Jan Tobochnik, Harvey Gould, and Jon Machta, "Understanding the temperature and the chemical potential through computer simulations," Am. J. Phys. **73** (8), 708–716 (2005). This paper extends the demon algorithm to compute the chemical potential.

Simon Trebst, David A. Huse, and Matthias Troyer, "Optimizing the ensemble for equilibration in broad-histogram Monte Carlo simulations," Phys. Rev. E **70**, 046701-1–5 (2004). The adaptive algorithm presented in this paper overcomes critical slowing down and improves upon the Wang–Landau algorithm and is another example of the flexibility of Monte Carlo algorithms.

I. Vattulainen, T. Ala-Nissila, and K. Kankaala, "Physical tests for random numbers in simulations," Phys. Rev. Lett. **73**, 2513 (1994).

B. Widom, "Some topics in the theory of fluids," J. Chem. Phys. **39**, 2808–2812 (1963). This paper discusses the insertion method for calculating the chemical potential.

CHAPTER

16 Quantum Systems

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 Sections 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

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

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() {
      // zeros wave function
      for(int i = 0;i<numberOfPoints;i++) {
         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 wave function value
         x[i] = state[2];
         solver.step();                  // steps Schroedinger equation
         // checks for diverging solution
```



```
         if(Math.abs(state[0])>1.0e9) {
            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;
   }

   // potential is nonzero for x > 0
   public double evaluatePotential(double x) {
      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 is also 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 analytic 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 *wave packet.*

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

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

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

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 the boundary condition $\phi(x = +a) = 0$. 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 analytic 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 needed 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 Figure 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 was 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?

(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

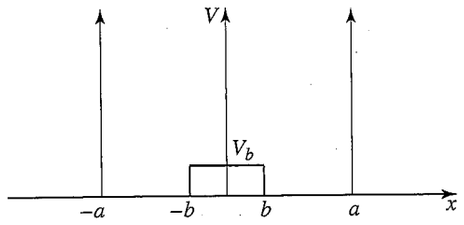

Figure 16.1 An infinite square well with a potential bump of height V_b in the middle.

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 found?

(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 analytic 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

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

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 number of elements.");
      }
      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];
      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 function");
   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, `BoxSuperpositionApp` uses a `ComplexPlotFrame` for plotting. `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 wave function 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 wave function 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 are the periods of the ground state and the first excited state wave functions? What are the periods for the probability densities?

(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

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

■

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 densities propagate 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

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

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:

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

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

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

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

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:

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

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

An implementation of (16.33) is given in the `TDHalfStep` class in Listing 16.7. The real part of the wave function is updated first 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;

   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])/dx2;
         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++) {
         double imH = potential[i]*imagPsi[i]-0.5*(imagPsi[i+1]-
               2*imagPsi[i]+imagPsi[i-1])/dx2;
         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])/dx2;
         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:

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

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

   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);
       // multiple computations per animation step
       setStepsPerDisplay(10);
       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 a 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 is also 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