The effect of including higher derivatives is discussed by MacDonald (see references). ■

Now that we have found that (10.12), a finite difference form of Laplace's equation, is consistent with Coulomb's law, we adopt (10.12) as the basis for computing the potential for systems for which we cannot calculate the potential directly. In particular, we consider problems where the potential is specified on a closed surface that divides space into interior and exterior regions in which the potential is independently determined. For simplicity, we consider only two-dimensional geometries. The approach, known as the *relaxation method*, is based on the following algorithm:

1. Divide the region of interest into a rectangular grid spanning the region. The region is enclosed by a surface (curve in two dimensions) with specified values of the potential along the curve.
2. Assign to a boundary site the potential of the boundary nearest the site.
3. Assign all interior sites an arbitrary potential (preferably a reasonable guess).
4. Compute new values for the potential V for each interior site. Each new value is obtained by finding the average of the previous values of the potential at the four nearest neighbor sites.
5. Repeat step (4) using the values of V obtained in the previous iteration. This iterative process is continued until the potential at each interior site is computed to the desired accuracy.

The program shown in Listing 10.5 implements this algorithm using a grid of voltages and a boolean grid to signal the presence of a conductor.

Listing 10.5 The `LaplaceApp` program solves the Laplace equation using the relaxation method.

```
package org.opensourcephysics.sip.ch10;
import java.awt.event.*;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.display.*;
import org.opensourcephysics.display2d.*;
import org.opensourcephysics.frames.*;

public class LaplaceApp extends AbstractSimulation implements
      InteractiveMouseHandler {
   Scalar2DFrame frame = new Scalar2DFrame("x", "y",
     "Electric potential");
   boolean[][] isConductor;
   double[][] potential; // electric potential
   double maximumError;
   int gridSize;         // number of sites on side of grid

   public LaplaceApp() {
      frame.setInteractiveMouseHandler(this);
   }

   public void initialize() {
      maximumError = control.getDouble("maximum error");
```



```
      gridSize = control.getInt("");
      initArrays();
      frame.setVisible(true);
      frame.showDataTable(true); // show the data table
   }

   public void initArrays() {
      isConductor = new boolean[gridSize][gridSize];
      potential = new double[gridSize][gridSize];
      frame.setPaletteType(ColorMapper.DUALSHADE);
      // isConductor array is false by default
      // voltage in potential array is 0 by default
      for(int i = 0;i<gridSize;i++) {       // initialize the sides
         isConductor[0][i] = true;          // left boundary
         isConductor[gridSize-1][i] = true; // right boundary
         isConductor[i][0] = true;          // bottom boundary
         isConductor[i][gridSize-1] = true; // top boundary
      }
      // set potential on inner conductor
      for(int i = 5;i<gridSize-5;i++) {
         potential[gridSize/3][i] = 100;
         isConductor[gridSize/3][i] = true;
         potential[2*gridSize/3][i] = -100;
         isConductor[2*gridSize/3][i] = true;
      }
      frame.setAll(potential);
   }

   public void doStep() {
      double error = 0;
      for(int i = 1;i<gridSize-1;i++) {
         for(int j = 1;j<gridSize-1;j++) {
            // change the voltage for nonconductors
            if(!isConductor[i][j]) {
               double v = (potential[i-1][j]+potential[i+1][j]
                           +potential[i][j-1]+potential[i][j+1])/4;
               double dv = potential[i][j]-v;
               error = Math.max(error, Math.abs(dv));
               potential[i][j] = v;
            }
         }
      }
      frame.setAll(potential);
      if(error<maximumError) {
         // stop the simulation thread
         animationThread = null;
         control.calculationDone("Computation done.");
      }
   }

   public void reset() {
      control.setValue("maximum error", 0.1);
      control.setValue("size", 31);
      initialize();
   }
```