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
   }
```