzero value of unity. A loop is then used to compute values of ϕ until the solution diverges or until $x \geq$ `xmax`.

`SchroedingerApp` in Listing 16.2 produces a graphical view of $\phi(x)$. We will use this program in Problem 16.1 to study the behavior of the solution as we vary the height of the potential step.

Listing 16.2: `SchroedingerApp` solves the one-dimensional time-independent Schrödinger equation for a given energy.

```
package org.opensourcephysics.sip.ch16;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.display.*;
import org.opensourcephysics.frames.*;

public class SchroedingerApp extends AbstractCalculation {
   PlotFrame frame = new PlotFrame("x", "phi", "Wave function");
   Schroedinger schroedinger = new Schroedinger();

   public SchroedingerApp() {
      frame.setConnected(0, true);
      frame.setMarkerShape(0, Dataset.NO_MARKER);
   }
```

```
    public void calculate() {
        schroedinger.xmin = control.getDouble("xmin");
        schroedinger.xmax = control.getDouble("xmax");
        schroedinger.stepHeight = control.getDouble("step height at x = 0");
        schroedinger.numberOfPoints = control.getInt("number of points");
        schroedinger.energy = control.getDouble("energy");
        schroedinger.initialize();
        schroedinger.solve();
        frame.append(0, schroedinger.x, schroedinger.phi);
    }

    public void reset() {
        control.setValue("xmin", -5);
        control.setValue("xmax", 5);
        control.setValue("step height at x = 0", 1);
        control.setValue("number of points", 500);
        control.setValue("energy", 1);
    }

    public static void main(String[] args) {
        CalculationControl.createApp(new SchroedingerApp(), args);
    }
}
```

Problem 16.1. Numerical solution of the time-independent Schrödinger equation

a. Sketch your guess for $\phi(x)$ for a potential step height of $V_0 = 3$ and energies $E = 1$, 2, 3, 4, and 5.

b. Choose `xmin = -10` and `xmax = 10`, and run `SchroedingerApp` with the parameters given in part (a). How well do your predictions match the numerical solution? Is there any discontinuity in ϕ or in the derivative $d\phi/dx$ at $x = 0$? Describe the wave function for both $x < 0$ and $x > 0$. Why does the wave function have a larger oscillatory amplitude when $x > 0$ than when $x < 0$ if the energy is greater than the potential step height?

c. Describe the behavior of the wave function as the energy approaches the potential step height. Consider E in the range 2.5 to 3.5 in steps of 0.1.

d. Repeat part (b) with the initial condition $\phi = 1$ and $d\phi/dx = 0$. Describe the differences, if any, in $\phi(x)$.

Problem 16.1 demonstrates that the nature of the solution of (16.7) changes dramatically depending on the relative values of the energy E and the potential energy. If E is greater than V_0, the wave function is oscillatory, whereas if E is less than or equal to V_0, the wave function grows exponentially. The differential equation solver may fail if the difference between the potential energy and E is too large. There also is an exponentially decaying solution in the region where $E < V_0$, but this solution is difficult to detect.

Problem 16.2. Analytic solutions of the time-independent Schrödinger equation

a. Find the analytic solution to (16.7) for the step potential for the cases: $E > V_0$, $E < V_0$, and $E = V_0$. We will use units such that $m = \hbar = 1$ in all the problems in this chapter.

b. Run `SchroedingerApp` for the three cases to obtain the numerical solution of (16.7). When the numercial solution shows spatial oscillations in a region of space, estimate the wavelength of the oscillations and compare your numerical solution to the analytical results. When the numerical solution shows exponential decay as a function of position, estimate the decay rate and compare your numerical solution with the analytic solution.

The solutions that we have obtained so far do not satisfy any condition other than that they solve (16.12). We have plotted only a portion of the wave function and the solutions can be extended by increasing the number of points and the range of x over which the computation is performed. Physically, these solutions are unrealistic because they cannot be normalized over all of space. The normalization problem can be solved by using a linear combination of energy eigenstates (16.10) with different values of E. This combination is called a *wavepacket*.

Although we used a fourth-order algorithm in Listing 16.1, simpler algorithms can be used. Recall that the solution of (16.7) with $V(x) = 0$ can be expressed as a linear combination of sine and cosine functions. The oscillatory nature of this solution leads us to expect that the Euler-Cromer algorithm introduced in Chapter 3 will yield satisfactory results.

16.3 Bound State Solutions

We first consider potentials for which a particle is confined to a specific region of space. Such a potential is known as the infinite square well and is described by

$$V(x) = \begin{cases} 0 & \text{for } |x| \le a \\ \infty & \text{for } |x| > a \end{cases} \tag{16.13}$$

For this potential, an acceptable solution of (16.7) must vanish at the boundaries of the well. We will find that the eigenstates, $\phi_n(x)$, can satisfy these boundary conditions only for specific values of the energy E_n.

Problem 16.3. The infinite square well

a. Show analytically that the energy eigenvalues of the infinite square well are given by $E_n = n^2\pi^2\hbar^2/8ma^2$, where n is a positive integer. Also show that the normalized eigenstates have the form

$$\phi_n(x) = \frac{1}{\sqrt{a}} \cos\frac{n\pi x}{2a} \qquad n = 1, 3, \ldots \qquad \text{(even parity)} \tag{16.14a}$$

$$\phi_n(x) = \frac{1}{\sqrt{a}} \sin\frac{n\pi x}{2a}. \qquad n = 2, 4, \ldots \qquad \text{(odd parity)} \tag{16.14b}$$

What is the parity of the ground state solution?

b. We can solve (16.7) numerically for the infinite square well by setting `stepHeight` $= 0$. `xmin` $= -a$, and `xmax` $= +a$ in `SchroedingerApp` and requiring that $\phi(x = +a) = 0$. What is the condition for $\phi(x = -a)$ in the program? Choose $a = 1$ and calculate the first four energy eigenvalues exactly using `SchroedingerApp`. Do the numerical and analytical solutions match? Do the solutions satisfy the boundary conditions exactly? Are your numerical solutions normalized?

Problem 16.4. Bound state solutions of the time-independent Schrödinger equation

a. Consider the potential energy function defined by

$$V(x) = \begin{cases} 0 & \text{for } -a \le x \le 0 \\ V_0 & \text{for } 0 < x \le a \\ \infty & \text{for } |x| > a. \end{cases} \tag{16.15}$$

As for the infinite square well, the eigenfunction is confined between infinite potential barriers at $x = \pm a$. In addition, there is a step potential at $x = 0$. Choose $a = 5$ and $V_0 = 1$ and run `SchroedingerApp` with an energy of $E = 0.15$. Repeat with an energy of $E = 0.16$. Why can you conclude that an energy eigenvalue is bracketed by these two values?

b. Choose a strategy for determining the value of E such that the boundary conditions at $x = +a$ are satisfied. Determine the energy eigenvalue to four decimal places. Does your answer depend on the number of points at which the wave function is computed?

c. Repeat the above procedure starting with energy values of 0.58 and 0.59 and find the energy eigenvalue of the second bound state.

If you were persistent in doing all of Problem 16.4, you would have discovered two energy eigenvalues, 0.1505 and 0.5857.

The procedure we used is known as the *shooting* algorithm. The allowed eigenvalues are imposed by the requirement that $\phi_n(x) \to 0$ at the boundaries. Although the shooting algorithm usually yields an eigenvalue solution, we often wish to find specific eigenvalues, such as the eigenvalue $E = 1.1195$ corresponding to the third excited state for the potential in (16.15). Because the energy of a wave function increases as the wavelength decreases, we can order the energy eigenvalues by counting the number of times the corresponding eigenstate crosses the x-axis, that is, by the number of nodes. The ground state eigenstate has no nodes. Why? Why can we order the eigenvalues by the number of nodes? The number of nodes can be used to narrow the energy bracket in the shooting algorithm. For example, if we are searching for the third energy eigenvalue and we observe 5 nodes, then the energy is too large. To find a specific quantum state, we automate the shooting method as follows:

1. Choose a value of the energy E and count the number of nodes.
2. Increase E and repeat step 1 until the number of nodes is equal to the desired number.

3. Decrease E and repeat step 1 until the number of nodes is one less than the desired number. The desired value of the energy eigenvalue is now bracketed. We can further narrow the energy by doing the following:
4. Set the energy to the bracket midpoint.
5. Initialize $\phi(x)$ at the left boundary and iterate $\phi(x)$ toward increasing x until ϕ diverges or until the right boundary is reached.
6. If the quantum number is even (odd) and the last value of $\phi(x)$ in step 4 is negative (positive), then the trial value of E is too large.
7. If the quantum number is even (odd) and the last value of $\phi(x)$ in step 4 is positive (negative), then the trial value of E is too small.
8. Repeat steps 2–7 until the wave function satisfies the right-hand boundary condition to an acceptable tolerance. This procedure is known as a binary search because every repetition decreases the energy bracket by a factor of two.

Problem 16.5 asks you to write a program that finds specific eigenvalues using this procedure.

Problem 16.5. Shooting algorithm

a. Modify `SchroedingerApp` to find the eigenvalue associated with a given number of nodes. How is the number of nodes related to the quantum number? Test your program for the infinite square well. What is the value of Δx need to determine E_1 to two decimal places? Three decimal places?

b. Add a method to normalize ϕ. Normalize and display the first five eigenstates.

c. Find the first five eigenstates and eigenvalues for the potential in (16.15). with $a = 1$ and $V - 0 = 1$.

d. Does your result for E_1 depend on the starting value of $d\phi/dx$?

Problem 16.6. Perturbation of the infinite square well

a. Determine the effect of a small perturbation on the eigenstates and eigenvalues of the infinite square well. Place a small rectangular bump of half-width b and height V_b symmetrically about $x = 0$ (see Fig. 16.1). Choose $b \ll a$ and determine how the ground state energy and eigenstate change with V_b and b. What is the relative change in the ground state energy for $V_b = 10$, $b = 0.1$ and $V_b = 20$, $b = 0.1$? with $a = 1$. Let ϕ_0 denote the ground state eigenstate for $b = 0$ and let ϕ_b denote the ground state eigenstate for $b \neq 0$. Compute the value of the overlap integral

$$\int_0^a \phi_b(x)\phi_0(x)\,dx. \tag{16.16}$$

This integral would be unity if the perturbation were not present (and the eigenstate was properly normalized). How is the change in the overlap integral related to the relative change in the energy eigenvalue?

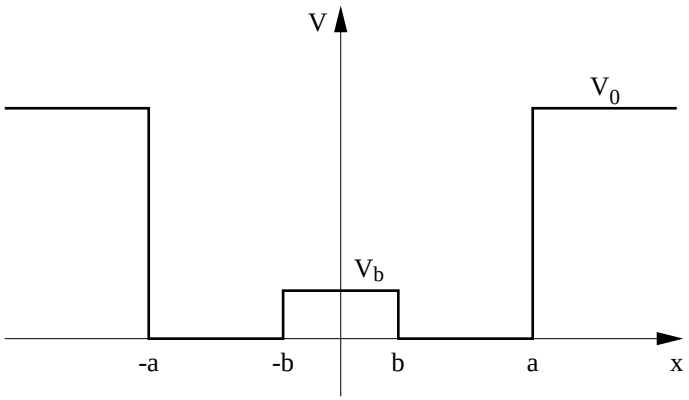


Figure 16.1: An infinite square well with a potential bump of height V_b in the middle.

b. Compute the ground state energy for $V_b = 20$ and $b = 0.05$. How does the value of E_1 compare to that found in part (a) for $V_b = 10$ and $b = 0.1$?

Because numerical solutions to the Schrödinger equation grow exponentially if $V(x) - E > 0$, it may not be possible to obtain a numerical solution for $\phi(x)$ that satisfies the boundary conditions if $V(x) - E$ is large over an extended region of space. The reason is that energy can be specified and ϕ can be computed only to finite accuracy. Problem 16.7 shows that we can sometimes solve this problem using simpler boundary conditions if the potential is symmetric. In this case,

$$V(x) = V(-x), \tag{16.17}$$

and $\phi(x)$ can be chosen to have definite parity. For even parity solutions, $\phi(-x) = \phi(x)$; odd parity solutions satisfy $\phi(-x) = -\phi(x)$. The definite parity of $\phi(x)$ allows us to specify either ϕ or ϕ' at $x = 0$. Hence, the parity of ϕ determines one of the boundary conditions. For simplicity, choose $\phi(0) = 1$ and $\phi'(0) = 0$ for even parity solutions, and $\phi(0) = 0$ and $\phi'(0) = 1$ for odd parity solutions.

Problem 16.7. Symmetric potentials

a. Modify `Schroedinger` to make use of symmetric potential boundary conditions for the harmonic oscillator:

$$V(x) = \frac{1}{2}x^2. \tag{16.18}$$

Start the solution at $x = 0$ using appropriate conditions for even and odd quantum numbers and find the first four energy eigenvalues such that the wave function approaches zero for large values of x. Because the computed $\phi(x)$ will diverge for sufficiently large x, we seek values of the energy such that a small decrease in E causes the wave function to diverge in one direction, and a small increase causes the wave function to diverge in the opposite direction. Initially choose `xmax` $= 5$, so that the classically forbidden region is sufficiently large so that $\phi(x)$ can decay to zero for the first few eigenstates. Increase `xmax` if necessary for the higher energy eigenvalues. Is there any pattern in the values of the energy eignevalues you find?

b. Repeat part (a) for the linear potential $V(x) = |x|$. Describe the differences between your results for this potential and for the harmonic oscillator potential. The quantum mechanical treatment of the linear potential can be used to model the energy spectrum of a bound quark-antiquark system known as quarkonium.

c. Obtain a numerical solution of the anharmonic oscillator, $V(x) = \frac{1}{2}x^2 + bx^4$. In this case there are no analytical solutions and numerical solutions are necessary for large values of b. How do the ground state energy and eigenstate depend on b for small b?

Problem 16.8. Finite square well

The finite square well potential is given by

$$V(x) = \begin{cases} 0 & \text{for } |x| \leq a \\ V_0 & \text{for } |x| > a \end{cases} \tag{16.19}$$

The input parameters are the well depth, V_0, and the half-width of the well, a.

a. Choose $V_0 = 10$ and $a = 1$. How do you expect the value of the ground state energy to compare to its corresponding value for the infinite square well? Compute the ground state eigenvalue and eigenstate by determining a value of E such that $\phi(x)$ has no nodes and is approximately zero for large x. (See Problem (16.7a) for the procedure for finding the eigenvalues.)

b. Because the well depth is finite, $\phi(x)$ is nonzero in the classically forbidden region for which $E < V_0$ and $x > |a|$. Define the penetration distance as the distance from $x = a$ to a point where ϕ is $\sim 1/e \approx 0.37$ of its value at $x = a$. Determine the qualitative dependence of the penetration distance on the magnitude of V_0.

c. What is the total number of bound excited states? Why is the total number of bound states finite?

As we have found, it is difficult to find bound state solutions of the time-independent Schrödinger equation because the exponential solution allows numerical errors to dominate when $V(x) - E > 0$ is large. Because we want to easily generate eigenstates in subsequent sections, we have written a general-purpose eigenstate solver that examines the maxima and minima of the solution as well as the nodes to determine the eigenstate's quantum number. The code for the `Eigenstate` class is in the ch16 package. The `EigenstateApp` target class shows how the `Eigenstate` class is used.

Listing 16.3: The `EigenstateApp` program tests the `Eigenstate` class.

```
package org.opensourcephysics.sip.ch16;
import org.opensourcephysics.frames.PlotFrame;
import org.opensourcephysics.numerics.Function;

public class EigenstateApp {
   public static void main(String[] args) {
      PlotFrame drawingFrame = new PlotFrame("x", "|phi|", "eigenstate");
      int numberOfPoints = 300;
      double
         xmin = -5, xmax = +5;
```

```
        Eigenstate eigenstate = new Eigenstate(new Potential(), numberOfPoints, xmin, xmax);
        int n = 3; // quantum number
        double[] phi = eigenstate.getEigenstate(n);
        double[] x = eigenstate.getXCoordinates();
        if(eigenstate.getErrorCode()==Eigenstate.NO_ERROR) {
           drawingFrame.setMessage("energy = "+eigenstate.energy);
        } else {
           drawingFrame.setMessage("eigenvalue did not converge");
        }
        drawingFrame.append(0, x, phi);
        drawingFrame.setVisible(true);
        drawingFrame.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
     }
}
class Potential implements Function {
     public double evaluate(double x) {
        return(x*x)/2;
     }
}
```

The `getEigenstate` method in the `Eigenstate` class computes the eigenstate for the specified quantum number and returns a zeroed wave function if the algorithm does not converge. We test the validity of the `Eigenstate` class in Problem 16.9.

Problem 16.9. The `Eigenstate` class

a. Examine the code of the `Eigenstate` class. What "trick" is used to handle the divergence in the forbidden region of deep wells?

b. Write a class that displays the eigenstates of the simple harmonic oscillator using the `Calculation` interface. Include input parameters that allow the user to vary the principal quantum number and the number of points.

c. Use a spatial grid of 300 points with $-5 < x < 5$ and compare the known analytic solution for the simple harmonic oscillator eigenstates to the numerical solution for the lowest three energy eigenstates. What is the largest energy eigenvalue that can be computed to an accuracy of 1%? What causes the decreasing accuracy for larger quantum numbers? What if the domain is increased to $-50 < x < 50$?

d. Describe the conditions under which the `Eigenstate` class fails and demonstrate this failure. Improve the `Eigenstate` class to handle at least one failure mode.

16.4 Time Development of Eigenstate Superpositions

If the Hamiltonian is independent of time, the time development of the wave function, $\Psi(x,t)$, can be expressed as a linear superposition of energy eigenstates, $\phi_n(x)$, with eigenvalue E_n.

$$\Psi(x,t) = \sum_n c_n \, \phi_n(x) \, e^{-iE_n t/\hbar}. \tag{16.20}$$

To understand the time dependence of $\Psi(x,t)$, we begin by studying superpositions of analytic solutions. The static `getEigenstate` method in the `BoxEigenstate` class generates these solutions for the infinite square well.

Listing 16.4: The `BoxEigenstate` class generates analytic stationary state solutions for the infinite square well.

```
package org.opensourcephysics.sip.ch16;
public class BoxEigenstate {
   static double a = 1; // length of box

   private BoxEigenstate() {
      // prohibit instantiation because all methods are static
   }

   static double[] getEigenstate(int n, int numberOfPoints) {
      double[] phi = new double[numberOfPoints];
      n++; // quantum number
      double norm = Math.sqrt(2/a);
      for(int i = 0;i<numberOfPoints;i++) {
         phi[i] = norm*Math.sin((n*Math.PI*i)/(numberOfPoints-1));
      }
      return phi;
   }

   static double getEigenvalue(int n) {
      n++;
      return(n*n*Math.PI*Math.PI)/2/a/a; // hbar = 1, mass = 1
   }
}
```

To visualize the evolution of $\Psi(x,t)$ in (16.20), we define a class that stores the energy eigenstates, $\phi_n(x)$, the real and imaginary parts of the expansion coefficients, c_n, and the eigenvalues, E_n. As the system evolves, the eigenstates are added together as in (16.20) using the expansion coefficients. The `BoxSuperposition` class shown in Listing 16.5 creates such a wave function for the infinite square well. Later we will modify this class to study other potentials.

Listing 16.5: The `BoxSuperposition` class models the time dependence of the wave function of an infinite square well using a superposition of eigenstates.

```
package org.opensourcephysics.sip.ch16;
public class BoxSuperposition {
   double[] realCoef;
   double[] imagCoef;
   double[][] states;     // eigenfunctions
   double[] eigenvalues; // eigenvalues
   double[] x, realPsi, imagPsi;
   double[] zeroArray;

   public BoxSuperposition(int numberOfPoints, double[] realCoef, double[] imagCoef) {
      if(realCoef.length!=imagCoef.length) {
```

```
            throw new IllegalArgumentException("Real and imaginary coefficients must have equal
        }
        this.realCoef = realCoef;
        this.imagCoef = imagCoef;
        int nstates = realCoef.length;
        // delay allocation of arrays for eigenstates
        states = new double[nstates][];     // eigenfunctions
        eigenvalues = new double[nstates]; // eigenvalues
        realPsi = new double[numberOfPoints];
        imagPsi = new double[numberOfPoints];
        zeroArray = new double[numberOfPoints];
        x = new double[numberOfPoints];
        double dx = BoxEigenstate.a/(numberOfPoints-1);
        double xo = 0;
        for(int j = 0, n = numberOfPoints;j<n;j++) {
           x[j] = xo;
           xo += dx;
        }
        for(int n = 0;n<nstates;n++) {
           states[n] = BoxEigenstate.getEigenstate(n, numberOfPoints);
           eigenvalues[n] = BoxEigenstate.getEigenvalue(n);
        }
        update(0); // compute the superpositon at t = 0
     }

     void update(double time) {
        // set real and imaginary parts of wave function to zero
        System.arraycopy(zeroArray, 0, realPsi, 0, realPsi.length);
        System.arraycopy(zeroArray, 0, imagPsi, 0, imagPsi.length);
        for(int i = 0, nstates = realCoef.length;i<nstates;i++) {
           double[] phi = states[i];
           double re = realCoef[i];
           double im = imagCoef[i];
           double sin = Math.sin(time*eigenvalues[i]);
           double cos = Math.cos(time*eigenvalues[i]);
           for(int j = 1, n = phi.length-1;j<n;j++) {
              realPsi[j] += (re*cos-im*sin)*phi[j];
              imagPsi[j] += (im*cos+re*sin)*phi[j];
           }
        }
     }
}
```

The `BoxSuperpositionApp` class in Listing 16.6 implements the eigenstate superposition and displays the wave function by extending the `AbstractAnimation` class and implementing the `doStep` method.

Listing 16.6: `BoxSuperpositionApp` shows the evolution of a particle in a box.

```
package org.opensourcephysics.sip.ch16;
import org.opensourcephysics.controls.*;
```

```
import org.opensourcephysics.frames.ComplexPlotFrame;

public class BoxSuperpositionApp extends AbstractSimulation {
   ComplexPlotFrame psiFrame = new ComplexPlotFrame("x", "|Psi|", "Time dependent wave functi
   BoxSuperposition superposition;
   double time, dt;

   public BoxSuperpositionApp() {
      psiFrame.limitAutoscaleY(-1, 1);
   }

   public void initialize() {
      time = 0;
      psiFrame.setMessage("t = "+decimalFormat.format(time));
      dt = control.getDouble("dt");
      double[] re = (double[]) control.getObject("real coef");
      double[] im = (double[]) control.getObject("imag coef");
      int numberOfPoints = control.getInt("number of points");
      superposition = new BoxSuperposition(numberOfPoints, re, im);
      psiFrame.append(superposition.x, superposition.realPsi, superposition.imagPsi);
   }

   public void doStep() {
      time += dt;
      superposition.update(time);
      psiFrame.clearData();
      psiFrame.append(superposition.x, superposition.realPsi, superposition.imagPsi);
      psiFrame.setMessage("t = "+decimalFormat.format(time));
   }

   public void reset() {
      control.setValue("dt", 0.005);
      control.setValue("real coef", new double[] {0.707, 0, 0.707});
      control.setValue("imag coef", new double[] {0, 0, 0});
      control.setValue("number of points", 50);
      initialize();
   }

   public static void main(String[] args) {
      SimulationControl.createApp(new BoxSuperpositionApp());
   }
}
```

Because wave functions have real and imaginary components, the `BoxSuperpositionApp` class uses a `ComplexPlotFrame` for plotting. The `ComplexPlotFrame` renders data using an envelope whose height is proportional to the magnitude and the region between the envelope is colored from red to blue to show the phase. A more traditional plotting style showing the real and imaginary parts of the wave function is available from the frame's Tools menu. (Also see Appendix 16A.) We use `BoxSuperpositionApp` to study the periodicity of the wave function in Problems 16.10 and

16.11.

Problem 16.10. Time-dependent wavefunction for the infinite square well

a. Add a second visualization to the `BoxSuperpositionApp` class that displays the probability density $\Psi(x,t)$.

b. Change the coefficient array so that the particle is in the ground state. Show that the wave function changes in time, but that the probability density does not. At what times does the ground state wave function return to its initial condition? Find the corresponding times for the first and second excited states.

c. Choose the coefficient array so that the particle is in a 50:50 superposition of the ground state and the first excited state. At what times does the wave function return to its initial condition? After what time does the probability density return to its initial condition?

d. Change the coefficient array so that the particle is in a 50:50 superposition of the first and second excited states. After what time does the wave function return to its initial condition? After what time does the probability density return to its initial condition?

e. Will the initial wave function always revive, that is, return to its initial condition? Explain.

Problem 16.11. Time-dependent wavefunction for the simple harmonic oscillator

a. Modify `BoxSuperpositionApp` and `BoxSuperposition` to superimpose the eigenstates of the simple harmonic oscillator using the `Eigenstate` class to compute the eigenstates. What is the period of the ground state and the first excited state wave functions and the probability density?

b. Change the coefficient array so that the particle is in a 50:50 superposition of the ground state and the first excited state. At what times does the wave function return to its initial condition? At what times does the probability density return to its initial condition? Compare these times with the period of the classical oscillator.

c. Repeat part (b) for a 50:50 superposition of the first and second excited states.

Problem 16.12. Linear potential

Does the linear potential, $V(x) = |x|$, exhibit periodicity if the particle is in a superposition state? Test your hypothesis using numerical solutions to the Schrödinger equation.

As we have seen, the evolution of an arbitrary wave function can be found by expanding the initial state in terms of the energy eigenstates. From the orthogonality property of eigenstates, it is easy to show that

$$c_n = \int_{-\infty}^{\infty} \phi_n^*(x)\Psi(x,0)dx. \tag{16.21}$$

This operation is known as a projection of Ψ onto ϕ_n.

Problem 16.13. Projections

a. Add a `projection` method to the `BoxSuperpositionApp` class using the signature:

```
double[] projection(int n, double[] realPhi, double[] imagPhi)
```

The projection method's arguments are the quantum number, the real component of the wave function, and the imaginary component of the wave function. The method returns a two component array containing the real and imaginary parts of the projection of the wave function on the nth eigenstate.

b. Test your projection method by projecting an eigenstate onto another eigenstate. That is, verify the orthogonality condition,

$$\delta_{nm} = \int_{-\infty}^{\infty} \phi_m(x)\phi_n(x)dx. \tag{16.22}$$

c. Compute the expansion coefficients for a particle in a box using the following initial Gaussian wave function:

$$\Psi(x,0) = e^{-64x^2}. \tag{16.23}$$

Assume a box width $a = 1$. Plot the amplitude of the resulting coefficients as a function of the quantum number n. How does the shape of this plot depend on the width of the Gaussian wave function?

d. Use the coefficients from part (c) to determine the evolution of the wave function. Does the wave function remain real? Does the initial state revive?

e. Repeat parts (c) and (d) using the initial wave function

$$\Psi(x,0) = \begin{cases} 2 & |x| \leq 1/8 \\ 0, & |x| > 1/8. \end{cases} \tag{16.24}$$

Problem 16.14. Coherent states

Because the energy eigenvalues of the simple harmonic oscillator are equally spaced, there exist wave functions known as *coherent states* whose probability density propagates quasi-classically.

a. Include a sufficient number of expansion coefficients for $V(x) = 10x^2$ to model an initial Gaussian wave function centered at the origin.

$$\Psi(x,0) = e^{-16x^2}. \tag{16.25}$$

Describe the evolution.

b. Repeat part (a) with

$$\Psi(x,0) = e^{-16(x-2)^2}. \tag{16.26}$$

c. Show that the wave functions in parts (a) and (b) change their width but not their Gaussian envelope. Construct a wave function with the following expansion coefficients and observe its behavior.

$$c_n^2 = \frac{\langle n\rangle^n}{n!} e^{-\langle n\rangle}. \tag{16.27}$$

The expectation of the number of quanta, $\langle n\rangle$, is given by

$$\langle n\rangle = \langle E\rangle - \frac{1}{2}\hbar\omega, \tag{16.28}$$

where $\langle E\rangle$ is the energy expectation value of the coherent state.

The expansion of an arbitrary wave function in terms of a set of eigenstates is closely related to Fourier analysis. Because the eigenstates of a particle in a box are sinusoidal functions, we could have used the fast Fourier transform algorithm (FFT) to compute the projection coefficients. Because these coefficients are calculated only once in Problem 16.14, evaluating (16.21) directly is reasonable. We will use the FFT to study wave functions in momentum space and to implement the operator splitting method for time evolution in Section 16.6.

16.5 The Time-Dependent Schrödinger Equation

Although the numerical solution of the time-independent Schrödinger equation (16.7) is straightforward for one particle, the numerical solution of the time-dependent Schrödinger equation (16.4) is not as simple. A naive approach to its numerical solution can be formulated by introducing a grid for the time coordinate and a grid for the spatial coordinate. We use the notation $t_n = t_0 + n\Delta t$, $x_s = x_0 + s\Delta x$, and $\Psi(x_s, t_n)$. The idea is to relate $\Psi(x_s, t_{n+1})$ to the value of $\Psi(x_s, t_n)$ for each value of x_s. An example of an algorithm that solves the Schrödinger-like equation $\partial\Psi/\partial t = \partial^2\Psi/\partial x^2$ to first-order in Δt is given by

$$\frac{1}{\Delta t}\big[\Psi(x_s, t_{n+1}) - \Psi(x_s, t_n)\big] = \frac{1}{(\Delta x)^2}\big[\Psi(x_{s+1}, t_n) - 2\Psi(x_s, t_n) + \Psi(x_{s-1}, t_n)\big]. \tag{16.29}$$

The right-hand side of (16.29) represents a finite difference approximation to the second derivative of Ψ with respect to x. Equation (16.29) is an example of an *explicit* scheme, because given Ψ at time t_n, we can compute Ψ at time t_{n+1}. Unfortunately, this explicit approach leads to unstable solutions, that is, the numerical value of Ψ diverges from the exact solution as Ψ evolves in time.

One way to avoid the instability is to retain the same form as (16.29), but to evaluate the spatial derivative on the right side of (16.29) at time t_{n+1} rather than time t_n:

$$\frac{1}{\Delta t}\big[\Psi(x_s, t_{n+1}) - \Psi(x_s, t_n)\big] = \frac{1}{(\Delta x)^2}\big[\Psi(x_{s+1}, t_{n+1}) - 2\Psi(x_s, t_{n+1}) + \Psi(x_{s-1}, t_{n+1})\big]. \tag{16.30}$$

Equation (16.30) is an *implicit* method because the unknown function $\Psi(x_s, t_{n+1})$ appears on both sides. To obtain $\Psi(x_s, t_{n+1})$, it is necessary to solve a set of linear equations at each time step. More details of this approach and the demonstration that (16.30) leads to stable solutions can be found in the references.

Visscher and others have suggested an alternative approach in which the real and imaginary parts of Ψ are treated separately and defined at different times. The algorithm ensures that the total probability remains constant. If we let

$$\Psi(x,t) = \mathrm{R}(x,t) + i\,\mathrm{I}(x,t), \tag{16.31}$$

then Schrödinger's equation, $i\partial\Psi(x,t)/\partial t = \hat{H}\Psi(x,t)$, becomes ($\hbar = 1$ as usual)

$$\frac{\partial \mathrm{R}(x,t)}{\partial t} = \hat{H}\,\mathrm{I}(x,t) \tag{16.32a}$$

$$\frac{\partial \mathrm{I}(x,t)}{\partial t} = -\hat{H}\,\mathrm{R}(x,t). \tag{16.32b}$$

A stable method of numerically solving (16.32) is to use a form of the half-step method (see Appendix 3A). The resulting difference equations are

$$\mathrm{R}(x,t+\Delta t) = \mathrm{R}(x,t) + \hat{H}\,\mathrm{I}(x,t+\frac{1}{2}\Delta t)\,\Delta t \tag{16.33a}$$

$$\mathrm{I}(x,t+\frac{3}{2}\Delta t) = \mathrm{I}(x,t+\frac{1}{2}\Delta t) - \hat{H}\,\mathrm{R}(x,t)\,\Delta t, \tag{16.33b}$$

where the initial values are given by $\mathrm{R}(x,0)$ and $\mathrm{I}(x,\frac{1}{2}\Delta t)$. Visscher has shown that this algorithm is stable if

$$\frac{-2\hbar}{\Delta t} \le V \le \frac{2\hbar}{\Delta t} - \frac{2\hbar^2}{(m\Delta x)^2}, \tag{16.34}$$

where the inequality (16.34) holds for all values of the potential V.

The appropriate definition of the probability density $P(x,t) = \mathrm{R}(x,t)^2 + \mathrm{I}(x,t)^2$ is not obvious, because R and I are not defined at the same time. The following choice conserves the total probability:

$$P(x,t) = \mathrm{R}(x,t)^2 + \mathrm{I}(x,t+\frac{1}{2}\Delta t)\,\mathrm{I}(x,t-\frac{1}{2}\Delta t) \tag{16.35a}$$

$$P(x,t+\frac{1}{2}\Delta t) = \mathrm{R}(t+\Delta t)\,\mathrm{R}(x,t) + \mathrm{I}(x,t+\frac{1}{2}\Delta t)^2. \tag{16.35b}$$

An implementation of (16.33) is given in the `TDHalfStep` class in Listing 16.7. The real part of the wave function first is updated for all positions, and then the imaginary part is updated using the new values of the real part.

Listing 16.7: The `TDHalfStep` class solves the one-dimensional time-dependent Schrödinger equation.

```
package org.opensourcephysics.sip.ch16;
public class TDHalfStep {
   double[] x, realPsi, imagPsi, potential;
   double dx, dx2;
   double dt = 0.001; // why not an input parameter? anyway its determined below

   public TDHalfStep(GaussianPacket packet, int numberOfPoints, double xmin, double xmax) {
```

```
    realPsi = new double[numberOfPoints];
    imagPsi = new double[numberOfPoints];
    potential = new double[numberOfPoints];
    x = new double[numberOfPoints];
    dx = (xmax-xmin)/(numberOfPoints-1);
    dx2 = dx*dx;
    double x0 = xmin;
    for(int i = 0, n = realPsi.length;i<n;i++) {
       x[i] = x0;
       potential[i] = getV(x0);
       realPsi[i] = packet.getReal(x0);
       imagPsi[i] = packet.getImaginary(x0);
       x0 += dx;
    }
    dt = getMaxDt();
    // advances the imaginary part by 1/2 step at start
    for(int i = 1, n = realPsi.length-1;i<n;i++) {
       // deltaRe = change in real part of psi in 1/2 step
       double deltaRe = potential[i]*realPsi[i]-0.5*(realPsi[i+1]-2*realPsi[i]+realPsi[i-1
       imagPsi[i] -= deltaRe*dt/2;
    }
 }

 double getMaxDt() {
    double dt = Double.MAX_VALUE;
    for(int i = 0, n = potential.length;i<n;i++) {
       if(potential[i]<0) {
          dt = Math.min(dt, -2/potential[i]);
       }
       double a = potential[i]+2/dx2;
       if(a>0) {
          dt = Math.min(dt, 2/a);
       }
    }
    return dt;
 }

 double step() {
    for(int i = 1, n = imagPsi.length-1;i<n;i++) {
       // don't like notation imH and reH. H is real
       double imH = potential[i]*imagPsi[i]-0.5*(imagPsi[i+1]-2*imagPsi[i]+imagPsi[i-1])/dx
       realPsi[i] += imH*dt;
    }
    for(int i = 1, n = realPsi.length-1;i<n;i++) {
       double reH = potential[i]*realPsi[i]-0.5*(realPsi[i+1]-2*realPsi[i]+realPsi[i-1])/dx
       imagPsi[i] -= reH*dt;
    }
    return dt;
 }
```

```
    public double getV(double x) {
        return 0; // change this statement to model other potentials
    }
}
```

Before we can use the `TDHalfStep` class, we need to choose an initial wave function. A convenient form is the Gaussian wave packet with a width w centered about x_0 given by

$$\Psi(x,0) = \left(\frac{1}{2\pi w^2}\right)^{1/4} e^{ik_0(x-x_0)}\, e^{-(x-x_0)^2/4w^2}. \tag{16.36}$$

The expectation value of the initial velocity of the wave packet is $\langle v\rangle = p_0/m = \hbar k_0/m$. Note that the wave function has a nonzero momentum expectation value, which is known as a *momentum boost*. An implementation of (16.36) is shown in the `GaussianPacket` class. The constructor is passed the width, center, and momentum of the packet. Real and imaginary values can then be calculated at any x to fill the wave function arrays.

Listing 16.8: The `GaussianPacket` class creates a wave function with a Gaussian probability distribution and a momentum boost.

```
package org.opensourcephysics.sip.ch16;
public class GaussianPacket {
    double w, x0, p0;
    double w42;
    double norm;

    public GaussianPacket(double width, double center, double momentum) {
        w = width;
        w42 = 4*w*w;
        x0 = center;
        p0 = momentum;
        norm = Math.pow(2*Math.PI*w*w, -0.25);
    }

    public double getReal(double x) {
        return norm*Math.exp(-(x-x0)*(x-x0)/w42)*Math.cos(p0*(x-x0));
    }

    public double getImaginary(double x) {
        return norm*Math.exp(-(x-x0)*(x-x0)/w42)*Math.sin(p0*(x-x0));
    }
}
```

To start the half-step algorithm, we need the value of $\mathrm{I}(x, t = \frac{1}{2}\Delta t)$ and $\mathrm{R}(x, t = 0)$. To obtain $\mathrm{I}(x, t = \frac{1}{2}\Delta t)$, we use the real component of the wave function to perform a half step.

$$\mathrm{I}(x, t + \frac{1}{2}\Delta t) = \mathrm{I}(x,t) - \hat{H}\,\mathrm{R}(x,t)\,\frac{\Delta t}{2} \tag{16.37}$$

The normalization factor must be computed after we correct the initial wave function using (16.37). For completeness, we list the `TDHalfStepApp` target class.

Listing 16.9: The `TDHalfStepApp` class solves the time-independent Schrödinger equation and displays the wave function.

```
package org.opensourcephysics.sip.ch16;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.frames.ComplexPlotFrame;

public class TDHalfStepApp extends AbstractSimulation {
   ComplexPlotFrame psiFrame = new ComplexPlotFrame("x", "|Psi|", "Wave function");
   TDHalfStep wavefunction;
   double time;

   public TDHalfStepApp() {
      psiFrame.limitAutoscaleY(-1, 1); // do not autoscale within this y-range.
   }

   public void initialize() {
      time = 0;
      psiFrame.setMessage("t="+0);
      double xmin = control.getDouble("xmin");
      double xmax = control.getDouble("xmax");
      int numberOfPoints = control.getInt("number of points");
      double width = control.getDouble("packet width");
      double x0 = control.getDouble("packet offset");
      double momentum = control.getDouble("packet momentum");
      GaussianPacket packet = new GaussianPacket(width, x0, momentum);
      wavefunction = new TDHalfStep(packet, numberOfPoints, xmin, xmax);
      psiFrame.clearData(); // removes old data
      psiFrame.append(wavefunction.x, wavefunction.realPsi, wavefunction.imagPsi);
   }

   public void doStep() {
      time += wavefunction.step();
      psiFrame.clearData();
      psiFrame.append(wavefunction.x, wavefunction.realPsi, wavefunction.imagPsi);
      psiFrame.setMessage("t="+decimalFormat.format(time));
   }

   public void reset() {
      control.setValue("xmin", -20);
      control.setValue("xmax", 20);
      control.setValue("number of points", 500);
      control.setValue("packet width", 1);
      control.setValue("packet offset", -15);
      control.setValue("packet momentum", 2);
      setStepsPerDisplay(10); // multiple computations per animation step
      enableStepsPerDisplay(true);
      initialize();
   }
```

```
   public static void main(String[] args) {
      SimulationControl.createApp(new TDHalfStepApp());
   }
}
```

Problem 16.15. Evolution of a wave packet

a. Add an array to `TDHalfStepApp` that saves the imaginary part of the wave function at the previous time step so that the probability density can be computed using (16.35). Show that the probability is conserved.

b. Use `TDHalfStepApp` to follow the motion of a wave packet in a potential-free region. Let $x_0 = -15$, $k_0 = 2$, $w = 1$, `dx` $= 0.4$, and `dt` $= 0.1$. Suitable values for the minimum and maximum values of x on the grid are `xmin` $= -20$ and `xmax` $= 20$. What is the shape of the wave packet at different times? Does the shape of the wave packet depend on your choice of the parameters k_0 and w?

c. Modify `TDHalfStepApp` so that the quantities $x_0(t)$ and $w(t)$, the position and width of the wave packet as a function of time, can be measured directly. What is a reasonable definition of $w(t)$? What is the qualitative dependence of x_0 and w on t? How do your results change if the initial width of the packet is reduced by a factor of four?

Problem 16.16. Evolution of wave packet incident on a potential step

a. Use `TDHalfStepApp` with a step potential beginning at $x = 0$ with height $V_0 = 2$. Choose $x_0 = -10$, $k_0 = 2$, $w = 1$, `dx` $= 0.4$, `dt` $= 0.1$, `xmin` $= -20$, and `xmax` $= 20$. Describe the motion of the wave packet. Does the shape of the wave packet remain a Gaussian for all t? What happens to the wave packet at $x = 0$? Determine the height and width of the reflected and transmitted wave packets, the time t_i for the incident wave to reach the barrier at $x = 0$, and the time t_r for the reflected wave to return to $x = x_0$. Is $t_r = t_i$? If these times are not equal, explain the reason for the difference.

b. Repeat the analysis in part (a) for a step potential of height $V_0 = 10$. Is $t_r \approx t_i$ in this case?

c. What is the motion of a classical particle with a kinetic energy corresponding to the central wave vector $k = k_0$?

Problem 16.17. Scattering of a wave packet from a potential barrier

a. Consider a potential barrier of the form

$$V(x) = \begin{cases} 0 & x < 0 \\ V_0 & 0 \le x \le a \\ 0 & x > a \end{cases} \tag{16.38}$$

Generate a series of snapshots that show the wave packet approaching the barrier and then interacting with it to generate reflected and transmitted packets. Choose $V_0 = 2$ and $a = 1$ and

consider the behavior of the wave packet for $k_0 = 1$, 1.5, 2, and 3. Does the width of the packet increase with time? How does the width depend on k_0? For what values of k_0 is the motion of the packet in qualitative agreement with the motion of a corresponding classical particle?

b. Consider a square well with $V_0 = -2$ and consider the same questions as in part (a).

Problem 16.18. Evolution of two wave packets

Modify `GaussianPacket` in Listing 16.8 to include two wave packets with identical widths and speeds, with the sign of k_0 chosen so that the two wave packets approach each other. Choose their respective values of x_0 so that the two packets are initially well separated. Let $V = 0$ and describe what happens when you determine their time dependence. Do the packets influence each other? What do your results imply about the existence of a superposition principle?

16.6 Fourier Transformations and Momentum Space

The position space wave function, $\Psi(x,t)$, is only one of many possible representations of a quantum mechanical state. A quantum system also is completely characterized by the momentum space wave function, $\Phi(p,t)$. The probability $P(p,t)\,\Delta p$ of the particle being in a "volume" element Δp centered about the momentum p at time t is equal to

$$P(p,t)\,\Delta p = |\Phi(p,t)|^2 \Delta p. \tag{16.39}$$

Because either a position space or a momentum space representation provides a complete description of the system, it is possible to transform the wave function from one space to another as:

$$\Phi(p,t) = \frac{1}{\sqrt{2\pi\hbar}} \int_{-\infty}^{\infty} \Psi(x,t) e^{-ipx/\hbar} dx, \tag{16.40}$$

$$\Psi(x,t) = \frac{1}{\sqrt{2\pi\hbar}} \int_{-\infty}^{\infty} \Phi(p,t) e^{ipx/\hbar} dp. \tag{16.41}$$

The momentum and position space transformations, (16.40) and (16.41), are Fourier integrals. Because a computer stores a wave function on a finite grid, these transformations simplify to the familiar Fourier series (see Section 9.3):

$$\Phi_m = \sum_{n=-N/2}^{N/2} \Psi_n e^{-ip_m x_n/\hbar}, \tag{16.42}$$

$$\Psi_n = \frac{1}{N} \sum_{m=-N/2}^{N/2} \Phi_m e^{ip_m x_n/\hbar}, \tag{16.43}$$

where $\Phi_m = \Phi(p_m)$ and $\Psi_n = \Psi(x_n)$. We have not explicitly shown the time dependence in (16.42) and (16.43).

We now use the `FFTApp` program introduced in Section 9.3 to transform a wave function between position and momentum space. Note that the wavenumber $2\pi/\lambda$ (or $2\pi/T$ in the time

domain) in classical physics has the same numerical value as momentum in quantum mechanics $p = h/\lambda = 2\pi\hbar/\lambda$ in units such that $\hbar = 1$. Consequently, we can use the `getWrappedOmega` and `getNaturalOmega` methods in the `FFT` class to generate arrays containing momentum values for a transformed position space wave function.

The `FFTApp` program in Listing 9.7 transforms N complex data points using an input array that has length $2N$. The real part of the jth data point is stored in array element $2j$ and the imaginary part is stored in element $2j + 1$. The `FFT` class transforms this array and maintains the same ordering of real and imaginary parts. However, the momenta (wavenumbers) are in warp-around order starting with the zero momentum coefficients in the first two elements and switching to negative momenta half way through the array. The `toNaturalOrder` class sorts the array in order of increasing momentum. We use the `FFTApp` class in Problem 16.19.

Problem 16.19. Transforming to momentum space

a. The `FFTApp` class initializes the wave function grid using the following complex exponential:

$$\Psi_n = \Psi(n\Delta x) = e^{in\Delta x} = \cos n\Delta x + i \sin n\Delta x. \tag{16.44}$$

Use `FFTApp` to show that a complex exponential has a definite momentum if the grid contains an integer number of wavelengths. In other words, show that there is only one nonzero Fourier component.

b. How small a wavelength (or how large a momentum) can be modeled if the spatial grid has N points and extends over a distance L?

c. Where do the maximum, zero, and minimum values of the momentum occur in wrap-around order?

After the transformation, the momentum space wave function is stored in an array. The array elements can be assigned a momentum value using the de Broglie relation $p = h/\lambda$. The longest wavelength that can exist on the grid is equal to the grid dimension, $L = (N - 1)\Delta x$, and this wave has a momentum of

$$p_0 = \frac{h}{L}. \tag{16.45}$$

Points on the momentum grid have momentum values with integer multiples of p_0.

Problem 16.20. Momentum visualization

Add a `ComplexPlotFrame` to the `FFTApp` program to show the momentum space wave function of a position space Gaussian wave packet. Add a user interface to control the width of the Gaussian wave packet and verify the Heisenberg uncertainty relation, $\Delta x \Delta p \geq \hbar/2$. Shift the center of the position space wavepacket and explain the change in the resulting momentum space wave function.

Problem 16.21. Momentum time evolution

Modify `TDHalfStepApp` so that it displays the momentum space wave function in addition to the position space wave function. Describe the momentum space evolution of a Gaussian packet for the infinite square well and a simple harmonic oscillator potential. What evidence of classical-like behavior do you observe?

The FFT can be used to implement a fast and accurate method for solving Schrödinger's equation. We start by writing (16.4) in operator notation as

$$i\hbar\frac{\partial\Psi(x,t)}{\partial t} = \hat{H}\Psi(x,t) = (\hat{T}+\hat{V})\Psi(x,t) \tag{16.46}$$

where $\hat{H}$, $\hat{T}$, and $\hat{V}$ are the Hamiltonian, kinetic energy, and potential energy operators, respectively. The formal solution to (16.46) is

$$\Psi(x,t) = e^{-i\hat{H}(t-t_0)/\hbar}\Psi(x,t_0) = e^{-i(\hat{T}+\hat{V})(t-t_0)/\hbar}\Psi(x,t_0). \tag{16.47}$$

The time evolution operator, $\hat{U}$, is defined as

$$\hat{U} = e^{-i\hat{H}(t-t_0)/\hbar} = e^{-i(\hat{T}+\hat{V})(t-t_0)/\hbar}. \tag{16.48}$$

It might be tempting to express the time evolution operator as

$$\hat{U} = e^{-i\hat{T}\Delta t/\hbar}e^{-i\hat{V}\Delta t/\hbar}, \tag{16.49}$$

but (16.49) is valid only for $\Delta t \equiv t - t_0 << 1$, because $\hat{T}$ and $\hat{V}$ do not commute. A more accurate approximation (accurate to second order in Δt) is obtained by using the following symmetric decomposition

$$\hat{U} = e^{-i\hat{V}\Delta t/2\hbar}e^{-i\hat{T}\Delta t/\hbar}e^{-i\hat{V}\Delta t/2\hbar}. \tag{16.50}$$

The key to using (16.50) to solve (16.46) is to use the position space wave function when applying $e^{-i\hat{V}\Delta t/2\hbar}$ and to use the momentum space wave function when applying $e^{-i\hat{T}\Delta t/2\hbar}$. In position space, the potential energy operator is equivalent to simply multiplying by the potential energy function. That is, the effect of the first and last terms in (16.50) is to multiply points on the position grid by a phase factor that is proportional to the potential energy:

$$\tilde{\Psi}_j = e^{-iV(x_j)\Delta t/2\hbar}\Psi_j. \tag{16.51}$$

Because the kinetic energy operator in position space involves partial derivatives, it is convenient to transform both the operator and the wave function to momentum space. In momentum space the kinetic energy operator is equivalent to multiplying by the kinetic energy $p^2/2m$. The middle term in (16.50) operates by multiplying points on the momentum grid by a phase factor that is proportional to the kinetic energy:

$$\tilde{\Phi}_j = e^{-ip_j^2\Delta t/2m}\Phi_j. \tag{16.52}$$

The split-operator algorithm jumps back and forth between position and momentum space to propagate the wave function. The algorithm starts in position space where each grid value, $\Psi_j = \Psi(x_j,t)$ is multiplied by (16.51). The wave function is then transformed to momentum space where every momentum value, Φ_j, is multiplied by (16.52). It is then transformed back to position space where (16.51) is applied a second time. A single time step can therefore be written as

$$\Psi(x,t+\Delta t) = e^{-iV(x)\Delta t/2\hbar}F^{-1}[e^{-ip^2\Delta t/2m}F[e^{-iV(x)\Delta t/2\hbar}\Psi(x,t)]], \tag{16.53}$$

where F is the Fourier transform to momentum space and F^{-1} is its inverse.

Problem 16.22. Split-operator algorithm

a. Write a program to implement the split operator algorithm. It is necessary to evaluate the exponential phase factors only once when implementing the split-operator algorithm. Store the complex exponentials in arrays that match the x values on the spatial grid and the p values on the momentum grid. Use wrap-around order when storing the momentum phase factors because the `FFT` class inverse transformation assumes that data are in wrap-around order. You can use the `getWrappedOmega` method in the FFT to obtain the momenta in this ordering.

b. Compare the evolution of a Gaussian wave packet using the split-operator and half-step algorithms using identical grids. How does the finite grid size affect each algorithm?

c. Compare the computation speed of the split-operator and half-step algorithms using a Gaussian wave packet in a square well. Disable plotting and other non-essential computation when comparing the speeds.

Problem 16.23. Split-operator accuracy

The split-operator and half-step algorithms fail if the time step is too large. Use both algorithms to evolve a simple harmonic oscillator coherent state (see Problem 16.14). Describe the error that occurs if the time step becomes too large.

16.7 Variational Methods

One way of obtaining a good approximation to the ground state energy is to use a variational method. This approach has numerous applications in chemistry, atomic and molecular physics, nuclear physics, and condensed matter physics. Consider a system whose Hamiltonian operator $\hat{H}$ is given by (16.8). According to the variational principle, the expectation value of the Hamiltonian for an arbitrary trial wave function Ψ is greater than or equal to the ground state energy E_0. That is,

$$\langle H \rangle = E[\Psi] = \frac{\int \Psi^*(x) \hat{H} \Psi(x)\, dx}{\int \Psi^*(x) \Psi(x)\, dx} \geq E_0, \tag{16.54}$$

where E_0 is the exact ground state energy of the system. We assume that the wave function is continuous and bounded. The inequality (16.54) reduces to an equality only if Ψ is an eigenstate of $\hat{H}$ with the eigenvalue E_0. For bound states, Ψ may be assumed to be real without loss of generality so that $\Psi^* = \Psi$ and thus $|\Psi(x)|^2 = \Psi(x)^2$. This assumption implies that we do not need to store two values representing the real and imaginary parts of Ψ.

The inequality (16.54) is the basis of the variational method. The procedure is to choose a physically reasonable form for the trial wave function $\Psi(x)$ that depends on one or more parameters. The expectation value $E[\Psi]$ is computed, and the parameters are varied until a minimum of $E[\Psi]$ is obtained. This value of $E[\Psi]$ is an upper bound to the true ground state energy. Often forms of Ψ are chosen so that the integrals in (16.54) can be done analytically. To avoid this restriction we can use numerical integration methods.

In most applications of the variational method the integrals in (16.54) are multidimensional and Monte Carlo integration methods are essential. For this reason we will use Monte Carlo

integration in the following, even though we will consider only one and two body problems. Because it is inefficient to simply choose points at random to compute $E[\Psi]$, we rewrite (16.54) in a form that allows us to use importance sampling. We write

$$E[\Psi] = \frac{\int \Psi(x)^2 E_L(x)\, dx}{\int \Psi(x)^2\, dx}, \tag{16.55}$$

where E_L is the *local energy*,

$$E_L(x) = \frac{\hat{H}\Psi(x)}{\Psi(x)}, \tag{16.56}$$

which can be calculated analytically using the trial wave function. The form of (16.55) is that of a weighted average with the weight equal to the normalized probability density $\Psi(x)^2/\int \Psi(x)^2\, dx$. As discussed in Section 11.6, we can sample values of x using the distribution $\Psi(x)^2$ so that the Monte Carlo estimate of $E[\Psi]$ is given by the sum

$$E[\Psi] = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} E_L(x_i), \tag{16.57}$$

where n is the number of times that x is sampled from Ψ^2. How can we sample from Ψ^2? In general, it is not possible to use the inverse transform method (see Section 11.5) to generate a nonuniform distribution. A convenient alternative is the Metropolis method which has the advantage that only an unnormalized Ψ^2 is needed for the proposed move.

Problem 16.24. Ground state energy of several one-dimensional systems

a. It is useful to test the variational method on an exactly solvable problem. Consider the one-dimensional harmonic oscillator with $V(x) = x^2/2$. Choose the trial wave function to be $\Psi(x) \propto e^{-\lambda x^2}$, with λ the variational parameter. Generate values of x chosen from a normalized $\Psi^2(x)$ using the inverse transform method, and verify that $\lambda = 1/2$ yields the smallest upper bound, by considering $\lambda = 1/2$ and four other values of λ near $1/2$. Another way to generate a Gaussian distribution is to use the Box-Muller method discussed in Section 11.5.

b. Repeat part (a) using the Metropolis method to generate x distributed according to $\Psi(x)^2 \propto e^{-2\lambda x^2}$ and evaluate (16.57). As discussed in Section 11.7, the Metropolis method can be summarized by the following steps:

   i. Choose a trial position $x_{\text{trial}} = x_n + \delta_n$, where δ_n is a uniform random number in the interval $[-\delta, \delta]$.

   ii. Compute $w = p(x_{\text{trial}})/p(x_n)$, where in this case $p(x) = e^{-2\lambda x^2}$.

   iii. If $w \geq 1$, accept the change and let $x_{n+1} = x_{\text{trial}}$.

   iv. If $w < 1$, generate a random number r and let $x_{n+1} = x_{\text{trial}}$ if $r \leq w$.

   v. If the trial change is not accepted, then let $x_{n+1} = x_n$.

Remember that it is necessary to wait for equilibrium (convergence to the distribution Ψ^2) before computing the average value of E_L. Look for a systematic trend in $\langle E_L \rangle$ over the course of the random walk. Choose a step size δ that gives a reasonable value for the acceptance ratio. How many trials are necessary to obtain $\langle E_L \rangle$ to within 1% accuracy compared to the exact analytic result?

c. Instead of finding the minimum of $\langle E_L \rangle$ as a function of the various variational parameters, minimize the quantity

$$\sigma_L^2 = \langle E_L^2 \rangle - \langle E_L \rangle^2. \tag{16.58}$$

Verify that the exact minimum value of $\sigma_L^2[\Psi]$ is zero, whereas the exact minimum value of $E_L[\Psi]$ is unknown in general.

d. Consider the anharmonic potential $V(x) = \frac{1}{2}x^2 + bx^4$. Plot $V(x)$ as a function of x for $b = 1/8$. Use first-order perturbation theory to calculate the lowest order change in the ground state energy due to the x^4 term. Then choose a reasonable form for your trial wave function and use your Monte Carlo program to estimate the ground state energy. How does your result compare with first-order perturbation theory?

e. Consider the anharmonic potential of part (d) with $b = -1/8$. Plot $V(x)$. Use first-order perturbation theory to calculate the lowest order change in the ground state energy due to the x^4 term, and then use your program to estimate E_0. Do your Monte Carlo estimates for the ground state energy have a lower bound? Why or why not?

f. Modify your program so that it can be applied to the ground state of the hydrogen atom. In this case we have $V(r) = -e^2/r$, where e is the magnitude of the charge on the electron. The element of integration dx in (16.55) is replaced by $4\pi r^2\, dr$. Choose $\Psi \propto e^{-r/a}$, where a is the variational parameter. Measure lengths in terms of the Bohr radius $\hbar^2/me^2$ and energy in terms of the Rydberg $me^4/2\hbar^2$. In these units $\mu = e^2 = \hbar = 1$. Find the optimal value of a. What is the corresponding energy?

g. Consider the Yukawa or screened Coulomb potential for which $V(r) = -e^2 e^{-\alpha r}/r$, where $\alpha > 0$. In this case the ground state and wave function can only be obtained numerically. For $\alpha = 0.5$ and $\alpha = 1.0$ the most accurate numerical estimates of E_0 are -0.14808 and -0.01016, respectively. What is a good choice for the form of the trial wave function? How close can you come to these estimates?

Problem 16.25. Variational estimate of the ground state of Helium

Helium has long served as a testing ground for atomic trial wave functions. Consider the ground state of the helium atom with the interaction

$$V(r_1, r_2) = -2e^2\Big(\frac{1}{r_1} + \frac{1}{r_2}\Big) + \frac{e^2}{r_{12}}, \tag{16.59}$$

where r_{12} is the separation between the two electrons. Assume that the nucleus is fixed and ignore relativistic effects. Choose $\Psi(\mathbf{r}_1, \mathbf{r}_2) = Ae^{-Z_{\text{eff}}(r_1+r_2)/a_0}$, where Z_{eff} is a variational parameter. Estimate the upper bound to the ground state energy based on this functional form of Ψ.

Our discussion of variational Monte Carlo methods has been only introductory in nature. One important application of variational Monte Carlo methods is to optimize a given trial wave function which is then used to "guide" the Monte Carlo methods discussed in Sections 16.8 and 16.9.

16.8 Random Walk Solutions of the Schrödinger Equation

We now introduce a Monte Carlo approach based on expressing the Schrödinger equation in imaginary time. This approach follows that of Anderson (see references). We will then discuss several other *quantum Monte Carlo* methods. We will see that although the systems of interest are quantum mechanical, we can convert them to systems for which we can use classical Monte Carlo methods.

To understand how we can interpret the Schrödinger equation in terms of a random walk in imaginary time, we substitute $\tau = it/\hbar$ into the time-dependent Schrödinger equation for a free particle and write (in one dimension)

$$\frac{\partial \Psi(x,\tau)}{\partial \tau} = \frac{\hbar^2}{2m}\frac{\partial^2 \partial(x,\tau)}{\partial x^2}. \tag{16.60}$$

Note that (16.60) is identical in form to the diffusion equation (16.1). Hence, we can interpret the wave function Ψ as a probability density with a diffusion constant $D = \hbar^2/2m$.

From our discussion in Chapter 7, we know that we can use the formal similarity between the diffusion equation and the imaginary-time free particle Schrödinger equation to solve the latter by replacing it by an equivalent random walk problem. To understand how we can interpret the role of the potential energy term in the context of random walks, we write Schrödinger's equation in imaginary time as

$$\frac{\partial \Psi(x,\tau)}{\partial \tau} = \frac{\hbar^2}{2m}\frac{\partial^2 \Psi(x,\tau)}{\partial x^2} - V(x)\Psi(x,\tau). \tag{16.61}$$

If we were to ignore the first-term (the diffusion term) on the right side of (16.61), the result would be a first-order differential equation corresponding to a decay or growth process depending on the sign of V. We can obtain the solution to this first-order equation by replacing it by a random decay or growth process, for example, radioactive decay. These considerations suggest that we can interpret (16.61) as a combination of diffusion and branching processes. In the latter, the number of walkers increases or decreases at a point x depending on the sign of $V(x)$. The walkers do not interact with each other because the Schrödinger equation (16.61) is linear in Ψ. Note that it is $\Psi\Delta x$ and *not* $\Psi^2\Delta x$ that corresponds to the probability distribution of the random walkers. This probabilistic interpretation requires that Ψ be nonnegative and real.

We now use this probabilistic interpretation of (16.61) to develop an algorithm for determining the ground state wave function and energy. The general solution of Schrödinger's equation can be written for imaginary time τ as (see (16.10))

$$\Psi(x,\tau) = \sum_n c_n\, \phi_n(x)\, e^{-E_n\tau}. \tag{16.62}$$

For sufficiently large τ, the dominant term in the sum in (16.62) comes from the term representing the eigenvalue of lowest energy. Hence we have

$$\Psi(x,\tau \to \infty) = c_0\, \phi_0(x)\, e^{-E_0\tau}. \tag{16.63}$$

From (16.63) we see that the spatial dependence of $\Psi(x,\tau \to \infty)$ is proportional to the ground state eigenstate $\phi_0(x)$. If $E_0 > 0$, we also see that $\Psi(x,\tau)$ and hence the population of walkers will

eventually decay to zero unless $E_0 = 0$. This problem can be avoided by measuring E_0 from an arbitrary reference energy $V_{\rm ref}$, which is adjusted so that an approximate steady state distribution of random walkers is obtained.

Although we could attempt to fit the τ-dependence of the computed probability distribution of the random walkers to (16.63) and thereby extract E_0, it is more convenient to compute E_0 directly from the relation

$$E_0 = \langle V \rangle = \frac{\sum n_i V(x_i)}{\sum n_i}, \tag{16.64}$$

where n_i is the number of walkers at x_i at time τ. An estimate for E_0 can be found by averaging the sum in (16.64) for several values of τ once a steady state distribution of random walkers has been reached. To derive (16.64), we rewrite (16.61) and (16.63) by explicitly introducing the reference potential $V_{\rm ref}$:

$$\frac{\partial \Psi(x,\tau)}{\partial \tau} = \frac{\hbar^2}{2m}\frac{\partial^2 \Psi(x,\tau)}{\partial x^2} - \big[V(x) - V_{\rm ref}\big]\Psi(x,\tau), \tag{16.65}$$

and

$$\Psi(x,\tau) \approx c_0 \phi_0(x)\, e^{-(E_0 - V_{\rm ref})\tau}. \tag{16.66}$$

We first integrate (16.65) with respect to x. Because $\partial\Psi(x,\tau)/\partial x$ vanishes in the limit $|x| \to \infty$, $\int (\partial^2\Psi/\partial x^2)dx = 0$, and hence

$$\int \frac{\partial \Psi(x,\tau)}{\partial \tau}\, dx = -\int V(x)\, \Psi(x,\tau)\, dx + V_{\rm ref}\int \Psi(x,\tau)\, dx. \tag{16.67}$$

If we differentiate (16.66) with respect to τ, we obtain the relation

$$\frac{\partial \Psi(x,\tau)}{\partial \tau} = (V_{\rm ref} - E_0)\Psi(x,\tau). \tag{16.68}$$

We then substitute (16.68) for $\partial\Psi/\partial\tau$ into (16.67) and find

$$\int (V_{\rm ref} - E_0)\Psi(x,\tau)\, dx = -\int V(x)\Psi(x,\tau)\, dx + V_{\rm ref}\int \Psi(x,\tau)\, dx. \tag{16.69}$$

If we cancel the terms proportional to $V_{\rm ref}$ in (16.69), we find that

$$E_0 \int \Psi(x,\tau)\, dx = \int V(x), \Psi(x,\tau)\, dx, \tag{16.70}$$

or

$$E_0 = \frac{\int V(x)\Psi(x,\tau)\, dx}{\int \Psi(x,\tau)\, dx}. \tag{16.71}$$

The desired result (16.64) follows by making the connection between $\Psi(x)\,\Delta x$ and the density of walkers between x and $x + \Delta x$.

Although the derivation of (16.64) is somewhat involved, the random walk algorithm is straightforward. A simple implementation of the algorithm is as follows:

1. Place a total of N_0 walkers at the initial set of positions x_i, where the x_i need not be on a grid.

2. Compute the reference energy, $V_{\rm ref} = \sum_i V_i/N_0$.

3. Randomly move the first walker to the right or left by a fixed step length Δs. The step length Δs is related to the time step $\Delta\tau$ by $(\Delta s)^2 = 2D\Delta\tau$. ($D = 1/2$ in units such that $\hbar = m = 1$.)

4. Compute $\Delta V = V(x) - V_{\rm ref}$ and a random number r in the unit interval. If $\Delta V > 0$ and $r < \Delta V\Delta\tau$, then remove the walker. If $\Delta V < 0$ and $r < -\Delta V\Delta\tau$, then add another walker at x. Otherwise, just leave the walker at x. This procedure is accurate only in the limit of $\Delta\tau << 1$. A more accurate procedure consists of computing $P_b = e^{-\Delta V\Delta\tau} - 1 = n + f$, where n is the integer part of P_b, and f is the fractional part. We then make n copies of the walker, and if $f > r$, we make one more copy.

5. Repeat steps 3 and 4 for each of the N_0 walkers and compute the mean potential energy (16.71) and the actual number of random walkers. The new reference potential is given by

$$V_{\rm ref} = \langle V\rangle - \frac{a}{N_0\Delta\tau}(N - N_0), \tag{16.72}$$

   where N is the new number of random walkers and $\langle V\rangle$ is their mean potential energy. The average of V is an estimate of the ground state energy. The parameter a is adjusted so that the number of random walkers N remains approximately constant.

6. Repeat steps 3–5 until the estimates of the ground state energy $\langle V\rangle$ have reached a steady state value with only random fluctuations. Average $\langle V\rangle$ over many Monte Carlo steps to compute the ground state energy. Do a similar calculation to estimate the distribution of random walkers.

The `QMWalk` class implements this algorithm for the harmonic oscillator potential. Initially, the walkers are randomly distributed within a distance `initialWidth` of the origin. The program also estimates the ground state wavefunction by accumulating the spatial distribution of the walkers at discrete intervals of position. The input parameters are the desired number of walkers N_0, the number of position intervals to accumulate data for the ground state wavefunction `numberOfBins`, and the step size `ds`. We also use `ds` for the interval size in the wavefunction computation. The program computes the current number of walkers, the estimate of the ground state energy, and the value of $V_{\rm ref}$. The unnormalized ground state wavefunction also is plotted.

Listing 16.10: The `QMWalk` class calculates the ground state of the simple harmonic oscillator using the random walk Monte Carlo algorithm.

```
package org.opensourcephysics.sip.ch16;
public class QMWalk {
   int numberOfBins = 1000; // number of bins to accumulate data for wave function
   double[] x;              // positions of walkers
   double[] phi0;           // estimate of ground state wave function
   double[] xv;             // x values for computing phi0
   int N0;                  // desired number of walkers
   int N;                   // actual number of walkers
   double ds;               // step size
   double dt;               // time interval
```

```
double vave = 0;          // mean potential
double vref = 0;          // reference potential
double eAccum = 0;        // accumulation of energy values
double xmin;              // minimum x
int mcs;

public void initialize() {
   N0 = N;
   x = new double[2*numberOfBins];
   phi0 = new double[numberOfBins];
   xv = new double[numberOfBins];
   xmin = -ds*numberOfBins/2.0; // minimum location for computing phi0
   double binEdge = xmin;
   for(int i = 0;i<numberOfBins;i++) {
      xv[i] = binEdge;
      binEdge += ds;
   }
   double initialWidth = 1; // initial width for location of walkers
   for(int i = 0;i<N;i++) {
      x[i] = (2*Math.random()-1)*initialWidth; // initial random location of walkers
      vref += potential(x[i]);
   }
   vave = 0;
   vref = 0;
   eAccum = 0;
   mcs = 0;
   dt = ds*ds;
}

void walk() {
   double vsum = 0;
   for(int i = N-1;i>=0;i--) {
      if(Math.random()<0.5) { // move walker
         x[i] += ds;
      } else {
         x[i] -= ds;
      }
      double pot = potential(x[i]);
      double dv = pot-vref;
      vsum += pot;
      if(dv<0) {              // decide to add or delete walker
         if(N==0||(Math.random()<-dv*dt)&&(N<x.length)) {
            x[N] = x[i];      // new walker at the current location
            vsum += pot;      // add energy of new walker
            N++;
         }
      } else {
         if((Math.random()<dv*dt)&&(N>0)) {
            N--;
            x[i] = x[N];      // relabel last walker to deleted walker index
```

```
                vsum -= pot;       // substract energy of deleted walker
            }
         }
      }
      vave = (N==0) ? 0 // if no walkers poential = 0
                    : vsum/N;
      vref = vave-(N-N0)/N0/dt;
      mcs++;
   }

   void doMCS() {
      walk();
      eAccum += vave;
      for(int i = 0;i<N;i++) {
         int bin = (int) Math.floor((x[i]-xmin)/ds); // calculate bin index
         if(bin>=0&&bin<numberOfBins) {
            phi0[bin]++;
         }
      }
   }

   void resetData() {
      for(int i = 0;i<numberOfBins;i++) {
         phi0[i] = 0;
      }
      eAccum = 0;
      mcs = 0;
   }

   public double potential(double x) {
      return 0.5*x*x;
   }
}
```

Listing 16.11: The `QMWalkApp` class computes and displays the result of a random walk Monte Carlo calculation.

```
package org.opensourcephysics.sip.ch16;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.frames.PlotFrame;

public class QMWalkApp extends AbstractSimulation {
   PlotFrame phiFrame = new PlotFrame("x", "Phi_0", "Phi_0(x)");
   QMWalk qmwalk = new QMWalk();

   public void initialize() {
      qmwalk.N = control.getInt("initial number of walkers");
      qmwalk.ds = control.getDouble("step size ds");
      qmwalk.numberOfBins = control.getInt("number of bins for wavefunction");
      qmwalk.initialize();
```

```
    }

    public void doStep() {
        qmwalk.doMCS();
        phiFrame.clearData();
        phiFrame.append(0, qmwalk.xv, qmwalk.phi0);
        phiFrame.setMessage("E = "+decimalFormat.format(qmwalk.eAccum/qmwalk.mcs)+" N = "+qmwa
    }

    public void reset() {
        control.setValue("initial number of walkers", 50);
        control.setValue("step size ds", 0.1);
        control.setValue("number of bins for wavefunction", 100);
        enableStepsPerDisplay(true);
    }

    public void resetData() {
        qmwalk.resetData();
        phiFrame.clearData();
        phiFrame.repaint();
    }

    public static void main(String[] args) {
        SimulationControl control = SimulationControl.createApp(new QMWalkApp());
        control.addButton("resetData", "Reset Data");
    }
}
```

Problem 16.26. Ground state of the harmonic and anharmonic oscillators

a. Use `QMWalk` and `QMWalkApp` to estimate the ground state energy E_0 and the corresponding eigenstate for $V(x) = x^2/2$. Choose the desired number of walkers $N_0 = 50$, the step length `ds` $= 0.1$, and `numberOfBins` $= 100$. Place the walkers at random within the range $-1 \leq x \leq 1$. Compare your Monte Carlo estimate for E_0 to the exact result $E_0 = 0.5$.

b. Reset your data averages after the averages seemed to have converged and compute the averages again. How many Monte Carlo steps per walker are needed for 1% accuracy in E_0? Plot the probability distribution of the random walkers and compare it to the exact result for the ground state wave function.

c. Modify `QMWalk` so that more than one copy of the walker can be created at each step (see step 4 on page 725). How much better does the algorithm work now? Can you use a larger step size or fewer Monte Carlo steps to obtain the same accuracy?

d. Obtain a numerical solution of the anharmonic oscillator with

$$V(x) = \frac{1}{2}x^2 + bx^3. \tag{16.73}$$

Consider $b = 0.1$, 0.2, and 0.5. A calculation of the effect of the x^3 term is necessary for the study of the anharmonicity of the vibrations of a physical system, for example, the vibrational spectrum of diatomic molecules.

Problem 16.27. Ground state of a square well

a. Modify `QMWalkApp` to find the ground state energy and wave function for the finite square well potential (16.13) with $a = 1$ and $V_0 = 5$. Choose $N_0 = 100$, `ds` $= 0.1$, and `numberOfBins` $= 100$. Place the walkers at random within the range $-1.5 \le x \le 1.5$.

b. Increase V_0 and find the ground state energy as a function of V_0. Use your results to estimate the limiting value of the ground state energy for $V_0 \to \infty$.

Problem 16.28. Ground state of a cylindrical box

Compute the ground state energy and wave function of the circular potential

$$V(r) = \begin{cases} 0 & r \le 1 \\ -V_0, & r > 1. \end{cases} \tag{16.74}$$

where $r^2 = x^2 + y^2$. Modify `QMWalkApp` by using Cartesian coordinates in two dimensions, for example, add an array to store the positions of the y coordinates of the walkers. What happens if you begin with an initial distribution of walkers that is not cylindrically symmetric?

16.9 Diffusion Quantum Monte Carlo

We now discuss an improvement of the random walk algorithm known as *diffusion quantum Monte Carlo*. Although some parts of the discussion might be difficult to follow initially, the algorithm is straightforward. Your understanding of the method will be enhanced by writing a program to implement the algorithm and then reading the following derivation again.

To provide some background, we introduce the concept of a Green's function or propagator defined by

$$\Psi(x,\tau) = \int G(x,x',\tau)\Psi(x,0)\,dx'. \tag{16.75}$$

From the form of (16.75) we see that $G(x,x',\tau)$ "propagates" the wave function from time zero to time τ. If we operate on both sides of (16.75) with first $(\partial/\partial\tau)$ and then with $(H - V_{\rm ref})$, we can verify that G satisfies the equation

$$\frac{\partial G}{\partial \tau} = -(\hat{H} - V_{\rm ref})G, \tag{16.76}$$

which is the same form as the imaginary time Schrödinger equation (16.65). It is easy to verify that $G(x,x',\tau) = G(x',x,\tau)$. A formal solution of (16.76) is

$$G(\tau) = e^{-(\hat{H}-V_{\rm ref})\tau}, \tag{16.77}$$

where the meaning of the exponential of an operator is given by its Taylor series expansion.

The difficulty with (16.77) is that the kinetic and potential energy operators $\hat{T}$ and $\hat{V}$ in $\hat{H}$ do not commute. For this reason, if we want to write the exponential in (16.77) as a product of two exponentials, we can only approximate the exponential for short times $\Delta\tau$. To first order in $\Delta\tau$ (higher order terms involve the commutator of $\hat{V}$ and $\hat{H}$), we have

$$G(\Delta\tau) \approx G_{\text{branch}} G_{\text{diffusion}} \tag{16.78}$$

$$= e^{-(V-V_{\text{ref}})\Delta\tau}\, e^{-\hat{T}\Delta\tau}, \tag{16.79}$$

where $G_{\text{diffusion}} \equiv e^{-\hat{T}\Delta\tau}$ and $G_{\text{branch}} \equiv e^{-(\hat{V}-V_{\text{ref}})\Delta\tau}$ correspond to the two random processes: diffusion and branching. From (16.76) we see that $G_{\text{diffusion}}$ and G_{branch} satisfy the differential equations:

$$\frac{\partial G_{\text{diffusion}}}{\partial\tau} = -\hat{T} G_{\text{diffusion}} = \frac{\hbar^2}{2m}\frac{\partial^2 G_{\text{diffusion}}}{\partial x^2} \tag{16.80}$$

$$\frac{\partial G_{\text{branch}}}{\partial\tau} = (V_{\text{ref}} - \hat{V}) G_{\text{branch}}. \tag{16.81}$$

The solutions to (16.79)–(16.81) that are symmetric in x and x' are

$$G_{\text{diffusion}}(x, x', \Delta\tau) = (4\pi D\Delta\tau)^{-1/2} e^{-(x-x')^2/4D}, \tag{16.82}$$

with $D \equiv \hbar^2/2m$, and

$$G_{\text{branch}}(x, x', \Delta\tau) = e^{-\left(\frac{1}{2}[V(x)+V(x')]-V_{\text{ref}}\right)\Delta\tau}. \tag{16.83}$$

From the form of (16.82) and (16.83), we can see that the diffusion quantum Monte Carlo method is similar to the random walk algorithm discussed in Section 16.8. An implementation of the diffusion quantum Monte Carlo method in one dimension can be summarized as follows:

1. Begin with a set of N_0 random walkers. There is no lattice so the positions of the walkers are continuous. It is advantageous to choose the walkers so that they are in regions of space where the wave function is known to be large.

2. Choose one of the walkers and displace it from x to x'. The new position is chosen from a Gaussian distribution with a variance $2D\Delta\tau$ and zero mean. This change corresponds to the diffusion process given by (16.82).

3. Weight the configuration x' by

$$w(x \to x', \Delta\tau) = e^{-\left(\frac{1}{2}[V(x)+V(x')]-V_{\text{ref}}\right)\Delta\tau}. \tag{16.84}$$

   One way to do this weighting is to generate duplicate random walkers at x'. For example, if $w \approx 2$, we would have two walkers at x' where previously there had been one. To implement this weighting (branching) correctly, we must make an integer number of copies that is equal on the average to the number w. A simple way to do so is to take the integer part of $w + r$, where r is a uniform random number in the unit interval. The number of copies can be any nonnegative integer including zero. The latter value corresponds to a removal of a walker.

4. Repeat steps 2 and 3 for all members of the ensemble, thereby creating a new ensemble at a later time $\Delta\tau$. One iteration of the ensemble is equivalent to performing the integration

$$\Psi(x,\tau) = \int G(x,x',\Delta\tau)\,\Psi(x',\tau-\Delta\tau)\,dx'. \tag{16.85}$$

5. The quantity of interest $\Psi(x,\tau)$ will be independent of the original ensemble $\Psi(x,0)$ if a sufficient number of Monte Carlo steps are taken. As before, we must ensure that $N(\tau)$, the number of walkers at time τ, is kept close to the desired number N_0.

Now we can understand how the simple random walk algorithm discussed in Section 16.8 is an approximation to the diffusion quantum MC algorithm. First, the Gaussian distribution gives the exact distribution for the displacement of a random walker in a time $\Delta\tau$, in contrast to the fixed step size in the simple random walk algorithm which gives the average displacement of a walker. Hence, there are no systematic errors due to the finite step size. Second, if we expand the exponential in (16.83) to first order in $\Delta\tau$ and set $V(x) = V(x')$, we obtain the branching rule used previously. (We use the fact that the uniform distribution r is the same as the distribution $1-r$.) However, the diffusion quantum MC algorithm is not exact because the branching is independent of the position reached by diffusion, which is only true in the limit $\Delta\tau \to 0$. This limitation is remedied in the Green's function Monte Carlo method where a short time approximation is not made (see the articles on Green's function Monte Carlo in the references).

One limitation of the two random walk methods we have discussed is that they can become very inefficient. This inefficiency is due in part to the branching process. If the potential becomes large and negative (as it is for the Coulomb potential when an electron approaches a nucleus), the number of copies of a walker will become very large. It is possible to improve the efficiency of these algorithms by introducing an importance sampling method. The idea is to use an initial guess $\Psi_T(x)$ for the wave function to guide the walkers toward the more important regions of $V(x)$. To implement this idea, we introduce the function $f(x,\tau) = \Psi(x,\tau)\Psi_T(x)$. If we calculate the quantity $\partial f/\partial t - D\,\partial^2 f/\partial x^2$, and use (16.65), we can show that $f(x,\tau)$ satisfies the differential equation:

$$\frac{\partial f}{\partial \tau} = D\frac{\partial^2 f}{\partial x^2} - D\frac{\partial\big[fF(x)\big]}{\partial x} - [E_L(x) - V_{\rm ref}]f, \tag{16.86}$$

where

$$F(x) = \frac{2}{\Psi_T}\frac{\partial \Psi_T}{\partial x}, \tag{16.87}$$

and the local energy $E_L(x)$ is given by

$$E_L(x) = \frac{\hat{H}\partial_T}{\Psi_T} = V(x) - \frac{D}{\Psi_T}\frac{\partial^2\Psi_T}{\partial x^2}. \tag{16.88}$$

The term in (16.86) containing F corresponds to a drift in the walkers away from regions where $|\Psi_T|^2$ is small (see Problem 7.43).

To incorporate the drift term into $G_{\rm diffusion}$, we replace $(x-x')^2$ in (16.82) by the term $\big(x-x'-D\Delta\tau F(x')\big)^2$, so that the diffusion propagator becomes

$$G_{\rm diffusion}(x,x',\Delta\tau) = (4\pi D\Delta\tau)^{-1/2} e^{-\left(x-x'-D\Delta\tau F(x')\right)^2/4D\Delta\tau}. \tag{16.89}$$

However, this replacement destroys the symmetry between x and x'. To restore it, we use the Metropolis algorithm for accepting the new position of a walker. The acceptance probability p is given by

$$p = \frac{|\Psi_T(x')|^2\, G_{\text{diffusion}}(x, x', \Delta\tau)}{|\Psi_T(x)|^2\, G_{\text{diffusion}}(x', x, \Delta\tau)}. \tag{16.90}$$

If $p > 1$, we accept the move; otherwise, we accept the move if $r \leq p$. The branching step is achieved by using (16.83) with $V(x) + V(x')$ replaced by $E_L(x) + E_L(x')$, and $\Delta\tau$ replaced by an effective time step. The reason for the use of an effective time step in (16.83) is that some diffusion steps are rejected. The effective time step to be used in (16.83) is found by multiplying $\Delta\tau$ by the average acceptance probability. It can be shown (see Hammond et al.) that the mean value of the local energy is an unbiased estimator of the ground state energy.

Another possible improvement is to periodically replace branching (which changes the number of walkers) with a weighting of the walkers. At each weighting step, each walker is weighted by G_{branch}, and the total number of walkers remains constant. After n steps, the kth walker receives a weight $W_k = \Pi_{i=1}^{n} G_{\text{branch}}^{(i,k)}$, where $G_{\text{branch}}^{(i,k)}$ is the branching factor of the kth walker at the ith time step. The contribution to any average quantity of the kth walker is weighted by W_k.

Problem 16.29. Diffusion Quantum Monte Carlo

a. Modify `QMWalkApp` to implement the diffusion quantum Monte Carlo method for the systems considered in Problems 16.26 or 16.27. Begin with $N_0 = 100$ walkers and $\Delta\tau = 0.01$. Use at least three values of $\Delta\tau$ and extrapolate your results to $\Delta\tau \to 0$. Reasonable results can be obtained by adjusting the reference energy every 20 Monte Carlo steps with $a = 0.1$.

b. Write a program to apply the diffusion quantum Monte Carlo method to the hydrogen atom. In this case a configuration is represented by three coordinates.

c.* Modify your program to include weights in addition to changing walker populations. Redo part (a) and compare your results.

***Problem 16.30.** Importance sampling

a. Derive the partial differential equation (16.86) for $f(x, \tau)$.

b. Modify `QMWalkApp` to implement the diffusion quantum Monte Carlo method with importance sampling. Consider the harmonic oscillator problem with the trial wave function $\Psi_T = e^{-\lambda x^2}$. Compute the statistical error associated with the ground state energy as a function of λ. How much variance reduction can you achieve relative to the naive diffusion quantum Monte Carlo method? Then consider another form of Ψ_T that does not have a form identical to the exact ground state. Try the hydrogen atom with $\Psi_T = e^{-\lambda r}$.

16.10 Path Integral Quantum Monte Carlo

The Monte Carlo methods we have discussed so far are primarily useful for estimating the ground state energy and wave function, although it also is possible to find the first few excited states with some effort. In this section we discuss a Monte Carlo method that is of particular interest for computing the thermal properties of quantum systems.

We recall (see Section 7.10) that classical mechanics can be formulated in terms of the principle of least action. That is, given two points in space-time, a classical particle chooses the path that minimizes the action given by

$$S = \int_{x_0,0}^{x,t} L\, dt. \tag{16.91}$$

The Lagrangian L is given by $L = T - V$. Quantum mechanics also can be formulated in terms of the action (cf. Feynman and Hibbs). The result of this *path integral* formalism is that the real-time propagator G can be expressed as

$$G(x, x_0, t) = A \sum_{\text{paths}} e^{iS/\hbar}, \tag{16.92}$$

where A is a normalization factor. The sum in (16.92) is over all paths between $(x_0, 0)$ and (x, t), not just the path that minimizes the classical action. The presence of the imaginary number i in (16.92) leads to interference effects. As before, the propagator $G(x, x_0, t)$ can be interpreted as the probability amplitude for a particle to be at x at time t given that it was at x_0 at time zero. G satisfies the equation (see (16.75))

$$\Psi(x, t) = \int G(x, x_0, t)\Psi(x_0, 0)\, dx_0 \qquad (t > 0). \tag{16.93}$$

Because G satisfies the same differential equation as Ψ in both x and x_0, G can be expressed as

$$G(x, x_0, t) = \sum_n \phi_n(x)\phi_n(x_0)e^{-iE_n t/\hbar}, \tag{16.94}$$

where the ϕ_n are the eigenstates of H. For simplicity, we set $\hbar = 1$ in the following. As before, we substitute $\tau = it$ into (16.94), and obtain

$$G(x, x_0, \tau) = \sum_n \phi_n(x)\phi_n(x_0)\, e^{-\tau E_n}. \tag{16.95}$$

We first consider the ground state. In the limit $\tau \to \infty$, we have

$$G(x, x, \tau) \to \phi_0(x)^2\, e^{-\tau E_0}. \qquad (\tau \to \infty) \tag{16.96}$$

From the form of (16.96) and (16.92), we see that we need to compute G and hence S to estimate the properties of the ground state.

To compute S, we convert the integral in (16.91) to a sum. The Lagrangian for a single particle of unit mass in terms of τ becomes

$$L = -\frac{1}{2}\Big(\frac{dx}{d\tau}\Big)^2 - V(x) = -E. \tag{16.97}$$

We divide the imaginary time interval τ into N equal steps of size $\Delta\tau$ and write E as

$$E(x_j,\tau_j) = \frac{1}{2}\frac{(x_{j+1}-x_j)^2}{(\Delta\tau)^2} + V(x_j), \tag{16.98}$$

where $\tau_j = j\Delta\tau$, and x_j is the corresponding displacement. The action becomes

$$S = -i\Delta\tau\sum_{j=0}^{N-1} E(x_j,\tau_j) = -i\Delta\tau\Big[\sum_{j=0}^{N-1}\frac{1}{2}\frac{(x_{j+1}-x_j)^2}{(\Delta\tau)^2} + V(x_j)\Big], \tag{16.99}$$

and the probability amplitude for the path becomes

$$e^{iS} = e^{\Delta\tau[\sum_{j=0}^{N-1}\frac{1}{2}(x_{j+1}-x_j)^2/(\Delta\tau)^2+V(x_j)]}. \tag{16.100}$$

Hence, the propagator $G(x,x_0,N\Delta\tau)$ can be expressed as

$$G(x,x_0,N\Delta\tau) = A\int dx_1\cdots dx_{N-1}\, e^{\Delta\tau[\sum_{j=0}^{N-1}\frac{1}{2}(x_{j+1}-x_j)^2/(\Delta\tau)^2+V(x_j)]}, \tag{16.101}$$

where $x \equiv x_N$ and A is an unimportant constant.

From (16.101) we see that $G(x,x_0,N\Delta\tau)$ has been expressed as a multidimensional integral with the displacement variable x_j associated with the time τ_j. The sequence $x_0,x_1,\cdots,x_N$ defines a possible path, and the integral in (16.101) is over all paths. Because the quantity of interest is $G(x,x,N\Delta\tau)$ (see (16.96)), we adopt the periodic boundary condition, $x_N = x_0$. The choice of x in the argument of G is arbitrary for finding the ground state energy, and the use of the periodic boundary conditions implies that no point in the closed path is unique. It is thus possible (and convenient) to rewrite (16.101) by letting the sum over j go from 1 to N:

$$G(x_0,x_0,N\Delta\tau) = A\int dx_1\cdots dx_{N-1}\, e^{-\Delta\tau[\sum_{j=1}^{N}\frac{1}{2}(x_j-x_{j-1})^2/(\Delta\tau)^2+V(x_j)]}, \tag{16.102}$$

where we have written x_0 instead of x because the x_j that is not integrated over is $x_N = x_0$.

The result of this analysis is to convert a quantum mechanical problem for a single particle into a statistical mechanics problem for N "atoms" on a ring connected by nearest neighbor "springs" with spring constant $1/(\Delta\tau)^2$. The label j denotes the order of the atoms in the ring.

Note that the form of (16.102) is similar to the form of the Boltzmann distribution. Because the partition function for a single quantum mechanical particle contains terms of the form $e^{-\beta E_n}$ and (16.95) contains terms proportional to $e^{-\tau E_n}$, we make the correspondence $\beta = \tau = N\Delta\tau$. We shall see in the following how we can use this identity to simulate a quantum system at a finite temperature.

We can use the Metropolis algorithm to simulate the motion of N "atoms" on a ring. Of course, these atoms are a product of our analysis just as were the random walkers we introduced in diffusion Monte Carlo and should not be confused with real particles. A possible path integral algorithm can be summarized as follows:

1. Choose N and $\Delta\tau$ such that $N\Delta\tau >> 1$ (the zero temperature limit). Also choose δ, the maximum trial change in the displacement of an atom, and `mcs`, the total number of Monte Carlo steps per atom.

2. Choose an initial configuration for the displacements x_j that is close to the approximate shape of the ground state probability amplitude.

3. Choose an atom j at random and a trial displacement $x_j \to x_j + (2r-1)\delta$, where r is a uniform random number in the unit interval. Compute the change ΔE in the energy E, where ΔE is given by

$$\Delta E = \frac{1}{2}\Big[\frac{x_{j+1}-x_j}{\Delta\tau}\Big]^2 + \frac{1}{2}\Big[\frac{x_j-x_{j-1}}{\Delta\tau}\Big]^2 + V(x_j) - \frac{1}{2}\Big[\frac{x_{j+1}-x_j}{\Delta\tau}\Big]^2 - \frac{1}{2}\Big[\frac{x_j-x_{j-1}}{\Delta\tau}\Big]^2 - V(x_j) \tag{16.103}$$

   If $\Delta E < 0$, accept the change; otherwise, compute the probability $p = e^{-\Delta\tau\Delta E}$ and a random number r in the unit interval. If $r \leq p$, then accept the move; otherwise reject the trial move.

4. Divide the possible x values into equal size bins of width Δx. Update $P(x)$, that is, let $P(x = x_j) \to P(x = x_j) + 1$, where x is the displacement of the atom chosen in step 3 after step 3 is completed. Do this update even if the trial move was rejected.

5. Repeat steps 3 and 4 until a sufficient number of Monte Carlo steps per atom has been obtained. (Do not take data until the memory of the initial path is lost and the system has reached "equilibrium.")

Normalize the probability density $P(x)$ by dividing by the product of N and `mcs`. The ground state energy E_0 is given by

$$E_0 = \sum_x P(x)[T(x) + V(x)], \tag{16.104}$$

where $T(x)$ is the kinetic energy as determined from the virial theorem,

$$\Big\langle 2T(x)\Big\rangle = \Big\langle x\frac{dV}{dx}\Big\rangle, \tag{16.105}$$

which is discussed in many texts (see Griffiths for example). It also is possible to compute T from averages over $(x_j - x_{j-1})^2$, but the virial theorem yields a smaller variance. The ground state wave function $\phi(x)$ is obtained from the normalized probability $P(x)\Delta x$ by dividing by Δx and taking the square root.

We also can find the thermodynamic properties of a particle that is connected to a heat bath at temperature $T = 1/\beta$ by not taking the $\beta = N\Delta\tau \to \infty$ limit. To obtain the ground state, which corresponds to the zero temperature limit ($\beta >> 1$), we had to make $N\Delta\tau$ as large as possible. However, we need $\Delta\tau$ to be as small as possible to approximate the continuum time limit. Hence, to obtain the ground state we need a large number of time intervals N. For the finite temperature simulation, we can use smaller values of N for the same level of accuracy as the zero temperature simulation.

The path integral method is very flexible and can be generalized to higher dimensions and many mutually interacting particles. For three dimensions, x_j is replaced by the three-dimensional displacement $\mathbf{r}_j$. Each real particle is represented by a ring of N "atoms" with a spring-like potential connecting each atom within a ring. Each atom in each ring also interacts with the

atoms in the other rings through an interparticle potential. If the quantum system is a fluid where indistinguishability is important, then we must consider the effect of exchange by treating the quantum system as a classical polymer system where the "atoms" represent the monomers of a polymer, and where polymers can split up and reform. Chandler and Wolynes discuss how the quantum mechanical effects due to exchanging identical particles can be associated with the chemical equilibrium of the polymers. They also discuss Bose condensation using path integral techniques.

Problem 16.31. Path integral calculation

a. Write a program to implement the path integral algorithm for the one-dimensional harmonic oscillator potential with $V(x) = x^2/2$. Use the structure of your Monte Carlo Lennard-Jones program from Chapter 15 as a guide.

b. Let $N\Delta\tau = 15$ and consider $N = 10$, 20, 40, and 80. Equilibrate for at least 2000 Monte Carlo steps per atom and average over at least 5000 mcs. Compare your results with the exact result for the ground state energy given by $E_0 = 0.5$. Estimate the equilibration time for your calculation. What is a good initial configuration? Improve your results by using larger values of $N\Delta\tau$.

c. Find the mean energy, $\langle E \rangle$, of the harmonic oscillator at the temperature T determined by $\beta = N\Delta\tau$. Find $\langle E \rangle$ for $\beta = 1$, 2, and 3, and compare it with the exact result $\langle E \rangle = \frac{1}{2}\coth(\beta/2)$.

d. Repeat the above calculations for the Morse potential $V(x) = 2(1 - e^{-x})^2$.

16.11 Projects

Many of the techniques described in this chapter can be extended to two-dimensional quantum systems. The `Complex2DFrame` tool in the frames package is designed to show two-dimensional complex scalar fields such as quantum wave functions. Listing 16.13 in Appendix A shows how this class is used to show a two-dimensional Gaussian wave packet with a momentum boost.

Project 16.32. Separable systems in two dimensions

The shooting method is inappropriate for the calculation of eigenstates and eigenvalues in two or more dimensions with arbitrary potential energy functions, $V(\mathbf{r})$. However, the special case of separable potentials can be reduced to several one-dimensional problems that can be solved using the numerical methods described in this chapter. Many molecular modeling programs use the Hartree-Fock self-consistent field approximation to model non-separable systems as a set of one-dimensional problems. Recently, there has been significant progress motivated by a molecular dynamics algorithm developed by Car and Parrinello.

Write a two-dimensional eigenstate class, `Eigenstate2d`, that calculates eigenstates and eigenvalues for a separable potential of the form:

$$V(x, y) = V_1(x) + V_2(y). \tag{16.106}$$

Test this class using the known analytic solutions for the two-dimensional rectangular box and two-dimensional harmonic oscillator. Use this class to model the evolution of superposition states. Under what conditions are there wave function revivals?

Project 16.33. Excited state wave functions using quantum Monte Carlo

Quantum Monte Carlo methods can be extended to compute the excited state wave functions using a Gram-Schmidt procedure to insure that each excited state is orthogonal to all lower lying states (see Roy et al.). A quantum Monte Carlo method is used to compute the ground state wave function. A trial wave function for the first exited state is then selected and the ground state component is subtracted from the trial wave function. This subtraction is repeated after every iteration of the Monte Carlo algorithm. Because excited states decay with a time constant $e^{-(E_j - E_0)}$, the lowest remaining excited state dominates the remaining wave function. After the first excited state is obtained, the second excited state is computed by subtracting both known states from the trial wave function. This process is repeated to obtain additional wave functions.

Implement this procedure to find the first few excited state wave functions for the one-dimensional harmonic oscillator. Then consider the one-dimensional double well oscillator

$$V(x) = -\frac{1}{2}kx^2 + a_3x^3 + a_4x^4, \tag{16.107}$$

with $k = 40$, $a_3 = 1$, and $a_4 = 1$.

Project 16.34. Quantum Monte Carlo in two dimensions

The procedure described in Project 16.33 can be used to compute two-dimensional wave functions (see Roy et al.).

a. Test your program using a separable two-dimensional double-well potential.

b. Find the first few excited states for the two-dimensional double-well potential

$$V(x,y) = -\frac{1}{2}k_xx^2 - \frac{1}{2}k_yy^2 + \frac{1}{2}(a_{xx}x^4 + 2a_{xy}x^2y^2 + a_{yy}y^4), \tag{16.108}$$

with $k_x = k_y = 20$ and $a_{xx} = a_{yy} = a_{xy} = 5$. Repeat with $k_x = k_y = 20$ and $a_{xx} = a_{yy} = a_{xy} = 1$.

Project 16.35. Evolution of a wave packet in two dimensions

Both the half-step and split operator algorithms can be extended to model the evolution of two-dimensional systems with arbitrary potentials, $V(x,y)$. (See *Numerical Recipes* for how the FFT algorithm is extended to more dimensions.) Implement either algorithm and model a wave packet scattering from a central barrier and a wave packet passing through a double slit.

A clever way to insure stability in the half-step algorithm is to use a boolean array to tag grid locations where the solution becomes unstable and to set the wave function to zero at these grid points.

```
double minV = -2/dt;
double maxVx = 2/dt-2/(dx*dx);
double maxVy = 2/dt-2/(dy*dy);
```

```
double maxV = Math.min(maxVx,maxVy);
for(int i = 0, n = potential.length; i <= n; i++) {
   for(int j = 0, m = potential[0].length; j <= m; j++) {
      if (potential[i][j] >= minV && potential[i][j] <= maxV) // stable
         stable[i][j] = true;  // stable
      else
         stable[i][j] = false; // unstable, set wave function to zero
       }
    }
}
```

Project 16.36. Two particle system

Rubin Landau has studied the time dependence of two particles interacting in one dimension with a potential that depends on their relative separation:

$$V(x_1, x_2) = V_0 e^{-(x_1-x_2)^2/2\alpha^2}. \tag{16.109}$$

Model a scattering experiment for particles having momentum p_1 and p_2 by assuming the following (unnormalized) initial wave function

$$\Psi(x_1, x_2) = e^{ip_1x_1}\, e^{-(x_1-a)^2/4w^2}\, e^{ip_2x_2}\, e^{-(x_2-a)^2/4w^2}, \tag{16.110}$$

where $2a$ is the separation and w is the variance in each particle's position. Do the particles bounce off of each other when the interaction is repulsive? What happens when the interaction is attractive?

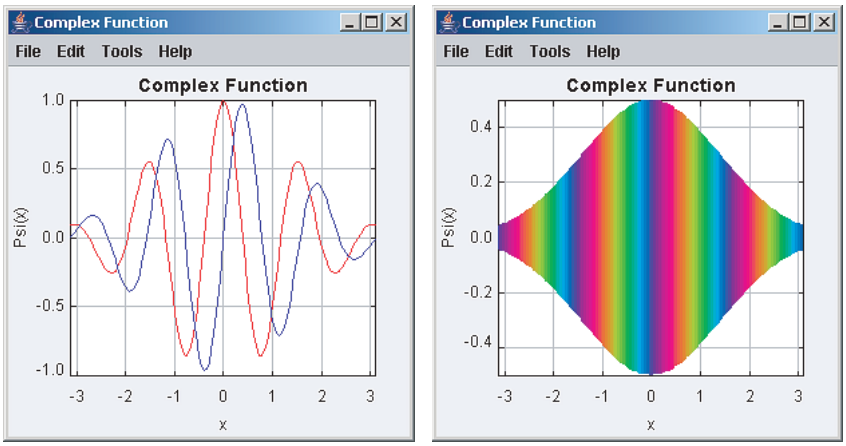
Appendix 16A: Visualizing Complex Functions

Complex functions are essential in quantum mechanics and the frames package contains classes for displaying and analyzing these functions. Listing 16.12 uses a `ComplexPlotFrame` to display a one-dimensional wave function.

Listing 16.12: The `ComplexPlotFrameApp` class displays a one-dimensional Gaussian wave packet with a momentum boost.

```
package org.opensourcephysics.sip.ch16;
import org.opensourcephysics.frames.ComplexPlotFrame;

public class ComplexPlotFrameApp {
   public static void main(String[] args) {
      ComplexPlotFrame frame = new ComplexPlotFrame("x", "Psi(x)", "Complex function");
      int n = 128;
      double
         xmin = -Math.PI, xmax = Math.PI;
      double
         x = xmin, dx = (xmax-xmin)/n;
      double[] xdata = new double[n];
      double[] zdata = new double[2*n]; // real and imaginary values alternate
```



(a) Real and imaginary. (b) Amplitude and phase.

Figure 16.2: Two representations of complex wave functions. (The actual output is in color.)

```
        int mode = 4;                          // test function is e^(-x*x/4)e^(i*mode*x) for x=[-p
        for(int i = 0;i<n;i++) {
            double a = Math.exp(-x*x/4);
            zdata[2*i] = a*Math.cos(mode*x);
            zdata[2*i+1] = a*Math.sin(mode*x);
            xdata[i] = x;
            x += dx;
        }
        frame.append(xdata, zdata);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
    }
}
```

Figure 16.2 shows two representations of a quantum wave function. The real and imaginary representation displays the real and imaginary parts of the wave function $\Psi(x)$ by drawing two curves. In the amplitude and phase representation the vertical height represents the wave function magnitude and the color indicates phase. Note that the complex phase is oscillating, indicating that the wave function has a nonzero momentum expectation value, which is known as a *momentum boost*.

Wave function visualizations can be selected at runtime using the `Tools` menu or they can be selected programmatically using convert methods such as `convertToPostView` and `convertToReImView`. The `Tools` menu also allows the user to select a table view to examine the data being used to draw the wave function and to display a phase legend that shows the color to phase relation.

A `Complex2DFrame` displays a two-dimensional complex scalar field such as a two-dimensional wave function. We instantiate a `Complex2DFrame` and then pass to it a multi-dimensional array containing the field's real and imaginary components. Listing 16.13 shows how this class is used

to show a two-dimensional Gaussian wave packet with a momentum boost.

Listing 16.13: The `Complex2DFrameApp` program displays a two-dimensional Gaussian wave packet with a momentum boost.

```
package org.opensourcephysics.sip.ch16;
import org.opensourcephysics.frames.Complex2DFrame;

public class Complex2DFrameApp {
   public static void main(String[] args) {
      Complex2DFrame frame = new Complex2DFrame("x", "y", "Complex field");
      frame.setPreferredMinMax(-1.5, 1.5, -1.5, 1.5);
      double[][][] field = new double[2][32][32]; // components of field
      frame.setAll(field);
      for(int i = 0, nx = field[0].length;i<nx;i++) {
         double x = frame.indexToX(i);
         for(int j = 0, ny = field[0][0].length;j<ny;j++) {
            double y = frame.indexToY(j);
            double a = Math.exp(-4*(x*x+y*y));
            field[0][i][j] = a*Math.cos(5*x); // real component
            field[1][i][j] = a*Math.sin(5*x); // complex component
         }
      }
      frame.setAll(field);
      frame.setVisible(true);
      frame.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
   }
}
```

The complex field is computed on a n row by m column grid and stored in an array with dimensions $2 \times m \times n$. The default visualization uses a grid in which every cell is colored using brightness to show the complex number's magnitude and color to show phase. Other visualizations can be programmed or selected at runtime using the menu.

References and Suggestions for Further Reading

The ALPS project, `<http://alps.comp-phys.org/>`, has open source simulation programs for strongly correlated quantum mechanical systems and C++ libraries for simplifying the development of such code. Although most of the code is beyond the level of this text, this open source project is another example of software for use in both research and education.

J. B. Anderson, "A random walk simulation of the Schrödinger equation: H_3^+," J. Chem. Phys. **63**, 1499–1503 (1975); "Quantum chemistry by random walk. H ^{2}P, H_3^+ D_{3h} $^1A'_1$, H_2 $^3\Sigma_u^+$, H_4 $^1\Sigma_g^+$, Be ^{1}S," J. Chem. Phys. **65**, 4121–4127 (1976); "Quantum chemistry by random walk: Higher accuracy," J. Chem. Phys. **73**, 3897–3899 (1980). These papers describe the random walk method, extensions for improved accuracy, and applications to simple molecules.

G. Baym, *Lectures on Quantum Mechanics*, Westview Press (1990). A discussion of the Schrödinger equation in imaginary time is given in Chapter 3.

M. A. Belloni, W. Christian, and A. Cox, *Physlet Quantum Physics*, Prentice Hall (2006) This book contains interactive exercises using Java applets to visualize quantum phenomena.

H. A. Bethe, *Intermediate Quantum Mechanics*, Westview Press (1997). Applications of quantum mechanics to atomic systems are discussed.

Jay S. Bolemon, "Computer solutions to a realistic 'one-dimensional' Schrödinger equation," Am. J. Phys. **40**, 1511 (1972).

Siegmund Brandt and Hans Dieter Dahmen, *The Picture Book of Quantum Mechanics*, third edition, Springer-Verlag (2001); Siegmund Brandt, Hans Dieter Dahmen, and Tilo Stroh, *Interactive Quantum Mechanics*, Springer-Verlag (2003).These books show computer generated pictures of quantum wave functions in different contexts.

R. Car and M. Parrinelli, "Unified approach for molecular dynamics and density-functional theory," Phys. Rev. Lett. **55**, 2471 (1985).

David M. Ceperley, "Path integrals in the theory of condensed helium," Rev. Mod. Phys. **67**, 279–355 (1995).

David M. Ceperley and Berni J. Alder, "Quantum Monte Carlo," Science **231**, 555 (1986). A survey of some of the applications of quantum Monte Carlo methods to physics and chemistry.

David Chandler and Peter G. Wolynes, "Exploiting the isomorphism between quantum theory and classical statistical mechanics of polyatomic fluids," J. Chem. Phys. **74** 4078–4095 (1981). The authors use path integral techniques to look at multiparticle quantum systems.

D. F. Coker and R. O. Watts, "Quantum simulation of systems with nodal surfaces," Mol. Phys. **58**, 1113–1123 (1986).

Jim Doll and David L. Freeman, "Monte Carlo methods in chemistry," Computing in Science and Engineering **1** (1), 22–32 (1994).

Robert M. Eisberg and Robert Resnick, *Quantum Physics*, second edition, John Wiley & Sons (1985). See Appendix G for a discussion of the numerical solution of Schrödinger's equation.

R. P. Feynman, "Simulating physics with computers," Int. J. Theor. Phys. **21**, 467–488 (1982). A provocative discussion of the intrinsic difficulties of simulating quantum systems. See also R. P. Feynman, *Feynman Lectures on Computation*, Westview Press (1996).

Richard P. Feynman and A. R. Hibbs, *Quantum Mechanics and Path Integrals*, McGraw-Hill (1965).

David J. Griffiths, *Introduction to Quantum Mechanics*, second edition, Prentice Hall (2005). An excellent undergraduate text that discusses the virial theorem in several problems.

B. L. Hammond, W. A. Lester Jr., and P. J. Reynolds, *Monte Carlo Methods in Ab Initio Quantum Chemistry*, World Scientific (1994). An excellent book on quantum Monte Carlo methods.

Steven E. Koonin and Dawn C. Meredith, *Computational Physics*, Addison-Wesley (1990). Solutions of the time-dependent Schrödinger equation are discussed in the context of parabolic partial differential equations in Chapter 7. Chapter 8 discusses Green's function Monte Carlo methods.

Rubin Landau, "Two-particle Schrödinger equation animations of wavepacket-wavepacket scattering," Am. J. Phys. **68** (12), 1113–1119 (2000).

Michel Le Bellac, Fabrice Mortessagne, and G. George Batrouni, *Equilibrium and Non-Equilibrium Statistical Thermodynamics*, Cambridge University Presss (2004). Chapter 7 of this graduate level text discusses the world line algorithm for bosons and fermions on a lattice.

M. A. Lee and K. E. Schmidt, "Green's function Monte Carlo," Computers in Physics **6** (2), 192 (1992). A short and clear explanation of Green's function Monte Carlo.

P. K. MacKeown, "Evaluation of Feynman path integrals by Monte Carlo methods," Am. J. Phys. **53**, 880—885 (1985). The author discusses projects suitable for an advanced undergraduate course. Also see P. K. MacKeown and D. J. Newman, *Computational Techniques in Physics*, Adam Hilger (1987).

Jean Potvin, "Computational quantum field theory. Part II: Lattice gauge theory," Computers in Physics **8**, 170 (1994) and "Computational quantum field theory," Computers in Physics **7**, 149 (1993).

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical Recipes*, second edition, Cambridge University Press (1992). The numerical solution of the time-dependent Schrödinger equation is discussed in Chapter 19.

Peter J. Reynolds, David M. Ceperley, Berni J. Alder, and William A. Lester Jr., "Fixed-node quantum Monte Carlo for molecules," J. Chem. Phys. **77**, 5593–5603 (1982). This paper describes a random walk algorithm for use in molecular applications including importance sampling and the treatment of Fermi statistics.

P. J. Reynolds, J. Tobochnik, and H. Gould, "Diffusion quantum Monte Carlo," Computers in Physics **4** (6), 882 (1990).

U. Rothlisberger, "15 Years of Car-Parrinello simulations in physics, chemistry and biology," in *Computational Chemistry: Reviews of Current Trends*, edited by Jerzy Leszczynski, World Scientific (2001), Vol. 6.

Amlan K. Roy, Neetu Gupta, and B. M. Deb, "Time-dependent quantum mechanical calculation of ground and excited states of anharmonic and double-well oscillators," Phys. Rev A **65**, 012109-1–7 (2001).

Amlan K. Roy, Ajit J. Thakkar, and B. M. Deb, "Low-lying states of two-dimensional double-well potentials," J. Phys. A **38**, 2189–2199 (2005).

K. E. Schmidt, Parhat Niyaz, A. Vaught, and Michael A. Lee, "Green's function Monte Carlo method with exact imaginary-time propagation," Phys. Rev. E **71**, 016707-1–17 (2005).

Bernd Thaller, *Visual Quantum Mechanics: Selected Topics with Computer-Generated Animations of Quantum-Mechanical Phenomena*, Telos (2000); Bernd Thaller, *Advanced Visual Quantum Mechanics*, Springer (2005).

J. Tobochnik, H. Gould, and K. Mulder, "An introduction to quantum Monte Carlo," Computers in Physics **4** (4), 431 (1990). An explanation of the path integral method applied to one particle.

P. B. Visscher, "A fast explicit algorithm for the time-dependent Schrödinger equation," Computers in Physics **5** (6), 596 (1991).