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);
         }
      }
```