fine grid, then $\{2i, 2j\}$ represents the positions of the coarse grid sites. The fine grid sites that are at the same position as a coarse grid point are assigned the value of the potential of the corresponding coarse grid point. The fine grid sites that have two coarse grid points as nearest neighbors are assigned the average value of these two coarse grid sites. The other fine grid sites have four coarse grid sites as next nearest neighbors and are assigned the average value of these four coarse grid sites. This prescription specifies how values on the fine grid are computed using the values on the coarse grid.

In the full weighting prolongation method, each coarse grid site receives one fourth of the potential of the fine grid site at the same position, one eighth of the potential for the four nearest neighbor sites of the fine grid, and one sixteenth of the potential for the four next nearest neighbor points of the fine grid. The sum of these fractions, $1/4 + 4(1/8) + 4(1/16)$, adds up to unity. An alternative procedure, known as half weighting, ignores the next nearest neighbors and uses one half of the potential of the fine grid site at the same position as the coarse grid site.

(a) Write a program that implements the multigrid method using Gauss–Seidel relaxation on a checkerboard lattice (see Problem 10.11b). In its simplest form, the program should allow the user to intervene and decide whether to go to a finer or coarser grid or to remain at the same level for the next relaxation step. Have the program print the potential at each site of the current level after each relaxation step. Test your program on a 4×4 grid whose boundary sites are all equal to unity and whose initial internal sites are set to zero. Make sure that the boundary sites of the coarser grids are also set to unity.

(b) The exact solution for part (a) gives a potential of unity at each point. How many relaxation steps does it take to reach unity within 0.1% at every site by simply using the 4×4 grid? How many steps does it take if you use one coarse grid and continue until the coarse grid values are within 0.1% of unity? Is it necessary to carry out any fine grid relaxation steps to reach the desired accuracy on the fine grid? Next start with the coarsest scale, which is just one site. How many relaxation steps does it take now?

(c) Repeat part (b) but change the boundary so that one side of the boundary is held at a potential of 0.5. Experiment with different sequences of prolongation, restriction, and relaxation.

(d) Assume that the boundary points alternate between zero and unity and repeat part (b). Does the multigrid method work? Should one go up and down in levels many times instead of staying at the coarsest level and then going down to the finest level? ■

APPENDIX 10A: VECTOR FIELDS

The frames package contains the `Vector2DFrame` class for displaying two-dimensional vector fields. To use this class we instantiate a multidimensional array to store components of the vector. The first array index indicates the component, the second index indicates the column or x-position, and the third index indicates the row or y-position. The vectors in the visualization are set by passing the data array to the frame using the `setAll` method. The

program in Listing 10.10 demonstrates how this is done by displaying the electric field of a unit charge located at the origin.

Listing 10.10 A vector field test program.

```
package org.opensourcephysics.sip.ch10;
import javax.swing.JFrame;
import org.opensourcephysics.frames.Vector2DFrame;

public class VectorPlotApp {
   public static void main(String[] args) {
      Vector2DFrame frame =
           new Vector2DFrame("x", "y", "Vector field");
      double a = 2; // half width of frame in world coordinates
      frame.setPreferredMinMax(-a, a, -a, a);
      int nx = 15, ny = 15; // grid sizes in x and y direction
      // generate sample data
      double[][][] vectorField = new double[2][nx][ny];
      frame.setAll(vectorField); // vector field displays zero data
      for(int i = 0;i<nx;i++) {
         double x = frame.indexToX(i);
         for(int j = 0;j<ny;j++) {
            double y = frame.indexToY(j);
            double r2 = x*x+y*y;            // distance squared
            double r3 = Math.sqrt(r2)*r2; // distance cubed
            vectorField[0][i][j] = (r2==0) ? 0 : x/r3; // x component
            vectorField[1][i][j] = (r2==0) ? 0 : y/r3; // y component
         }
      }
      frame.setAll(vectorField); // vector field displays new data
      frame.setVisible(true);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
}
```

The arrows in the visualization have a fixed length that is chosen to fill the viewing area. The arrow's color represents the field's magnitude. We have found that using an arrow's color rather than its length to represent field strength produces a more effective representation of vector fields over a wider dynamic range. The frame's Legend menu item under Tools shows this mapping. The appropriate representation of vector fields is an active area of interest.

Problem 10.27 Gradient of a scalar field

The gradient of a scalar field, $A(x, y)$, defines a vector field. In a two-dimensional Cartesian coordinate system, the components of the gradient are equal to the derivative of the scalar field along the x- and y-axes, respectively:

$$\nabla A = \frac{\partial A}{\partial x}\hat{\mathbf{x}} + \frac{\partial A}{\partial y}\hat{\mathbf{y}}. \tag{10.60}$$

Write a short program that displays both a scalar field and its gradient. (Hint: Define a function and use numerical derivatives along the rows and columns.) Create separate frames for the scalar and vector field visualizations. *Open Source Physics: A User's Guide with Examples* describes how a vector field visualization can be superimposed on a scalar field visualization. ■