# Chapter 10

# Electrodynamics

We compute the electric fields due to static and moving charges, describe methods for computing the electric potential in boundary value problems, and solve Maxwell's equations numerically.

## 10.1 Static Charges

Suppose we want to know the electric field $\mathbf{E}(\mathbf{r})$ at the point $\mathbf{r}$ due to $N$ point charges $q_1, q_2, \ldots, q_N$ at fixed positions $\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N$. We know that $\mathbf{E}(\mathbf{r})$ satisfies a superposition principle and is given by

$$\mathbf{E}(\mathbf{r}) = K \sum_i^N \frac{q_i}{|\mathbf{r} - \mathbf{r}_i|^3} (\mathbf{r} - \mathbf{r}_i), \tag{10.1}$$

where $\mathbf{r}_i$ is the fixed location of the $i$th charge and $K$ is a constant that depends on our choice of units. One of the difficulties associated with electrodynamics is the competing systems of units. In the SI (or rationalized MKS) system of units, the charge is measured in coulombs (C) and the constant $K$ is given by

$$K = \frac{1}{4\pi\epsilon_0} \approx 9.0 \times 10^9\,\mathrm{N \cdot m^2/C^2}. \qquad \text{(SI units)} \tag{10.2}$$

The constant $\epsilon_0$ is the electrical permittivity of free space. This choice of units is not convenient for computer programs because $K \gg 1$. Another popular system of units is the Gaussian (cgs) system for which the constant $K$ is absorbed into the unit of charge so that $K = 1$. Charge is in electrostatic units or esu. One feature of Gaussian units is that the electric and magnetic fields have the same units. For example, the (Lorentz) force on a particle of charge $q$ and velocity $\mathbf{v}$ in an electric field $\mathbf{E}$ and a magnetic field $\mathbf{B}$ has the form

$$\mathbf{F} = q(\mathbf{E} + \frac{\mathbf{v}}{c} \times \mathbf{B}). \qquad \text{(Gaussian units)} \tag{10.3}$$

These virtues of the Gaussian system of units lead us to adopt this system for this chapter even though SI units are used in introductory texts.

## 10.2 Electric Fields

The electric field is an example of a *vector field* because it defines a vector quantity at every point in space. One way to visualize this field is to divide space into a discrete grid and to draw arrows in the direction of **E** at the vertices of this grid. The length of the arrow can be chosen to be proportional to the magnitude of the electric field. Another possibility is to use color or gray scale to represent the magnitude. Because 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 `Vector2DFrame` class in the Open Source Physics frames package uses the color representation (see Appendix 10A).

The `ElectricFieldApp` program computes the electric field due to an arbitrary number of point charges. A charge is created using the control's $x$, $y$, and $q$ parameters when the Calculate button is clicked. Each time the Calculate button is clicked, another charge is created. Whenever a charge is added or moved, the electric field is recomputed in the `calculateField` method. The Reset button removes all charges.

Because the number of charges can change, we need a way to obtain the position and charge for each point charge so that the electric field can be computed. The drawing panel for `frame` contains a list of all drawable objects, but we need a way to obtain only those objects that are of type `Charge`. The following statements show one way of doing this.

```
List chargeList = frame.getDrawables(Charge.class);
Iterator it = chargeList.iterator();
```

The argument `Charge.class` tells the `frame.getDrawables` method to return only a list of objects that can be cast to the `Charge` class.[1] These objects are then placed in an object of type `ArrayList` which implements the Java interface `List`. The `iterator` method returns an object called `it` which implements the `Iterator` interface. We then use the object `it` to loop through all the charges using the `next` and `hasNext` methods of the interface `Iterator`. As the name implies, an iterator is a convenient way to access a list without explicitly counting its elements. You will modify `ElectricFieldApp` to include a moving test charge in Problem 10.1.

Listing 10.1: `ElectricFieldApp` computes and displays the electric field from a list of point charges.

```
package org.opensourcephysics.sip.ch10;
import java.awt.event.MouseEvent;
import java.util.Iterator;
import java.util.List;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.display.*;
import org.opensourcephysics.frames.Vector2DFrame;
```

[1]The syntax `Charge.class` uses a small portion of the Java reflection API to determine the type of object that is being requested. The reflection API is an advanced feature of Java.

```
public class ElectricFieldApp extends AbstractCalculation implements InteractiveMouseHandler
   int n = 20;                                   // grid points on a side
   double a = 10;                                // viewing side length
   double[][][] eField = new double[2][n][n]; // stores electric field
   Vector2DFrame frame = new Vector2DFrame("x", "y", "Electric field");

   public ElectricFieldApp() {
      frame.setPreferredMinMax(-a/2, a/2, -a/2, a/2);
      frame.setZRange(false, 0, 2);
      frame.setAll(eField); // sets the vector field
      frame.setInteractiveMouseHandler(this);
   }

   public void calculate() {
      double x = control.getDouble("x");
      double y = control.getDouble("y");
      double q = control.getDouble("q");
      Charge charge = new Charge(x, y, q);
      frame.addDrawable(charge);
      calculateField();
   }

   public void reset() {
      control.println("Calculate creates a new charge and updates the field.");
      control.println("You can drag charges.");
      frame.clearDrawables(); // removes all charges
      control.setValue("x", 0);
      control.setValue("y", 0);
      control.setValue("q", 1);
      calculateField();
   }

   void calculateField() {
      for(int ix = 0;ix<n;ix++) {
         for(int iy = 0;iy<n;iy++) {
            eField[0][ix][iy] = eField[1][ix][iy] = 0; // zeros field
         }
      }
      // the charges in the frame
      List chargeList = frame.getDrawables(Charge.class);
      Iterator it = chargeList.iterator();
      while(it.hasNext()) {
         Charge charge = (Charge) it.next();
         double
            xs = charge.getX(), ys = charge.getY();
         for(int ix = 0;ix<n;ix++) {
            double x = frame.indexToX(ix);
            double dx = (x-xs);               // distance of charge to gridpoint
            for(int iy = 0;iy<n;iy++) {
               double y = frame.indexToY(iy);
               double dy = (y-ys);           // charge to gridpoint
               double r2 = dx*dx+dy*dy;      // distance squared
               double r3 = Math.sqrt(r2)*r2; // distance cubed
               if(r3>0) {
                  eField[0][ix][iy] += charge.q*dx/r3;
                  eField[1][ix][iy] += charge.q*dy/r3;
               }
            }
         }
      }
      frame.setAll(eField);
   }

   public void handleMouseAction(InteractivePanel panel, MouseEvent evt) {
      panel.handleMouseAction(panel, evt); // panel moves the charge
      if(panel.getMouseAction()==InteractivePanel.MOUSE_DRAGGED) {
         calculateField(); // remove this line if user interface is slugish
         panel.repaint();
      }
   }

   public static void main(String[] args) {
      CalculationControl.createApp(new ElectricFieldApp());
   }
}
```

To make the program interactive, the `ElectricFieldApp` class implements the `InteractiveMouseHandler` to process mouse events when a charge is dragged. (See Section 5.7 for a discussion of interactive panels and interactive mouse handlers.) The class registers its interest in handling these events using the `setInteractiveMouseHandler` method. The handler passes the event to the panel to move the charge and then recalculates the field. Note that the `Charge` class in Listing 10.2 inherits from the `InteractiveCircle` class.[2]

Listing 10.2: The `Charge` class extends the `InteracticeCircle` class and adds the charge property.

```
package org.opensourcephysics.sip.ch10;
import java.awt.Color;
import org.opensourcephysics.display.InteractiveCircle;

public class Charge extends InteractiveCircle {
   double q = 0;

   public double getQ() {
      return q;
   }

   public Charge(double x, double y, double q) { //
      super(x, y);
```

[2]Dragging may become sluggish if too many computations are performed within the mouse action method.

```
      this.q = q;
      if(q>0) {
         color = Color.red;
      } else {
         color = Color.blue;
      }
   }
}
```

**Problem 10.1.** Motion of a charged particle in an electric field

a. Test `ElectricFieldApp` by adding one charge at a time at various locations. Do the electric field patterns look reasonable? For example, does the electric field point away from positive charges and toward negative charges? How well is the magnitude of the electric field represented?

b. Modify `ElectricFieldApp` so that it uses an `AbstractAnimation` to compute the motion of a test particle of mass $m$ and charge $q$ in the presence of the electric field created by a fixed distribution of point charges. That is, create a drawable test charge that implements the `ODE` interface and add it to the vector field frame. Use the same approach that was used for the trajectory problems in Chapter 5. The acceleration of the charge is given by $q\mathbf{E}/m$, where $\mathbf{E}$ is the electric field due to the fixed point charges. Use a higher-order algorithm to advance the position and velocity of the particle. (Ignore the effects of radiation due to accelerating charges.)

c. Assume that $\mathbf{E}$ is due to a charge `q(1)` $= 1.5$ fixed at the origin. Simulate the motion of a charged particle of mass $m = 0.1$ and charge $q = 1$ initially at $x = 1, y = 0$. Consider the following initial conditions for its velocity: $v_x = 0, v_y = 0$; $v_x = 1, v_y = 0$; $v_x = 0, v_y = 1$; and $v_x = -1$, $v_y = 0$. Is the trajectory of the particle tangent to the field vectors? Explain.

d. Assume that the electric field is due to two fixed point charges: `q(1)` $= 1$ at `x(1)` $= 2,$ `y(1)` $= 0$ and `q(2)` $= -1$ at `x(2)` $= -2,$ `y(2)` $= 0$. Place a charged particle of unit mass and unit positive charge at $x = 0.05, y = 0$. What do you expect the motion of this charge to be? Do the simulation and determine the qualitative nature of the motion.

e.* Consider the motion of a charged particle in the vicinity of the electric dipole defined in part (d). Choose the initial position to be five times the separation of the charges in the dipole. Do you find any bound orbits? Do you find any closed orbits or do all orbits show some precession?

## 10.3 Electric Field Lines

Another way of visualizing the electric field is to draw *electric field lines.* The properties of these lines are as follows:

1. An electric field line is a directed line whose tangent at every position is parallel to the electric field at that position.

2. The lines are smooth and continuous except at singularities such as point charges. (It makes no sense to talk about the electric field *at* a point charge.)
3. The density of lines at any point in space is proportional to the magnitude of the field at that point. This property implies that the total number of electric field lines from a point charge is proportional to the magnitude of that charge. The value of the proportionality constant is chosen to provide the clearest pictorial representation of the field. The drawing of field lines is art plus science.

The `FieldLineApp` program draws electric field lines in two dimensions. The program makes extensive use of the `FieldLine` class which implements the following algorithm:

1. Begin at a point $(x, y)$ and compute the components $E_x$ and $E_y$ of the electric field vector $\mathbf{E}$ using (10.1).
2. Draw a small line segment of size $\Delta s = |\Delta \mathbf{s}|$ tangent to $\mathbf{E}$ at that point. The components of the line segment are given by
$$\Delta x = \Delta s \frac{E_x}{|\mathbf{E}|} \text{ and } \Delta y = \Delta s \frac{E_y}{|\mathbf{E}|}. \tag{10.4}$$
3. Iterate the process beginning at the new point $(x + \Delta x, y + \Delta y)$. Continue until the field line approaches a point charge singularity or escapes toward infinity.

This field line algorithm is equivalent to solving the following differential equations:

$$\frac{dx}{ds} = \frac{E_x}{|\mathbf{E}|} \tag{10.5a}$$
$$\frac{dy}{ds} = \frac{E_y}{|\mathbf{E}|}. \tag{10.5b}$$

Because a field line extends in both directions from the algorithm's starting point, the computation must be repeated in the $(-E_x/|\mathbf{E}|, -E_y/|\mathbf{E}|)$ direction to obtain a complete visualization of the field line. Note that this algorithm draws a correct field line, but does not draw a collection of field lines with a density proportional to the field intensity.

To draw the field lines we start a computation when a user double clicks in the panel and end the computation when the field line approaches a point charge or when the magnitude of the field becomes too small. Although we can easily describe these stopping conditions, we do not know how long the computation will take, and we might want to compute multiple field lines simultaneously. An elegant way to do this computation is to use threads.

As we discussed in Section 2.6, Java programs can have multiple threads to separate and organize related tasks. A thread is an *independent* task within a single program that shares the program's data with other threads.[3] In the following example, we create a thread to compute the solution of the differential equation for an electric field line. It is natural to use threads in this context because the drawing of a field line involves starting the field line, drawing each piece

[3]The *Open Source Physics User's Guide* describes simulation threads in more detail.

of the field line, and then stopping the calculation when some stopping condition is met. The computation begins when the `FieldLine` object is created and ends when the stopping condition is satisfied.

A thread executes statements within an object, such as `FieldLine`, that implements the `Runnable` interface. This interface consists of a single method, the `run` method, and the thread executes the code within this method. The run method is not invoked directly, but is invoked automatically by the thread after the thread is started. When the run method exits, the thread that invoked the run method stops executing and is said to die. After a thread dies, it cannot be restarted. Another thread must be created if we wish to invoke the run method a second time.

We build a `FieldLine` class by subclassing `Thread` and adding the necessary drawing and differential equation capabilities using the `Drawable` and `ODE` interfaces, respectively. This class is shown in Listing 10.3.

Listing 10.3: The `FieldLine` class computes an electric field line using a `Thread`.

```java
package org.opensourcephysics.sip.ch10;
import java.awt.Graphics;
import java.util.*;
import org.opensourcephysics.display.*;
import org.opensourcephysics.numerics.*;

public class FieldLine implements Drawable, ODE, Runnable {
   DrawingFrame frame;
   double[] state = new double[2]; // Ex and Ey for ODE
   ODESolver odeSolver = new RK45MultiStep(this);
   ArrayList chargeList;            // list of charged particles
   Trail trail;
   double stepSize;
   volatile boolean done = false;

   public FieldLine(DrawingFrame frame, double x0, double y0, double stepSize) {
      this.stepSize = stepSize;
      this.frame = frame;
      odeSolver.setStepSize(stepSize);
      state[0] = x0;
      state[1] = y0;
      chargeList = frame.getDrawables(Charge.class);
      trail = new Trail();
      trail.addPoint(x0, y0);
      Thread thread = new Thread(this);
      thread.start();
   }

   public double[] getState() {
      return state;
   }

   public void getRate(double[] state, double[] rate) {
      double ex = 0;
      double ey = 0;
      for(Iterator it = chargeList.iterator();it.hasNext();) {
         Charge charge = (Charge) it.next();
         double dx = (charge.getX()-state[0]);
         double dy = (charge.getY()-state[1]);
         double r2 = dx*dx+dy*dy;
         double r = Math.sqrt(r2);
         if((r<2*stepSize)||(r>100)) { // done if too close or too far
            done = true;
         }
         ex += (r==0) ? 0 : charge.q*dx/r2/r;
         ey += (r==0) ? 0 : charge.q*dy/r2/r;
      }
      double mag = Math.sqrt(ex*ex+ey*ey);
      rate[0] = (mag==0) ? 0 : ex/mag;
      rate[1] = (mag==0) ? 0 : ey/mag;
   }

   public void run() {
      int counter = 0;
      while(((counter<1000)&&!done)) {
         odeSolver.step();
         trail.addPoint(state[0], state[1]);
         if(counter%50==0) {      // repaint every 50th step
            frame.repaint();
            try {
               Thread.sleep(20); // give the event queue a chance
            } catch(InterruptedException ex) {}
         }
         counter++;
         Thread.yield();
      }
      frame.repaint();
   }

   public void draw(DrawingPanel panel, Graphics g) {
      trail.draw(panel, g);
   }
}
```

The **`FieldLine`** constructor saves a reference to the list of charges to calculate the electric field using (10.1). The loop in the `run` method solves the differential equation and stores the solution in a drawable trail. The loop is exited when the field line is close to a charge or when the magnitude of the field becomes too small. Because there are situations where the field line will never stop, this loop is executed no more than 1000 times.

The `FieldLineApp` program instantiates a field line when the user double clicks within the panel. Adding a charge or moving a charge removes all field lines from the panel. Study how the `handleMouseAction` allows the user to drag charges and to initiate the drawing of field lines. You are asked to modify this program in Problem 10.2.

Listing 10.4: The `FieldLineApp` program computes an electric field line when the user clicks within the panel.

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

public class FieldLineApp extends AbstractCalculation implements InteractiveMouseHandler {
   DisplayFrame frame = new DisplayFrame("x", "y", "Field lines");

   public FieldLineApp() {
      frame.setInteractiveMouseHandler(this);
      frame.setPreferredMinMax(-10, 10, -10, 10);
   }

   public void calculate() {
      frame.removeObjectsOfClass(FieldLine.class); // remove old field lines
      double x = control.getDouble("x");
      double y = control.getDouble("y");
      double q = control.getDouble("q");
      Charge charge = new Charge(x, y, q);
      frame.addDrawable(charge);
   }

   public void reset() {
      control.println("Calculate creates a new charge and clears the field lines.");
      control.println("You can drag charges.");
      control.println("Double click in display to compute a field line.");
      frame.clearDrawables(); // remove charges and field lines
      control.setValue("x", 0);
      control.setValue("y", 0);
      control.setValue("q", 1);
   }

   public void handleMouseAction(InteractivePanel panel, MouseEvent evt) {
      panel.handleMouseAction(panel, evt); // panel handles dragging
      switch(panel.getMouseAction()) {
      case InteractivePanel.MOUSE_DRAGGED :
         if(panel.getInteractive()==null) {
            return;
         }
         frame.removeObjectsOfClass(FieldLine.class); // field is invalid
         frame.repaint();                             // repaint to keep the screen up to da
         break;
      case InteractivePanel.MOUSE_CLICKED :
         if(evt.getClickCount()>1) {                  // check for double click
            double
               x = panel.getMouseX(), y = panel.getMouseY();
            FieldLine fieldLine = new FieldLine(frame, x, y, +0.1);
            panel.addDrawable(fieldLine);
            fieldLine = new FieldLine(frame, x, y, -0.1);
            panel.addDrawable(fieldLine);
         }
         break;
      }
   }

   public static void main(String[] args) {
      CalculationControl.createApp(new FieldLineApp());
   }
}
```

**Problem 10.2.** Verification of field line program

a. Draw field lines for a few simple sets of one, two, and three charges. Choose sets of charges for which all have the same sign, and sets for which they are different. Verify that the field lines never connect charges of the same sign. Why do field lines never cross? Are the units of charge and distance relevant?

b. Compare `FieldLineApp` and `ElectricFieldApp`. Which representation conveys more information? Consider how each program provides (or does not provide) information about the electric field magnitude and direction. Discuss some of the difficulties with making an accurate field line diagram.

c. `FieldLine` uses a constant value for $\Delta s$. Modify the algorithm so that the calculation continues when a field line moves off the screen, but speed up the algorithm by increasing the value of $\Delta s$.

d. Removing a field line from the drawing panel in the `reset` method does not stop the thread. Improve the performance of the program by modifying `ElectricFieldApp` so that a field line's `done` variable is set to false when it is removed from the drawing panel.

**Problem 10.3.** Electric field lines from point charges

a. Modify `FieldLineApp` so that a charge starts ten field lines per unit of charge whenever a new charge is added to the panel or when a charge is moved. Start these field lines close to each charge in such a way that they propagate away from the charge. Should you start these field lines on both positive and negative charges? Explain your answer.

b. Draw the field lines for an electric dipole.

c. Draw the field lines for the electric quadrupole with `q(1)` $= 1$, `x(1)` $= 1$, `y(1)` $= 1$, `q(2)` $= -1$, `x(2)` $= -1$, `y(2)` $= 1$, `q(3)` $= 1$, `x(3)` $= -1$, `y(3)` $= -1$, and `q(4)` $= -1$, `x(4)` $= 1$, and `y(4)` $= -1$.

d. A continuous charge distribution can be approximated by a large number of closely spaced point charges. Draw the electric field lines due to a row of ten equally spaced unit charges located between $-2.5$ and $+2.5$ on the $x$ axis. How does the electric field distribution compare to the distribution due to a single point charge?

e. Repeat part (c) with two rows of equally spaced positive charges on the lines $y = 0$ and $y = 1$, respectively. Then consider one row of positive charges and one row of negative charges.

**Problem 10.4.** Field lines due to infinite line of charge

a. The `FieldLineApp` program plots field lines in two dimensions. Sometimes this restriction can lead to spurious results (see Freeman). Consider four identical charges placed at the corners of a square. Use the program to plot the field lines. What, if anything, is wrong with the results? What should happen to the field lines near the center of the square?

b. The two-dimensional analog of a point charge is an infinite line (thin cylinder) of charge perpendicular to the plane. The electric field due to an infinite line of charge is proportional to the linear charge density and inversely proportional to the distance (instead of the distance squared) from the line of charge to a point in the plane. Modify the `FieldLine` class to compute the field lines from line charges with $E(r) = 1/r$. Use your modified class to draw the field lines due to four identical line charges located at the corners of a square, and compare the field lines with your results in part (a).

c. Use your modified program from part (b) to draw the field lines for the two-dimensional analogs of the distributions considered in Problem 10.3. Compare the results for two and three dimensions, and discuss any qualitative differences.

d. Can your program be used to demonstrate Gauss's law using point charges? What about line charges?

## 10.4 Electric Potential

It often is easier to analyze the behavior of a system using energy rather than force concepts. We define the electric potential $V(\mathbf{r})$ by the relation

$$V(\mathbf{r}_2) - V(\mathbf{r}_1) = -\int_{\mathbf{r}_1}^{\mathbf{r}_2} \mathbf{E} \cdot d\mathbf{r}, \tag{10.6}$$

or

$$\mathbf{E}(\mathbf{r}) = -\nabla V(\mathbf{r}). \tag{10.7}$$

Only differences in the potential between two points have physical significance. The gradient operator $\nabla$ is given in Cartesian coordinates by

$$\nabla = \frac{\partial}{\partial x}\hat{\mathbf{x}} + \frac{\partial}{\partial y}\hat{\mathbf{y}} + \frac{\partial}{\partial z}\hat{\mathbf{z}}, \tag{10.8}$$

where the vectors $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{z}}$ are unit vectors along the $x$, $y$, and $z$ axes respectively. If $V$ depends only on the magnitude of $\mathbf{r}$, then (10.7) becomes $E(r) = -dV(r)/dr$. Recall that $V(r)$ for a point charge $q$ relative to a zero potential at infinity is given by

$$V(r) = \frac{q}{r}. \qquad \text{(Gaussian units)} \tag{10.9}$$

The surface on which the electric potential has an equal value everywhere is called an *equipotential surface* (a curve in two dimensions). Because **E** is in the direction in which the electric potential decreases most rapidly, the electric field lines are orthogonal to the equipotential surfaces at any point.

The Open Source Physics frames package contains the `Scalar2DFrame` class to provide graphical representations of scalar fields (see Appendix 9B). Problem 10.5 uses a scalar field plot to show the electric potential. The following code fragment shows how to calculate the electric potential at a grid point.

```java
List chargeList = frame.getDrawables(Charge.class);
Iterator it = chargeList.iterator();
while (it.hasNext()) {
   Charge charge = (Charge) it.next();
   double xs = charge.getX(), ys = charge.getY();
   for (int ix = 0; ix < n; ix++) {
      double x= frame.indexToX(ix);
      double dx = (xs - x); // charge gridpoint separation
      for (int iy = 0; iy < n; iy++) {
         double y= frame.indexToY(iy);
         double dy = (ys -y); //charge gridpoint separation
         double r2 = dx * dx + dy * dy;
         double r = Math.sqrt(r2);
         if (r > 0) {
            eField[ix][iy] += charge.q/r;
         }
      }
   }
}
frame.setAll(eField);
```

**Problem 10.5.** Equipotential contours

a. Write a program based on `ElectricFieldApp` that draws equipotential lines using the charge distributions considered in Problem 10.3.

b. Explain why equipotential surfaces (lines in two dimensions) never cross.

We can use the orthogonality between the electric field lines and the equipotential lines to modify `FieldLineApp` so that it draws the latter. Because the components of the line segment $\Delta\mathbf{s}$ parallel to the electric field line are given by $\Delta x = \Delta s(E_x/E)$ and $\Delta y = \Delta s(E_y/E)$, the components of the line segment perpendicular to **E**, and hence parallel to the equipotential line, are given by $\Delta x = -\Delta s(E_y/E)$ and $\Delta y = \Delta s(E_x/E)$. It is unimportant whether the minus sign is assigned to the $x$ or $y$ component, because the only difference would be the direction that the equipotential lines are drawn.

**Problem 10.6.** Equipotential lines

a. Write a program that is based on `FieldLineApp` and `FieldLine` to draw some of the equipotential lines for the charge distributions considered in Problem 10.3. Use a mouse click to determine the initial position of an equipotential line. The equipotential calculation should stop when the line returns close to the starting point or after an unreasonable number of calculations. You should also kill the thread when the user moves a charge, hits the Reset button, or when the application terminates.

b. What would a higher density of equipotential lines mean if we drew lines such that each adjacent line differed from a neighboring one by a fixed potential difference?

c. Explain why equipotential surfaces never cross.

**Problem 10.7.** The electric potential due to a finite sheet of charge

Consider a uniformly charged nonconducting plate of total charge $Q$ and linear dimension $L$ centered at $(0,0,0)$ in the $x$-$y$ plane. In the limit $L \to \infty$ with the charge density $\sigma = Q/L^2$ a constant, we know that the electric field is normal to the sheet and its magnitude is given by $2\pi\sigma$ (Gaussian units). What is the electric field due to a finite sheet of charge? A simple method is to divide the plate into a grid of $p$ square regions on a side such that each region is sufficiently small to be approximated by a point charge of magnitude $q = Q/p^2$. Because the potential is a scalar, it is easier to compute the total potential rather than the total electric field from the $N = p^2$ point charges. Use the relation (10.9) for the potential from a point charge and write a program to compute $V(z)$ and hence $E_z = -\partial V(z)/\partial z$ for points along the $z$-axis and perpendicular to the sheet. Take $L = 1$, $Q = 1$, and $p = 10$ for your initial calculations. Increase $p$ until your results for $V(z)$ do not change significantly. Plot $V(z)$ and $E_z$ as a function of $z$ and compare their $z$-dependence to their infinite sheet counterparts.

***Problem 10.8.** Electrostatic shielding

We know that the (static) electric field is zero inside a conductor, all excess charges reside on the surface of the conductor, and the surface charge density is greatest at the points of greatest curvature. Although these properties are plausible, it is instructive to do a simulation to see how these properties follow from Coulomb's law. For simplicity, consider the conductor to be two-dimensional so that the potential energy is proportional to $\ln r$ rather than $1/r$ (see Problem 10.4). It also is convenient to choose the surface of the conductor to be an ellipse.

a. If we are interested only in the final distribution of the charges and not in the dynamics of the system, we can use a Monte Carlo method. Our goal is to find the minimum energy configuration beginning with the $N$ charges randomly placed within a conducting ellipse. One method is to choose a charge $i$ at random, and make a trial change in the position of the charge. The trial position should be no more than $\delta$ from the old position and still be within the ellipse. Choose $\delta \approx b/10$, where $b$ is the semiminor axis of the ellipse. Compute the change in the total potential energy given by (in arbitrary units)

$$\Delta U = -\sum_j [\ln r_{ij}^{(\text{new})} - \ln r_{ij}^{(\text{old})}]. \tag{10.10}$$

The sum is over all charges in the system not including $i$. If $\Delta U > 0$, then reject the trial move, otherwise accept it. Repeat this procedure many times until very few trial moves are accepted. Write a program to implement this Monte Carlo algorithm. Run the simulation for $N \geq 20$ charges inside a circle and then repeat the simulation for an ellipse. How are the charges distributed in the (approximately) minimum energy distribution? Which parts of the ellipse have a higher charge density?

b. Repeat part (a) for a two-dimensional conductor, but assume that the potential energy $U \sim 1/r$. Do the charges move to the surface?

c. Is it sufficient that the interaction be repulsive for the results of parts (a) and (b) to hold?

d. Repeat part (a) with the added condition that there is a fixed positive charge of magnitude $N/2$ located outside the ellipse. How does this fixed charge effect the charge distribution? Are the excess free charges still at the surface? Try different positions for the fixed charge.

e. Repeat parts (a) and (b) for $N = 50$ charges located within an ellipsoid in three dimensions.

## 10.5 Numerical Solutions of Boundary Value Problems

In Section 10.1 we found the electric fields and potentials due to a fixed distribution of charges. Suppose that we do not know the positions of the charges and instead know only the potential on a set of boundaries surrounding a charge-free region. This information is sufficient to determine the potential $V(\mathbf{r})$ at any point within the charge-free region.

The direct method of solving for $V(x,y,z)$ is based on Laplace's equation which can be expressed in Cartesian coordinates as

$$\nabla^2 V(x,y,z) \equiv \frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} + \frac{\partial^2 V}{\partial z^2} = 0. \tag{10.11}$$

The problem is to find the function $V(x,y,z)$ that satisfies (10.11) and the specified boundary conditions. This type of problem is an example of a *boundary value* problem. Because analytical methods for regions of arbitrary shape do not exist, the only general approach is to use numerical methods. Laplace's equation is not a new law of physics, but can be derived directly from (10.7) and the relation $\nabla \cdot \mathbf{E} = 0$ or indirectly from Coulomb's law in regions of space where there is no charge.

For simplicity, we consider only two-dimensional boundary value problems for $V(x,y)$. We use a finite difference method and divide space into a discrete grid of sites located at the coordinates $(x,y)$. In Problem 10.9b, we show that in the absence of a charge at $(x,y)$, the discrete form of Laplace's equation satisfies the relation

$$\begin{aligned} V(x,y) \approx \frac{1}{4}[&V(x+\Delta x,y) + V(x-\Delta x,y) \\ &+ V(x,y+\Delta y) + V(x,y-\Delta y)], \qquad \text{(two dimensions)} \end{aligned} \tag{10.12}$$

where $V(x,y)$ is the value of the potential at the site $(x,y)$. Equation (10.12) says that $V(x,y)$ is the average of the potential of its four nearest neighbor sites. This remarkable property of $V(x,y)$ can be derived by approximating the partial derivatives in (10.11) by finite differences (see Problem 10.9b).

In Problem 10.9a we verify (10.12) by calculating the potential due to a point charge at a point in space selected by the user and at the four nearest neighbors. As the form of (10.12) implies, the average of the potential at the four neighboring sites should equal the potential at the center site. We assume the form (10.9) for the potential $V(r)$ due to a point charge, a form that satisfies Laplace's equation for $r \neq 0$.

**Problem 10.9.** Verification of the difference equation for the potential

a. Modify `PotentialFieldApp` to compare the computed potential at a point to the average of the potential at its four nearest neighbor sites. Choose reasonable values for the spacings $\Delta x$ and $\Delta y$ and consider a point that is not too close to the source charge. Do similar measurements for other points. Does the relative agreement with (10.12) depend on the distance of the point to the source charge? Choose smaller values of $\Delta x$ and $\Delta y$ and determine if your results are in better agreement with (10.12). Does it matter whether $\Delta x$ and $\Delta y$ have the same value?

b. Derive the finite difference equation (10.12) for $V(x, y)$ using the second-order Taylor expansion:

$$V(x+\Delta x, y) = V(x, y) + \Delta x \frac{\partial V(x, y)}{\partial x} + \frac{1}{2}(\Delta x)^2 \frac{\partial^2 V(x, y)}{\partial x^2} + \ldots \tag{10.13}$$

$$V(x, y+\Delta y) = V(x, y) + \Delta y \frac{\partial V(x, y)}{\partial y} + \frac{1}{2}(\Delta y)^2 \frac{\partial^2 V(x, y)}{\partial y^2} + \ldots \tag{10.14}$$

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.

```java
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
      }
      for(int i = 5;i<gridSize-5;i++) { // set potential on inner conductor
         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++) {
            if(!isConductor[i][j]) { // change the voltage for non-conductors
               double v = (potential[i-1][j]+potential[i+1][j]+potential[i][j-1]+potential[i
                           /4;
               double dv = potential[i][j]-v;
               error = Math.max(error, Math.abs(dv));
               potential[i][j] = v;
            }
         }
      }
      frame.setAll(potential);
      if(error<maximumError) {
         animationThread = null; // this will stop the simulation thread
         control.calculationDone("Computation done.");
      }
   }

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

   public void handleMouseAction(InteractivePanel panel, MouseEvent evt) {
      switch(panel.getMouseAction()) {
      case InteractivePanel.MOUSE_DRAGGED :
      case InteractivePanel.MOUSE_PRESSED :
         double x = panel.getMouseX(); // mouse x in world units
         double y = panel.getMouseY();
         int i = frame.xToIndex(x);     // closest array index
         int j = frame.yToIndex(y);
         frame.setMessage("V="+decimalFormat.format(potential[i][j]));
         break;
      case InteractivePanel.MOUSE_RELEASED :
         panel.setMessage(null);
         break;
      }
   }

   public static void main(String[] args) {
      SimulationControl.createApp(new LaplaceApp());
   }
}
```

As the algorithm loops through the grid sites, it first checks if each grid site is a conductor. If it is, the site is skipped. If not, a new potential is calculated and assigned to the proper element in the `potential` array. A local variable named `maximumError` keeps track of the maximum difference between the potential at a site and the average potential of the four neighbors. This variable is used to determine the end of the simulation.

In Problems 10.10–10.12 you are asked to modify `LaplaceApp` to compute the potential for various geometries.

**Problem 10.10.** Numerical solution of the potential within a rectangular region

a. Modify `LaplaceApp` to determine the potential $V(x, y)$ in a square region with linear dimension $L = 10$. The boundary of the square is at a potential $V = 10$. Choose the grid size $\Delta x = \Delta y = 1$. Before you run the program, guess the exact form of $V(x, y)$ and set the initial values of the interior potential 10% lower than the exact answer. How many iterations are necessary to achieve 1% accuracy? Decrease the grid size by a factor of two, and determine the number of iterations that are now necessary to achieve 1% accuracy.

b. Consider the same geometry as in part (a), but set the initial potential at the interior sites equal to zero except for the center site whose potential is set equal to four. Does the potential distribution evolve to the same values as in part (a)? What is the effect of a poor initial guess? Are the final results independent of your initial guess?

c. Modify `LaplaceApp` so that the value of the potential at the four sides is 5, 10, 5, and 10, respectively (see Figure 10.1). Sketch the equipotential surfaces. What happens if the potential is 10 on three sides and 0 on the fourth? Start with a reasonable guess for the initial values of the potential at the interior sites and iterate until 1% accuracy is obtained.

d.* Consider the same initial choice of the potential as in part (b) and focus your attention on the potential at the sites near the center of the square. If the central site has an initial potential of four, what is the potential at the nearest neighbor sites after the first iteration? Follow the distribution of the potential as a function of the number of iterations and verify that the nature of the relaxation of the potential to its correct distribution is closely related to *diffusion* (see Chapter 7). It may be helpful to increase the number of sites in the grid and the initial value of the potential at the central site to see the nature of the relaxation more clearly.

In Problem 10.10, we implemented a simple version of the relaxation method known as the Jacobi method. In particular, the new potential at each site is based on the values of the potentials at the neighboring sites at the previous iteration. After the entire lattice is visited, the potential at each site is updated *simultaneously*. The difficulty with this relaxation method is that it converges very slowly. The use of more general relaxation methods is discussed in many texts (cf. Sadiku or Press et al.). In Problem 10.11 we consider a method known as *Gauss-Seidel* relaxation.

**Problem 10.11.** Gauss-Seidel relaxation

a. Modify the program that you used in Problem 10.10 so that the potential at each site is updated sequentially. That is, after the average potential of the nearest neighbor sites of site $i$ is computed, update the potential at $i$ immediately. In this way the new potential of the next site is computed using the most recently computed values of its nearest neighbor potentials. Are your results better, worse, or about the same as for the simple relaxation method?

b. Imagine coloring the alternate sites of a grid red and black, so that the grid resembles a checkerboard. Modify the program so that all the red sites are updated first, and then all the black sites are updated. This ordering is repeated for each iteration. Do your results converge any more quickly than in part (a)?

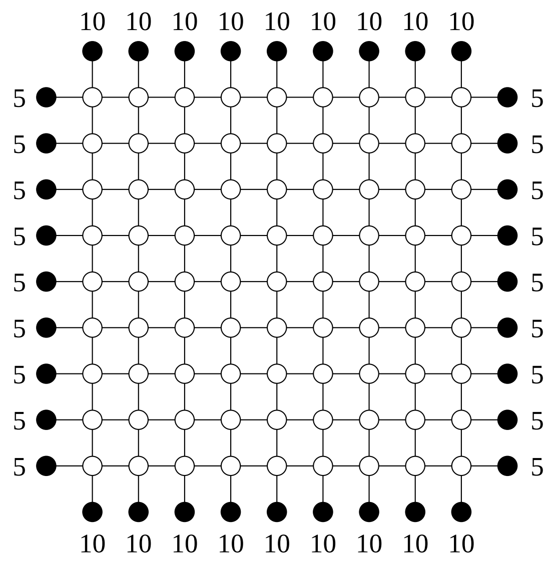


Figure 10.1: Potential distribution considered in Problem 10.10c. The number of interior sites in each direction is nine.

c.* The slow convergence of the relaxation methods we have explored is due to the fact that it takes a long time for a change in the potential at one site to effect changes further away. We can improve the Gauss-Seidel method by using an *overrelaxation* method that updates the new potential as follows:

$$V_{\text{new}}(x, y) = wV_{\text{ave}}(x, y) + (1 - w)V(x, y), \tag{10.15}$$

where $V_{\text{ave}}(x, y)$ is the average of the potential of the four neighbors of $(x, y)$. The overrelaxation parameter $w$ is in the range $1 < w < 2$. The effect of $w$ is to cause the potential to change by a greater amount than in the simple relaxation procedure. Explore the dependence of the rate of convergence on $w$. A relaxation method that increases the rate of convergence is explored in Project 10.26.

**Problem 10.12.** The capacitance of concentric squares

a. Use a relaxation method to compute the potential distribution between the two concentric square cylinders shown in Figure 10.2. The potential of the outer square conductor is $V_{\text{out}} = 10$ and the potential of the inner square conductor is $V_{\text{in}} = 5$. The linear dimensions of the exterior and interior squares are $L_{\text{out}} = 25$ and $L_{\text{in}} = 5$, respectively. Modify your program so that the potential of the interior square is fixed. Sketch the equipotential surfaces.

b. A system of two conductors with charge $Q$ and $-Q$ respectively has a capacitance $C$ that is defined as the ratio of $Q$ to the potential difference $\Delta V$ between the two conductors. Determine the capacitance per unit length of the concentric cylinders considered in part (a). In this case $\Delta V = 5$. The charge $Q$ can be determined from the fact that near a conducting surface, the surface charge density $\sigma$ is given by $\sigma = E_n/4\pi$, where $E_n$ is the magnitude of the electric

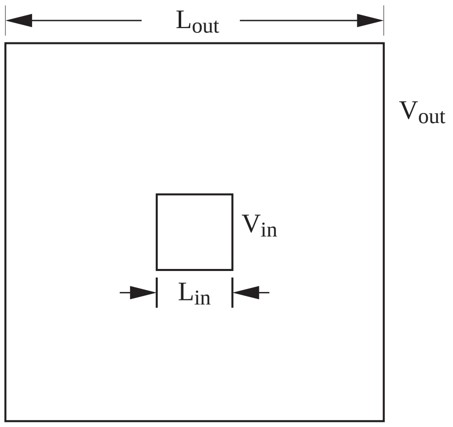


Figure 10.2: The geometry of the two concentric squares considered in Problem 10.12.

field normal to the surface. $E_n$ can be approximated by the relation $-\delta V/\delta r$, where $\delta V$ is the potential difference between a boundary site and an adjacent interior site a distance $\delta r$ away. Use the result of part (a) to compute $\delta V$ for each site adjacent to the two square surfaces. Use this information to determine $E_n$ for the two surfaces and the charge per unit length on each conductor. Are the charges equal and opposite in sign? Compare your numerical result to the capacitance per unit length, $1/2 \ln r_{\text{out}}/r_{\text{in}}$, of a system of two concentric circular cylinders of radii $r_{\text{out}}$ and $r_{\text{in}}$. Assume that the circumference of each cylinder equals the perimeter of the corresponding square, that is, $2\pi r_{\text{out}} = 4L_{\text{out}}$ and $2\pi r_{\text{in}} = 4L_{\text{in}}$.

c. Move the inner square 1 cm off center and repeat the calculations of parts (a) and (b). How do the potential surfaces change? Is there any qualitative difference if we set the inner conductor potential equal to $-5$?

Laplace's equation holds only in charge-free regions. If there is a charge density $\rho(x, y, z)$ in the region, we need to use *Poisson's* equation which can be written as

$$\nabla^2 V(\mathbf{r}) = \frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} + \frac{\partial^2 V}{\partial z^2} = -4\pi\rho(\mathbf{r}), \tag{10.16}$$

where $\rho(\mathbf{r})$ is the charge density. The difference form of Poisson's equation is given in two dimensions by

$$\begin{aligned} V(x,y) \approx \frac{1}{4}\big[&V(x+\Delta x, y) + V(x-\Delta x, y) + V(x, y+\Delta y) + V(x, y-\Delta y)\big] \\ &+ \frac{1}{4}\Delta x \Delta y\, 4\pi\rho(x,y). \end{aligned} \tag{10.17}$$

Note that the product $\rho(x, y)\Delta x\Delta y$ is the total charge in a $\Delta x \times \Delta y$ region centered at $(x, y)$.

**Problem 10.13.** Surface charge

a. Poisson's equation can be used to find the surface charge on a conductor after Laplace's equation has been solved. The potential is fixed at the boundary sites. If we assume the boundary is a conductor with some thickness then we can assume that the potential for the next layer of sites outside the boundary has the same potential as the boundary. If we use this assumption, then after we have solved numerically for the potential of the interior sites, we will find that the average value of the neighbors of a boundary site will not equal the imposed potential. From (10.17) the difference will equal $\Delta x \Delta y \pi \rho(x, y)$. Modify `LaplaceApp` to calculate and display the surface charge density assuming $\Delta x = \Delta y = 1$. Notice that because we are in two dimensions the "surface" charge density, $\Delta x \Delta y \rho(x, y)$, is a linear density of charge per unit length.

b. Using the same system as in Problem 10.10.c find the surface charge density on all boundary sites. Make a reasonable choice for assigning the potential at the corner sites.

c. Model a system with the boundary at a potential $V = 0$ and a centered interior rectangle of $6 \times 12$ at a potential of $V = 10$. Where is the charge density the highest?

d. Repeat if the interior rectangle is placed close to an edge.

**Problem 10.14.** Numerical solution of Poisson's equation

a. Consider a square of linear dimension $L = 25$ whose boundary is fixed at a potential equal to $V = 10$. Assume that the interior region has a uniform charge density $\rho$ such that the total charge is $Q = 1$. Modify `LaplaceApp` to compute the potential distribution for this case. Compare the equipotential surfaces obtained for this case to that found in Problem 10.12.

b. Find the potential distribution if the charge distribution of part (a) is restricted to a $5 \times 5$ square at the center.

c. Find the potential distribution if the charge distribution of part (a) is restricted to a $1 \times 1$ square at the center. How does the potential compare to that of a point charge without the boundary?

***Problem 10.15.** Vector potential and magnetic fields

The magnetic field from arbitrary currents also can be obtained using Poisson's equation. The field is generated from a vector potential $\mathbf{A}$ that satisfies

$$\nabla^2 \mathbf{A} = \mu \mathbf{j} \tag{10.18}$$

where $\mathbf{j}$ is the current density in the wires and $\mu$ is the magnetic permeability. If current flows only in the $z$ direction, then $\mathbf{j} = (0, 0, j_z(x, y))$ and $\mathbf{A} = (0, 0, A_z(x, y))$, and we again have a two-dimensional problem that can be solved using the relaxation method.

Do a simulation that models the magnetic field from an arbitrary number of wires. Combine features of the `ElectricFieldApp` and the `LaplaceApp` programs. The program should read the control and create a current carrying wire when a custom button is clicked. The computation is performed using the animation's `doStep` method to perform a Gauss-Seidel relaxation step. Compute the magnetic field after the computation converges by computing the curl of the vector potential:

$$\mathbf{B} = \nabla \times \mathbf{A}. \tag{10.19}$$

See (10.52) for how to compute the curl when only discrete values are available.

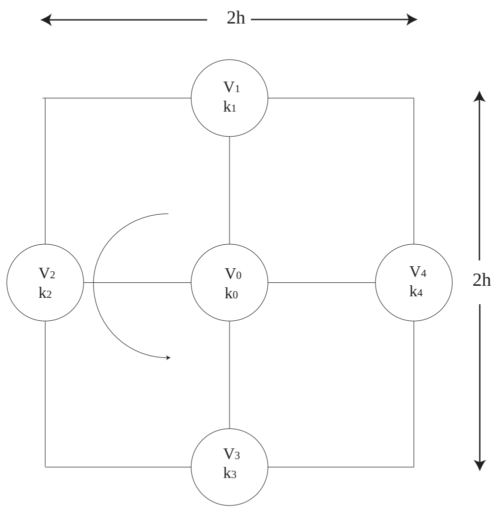


Figure 10.3: The path integral around the grid sites $V_1 \to V_2 \to V_3 \to V_4$ is based on Gauss's law for the electric field.

Dielectrics can be added to the solution of Laplace's equation by adding an array to store the dielectric constant $k$ at every grid site and imposing the condition:

$$D_{1n} = D_{2n}, \tag{10.20}$$

where $\mathbf{D} = k\mathbf{E}$ and $k$ is the dielectric susceptibility. This condition is equivalent to

$$0 = \oint_l k\nabla V \cdot d\mathbf{l} = \oint_l k\frac{\partial V}{\partial n} dl, \tag{10.21}$$

where $\partial V/\partial n$ denotes the derivative of $V$ normal to the contour $l$. If we approximate (10.21) along each edge of length $2h$ using a finite difference for the derivative, we obtain

$$0 = k_1\frac{V_1 - V_0}{h}2h + k_2\frac{V_2 - V_0}{h}2h + k_3\frac{V_3 - V_0}{h}2h + k_4\frac{V_4 - V_0}{h}2h. \tag{10.22}$$

We rearrange terms in (10.22) and find a modified form of (10.12) that includes the dielectric

$$V_0 = \frac{1}{4(k_1 + k_2 + k_3 + k_4)}[k_1V_1 + k_2V_2 + k_3V_3 + k_4V_4], \tag{10.23}$$

where $k_i$ is the average dielectric constant at a site where the electric potential is $V_i$.

**Problem 10.16.** Capacitor with dielectric

a. Modify your Laplace program to include a dielectric medium. That is, create an array of dielectric susceptibilities and implement (10.23) using a relaxation algorithm. Be sure to set the dielectric array elements to unity in free space and inside conductors.

b. Test your algorithm by creating a capacitor consisting of +10 and −10 potential plates near the center of the grid. Initialize the dielectric susceptibility to two in half the capacitor and run the program. Use a `Scalar2DFrame` to display the electric potential, but note that some representations of the scalar field are more appropriate than others. Compare the spacing between the contour lines inside and outside the dielectric. Why does the spacing change?

c. The bound charge on the surface of a dielectric can be computed by subtracting $V(x, y)$ from the average of the potential at the four nearest neighbor sites. You are, in effect, using (10.17) to solve for the charge. Implement this calculation and describe the bound charge on the surface of the dielectric.

## 10.6 Random Walk Solution of Laplace's Equation

In Section 10.5 we found that the solution to Laplace's equation in two dimensions at the point $(x, y)$ is given by

$$V(x,y) = \frac{1}{4}\sum_{i=1}^{4} V(i), \tag{10.24}$$

where $V(i)$ is the value of the potential at the $i$th neighbor. A generalization of this result is that the potential at any point equals the average of the potential on a circle (or sphere in three dimensions) centered about that point.

The relation (10.24) can be given a probabilistic interpretation in terms of random walks (see Problem 10.10d). Suppose that many random walkers are at the site $(x, y)$ and each walker "jumps" to one of its four neighbors (on a square grid) with equal probability $p = 1/4$. From (10.24) we see that the average potential found by the walkers after jumping one step is the potential at $(x, y)$. This relation generalizes to walkers that visit a site on a closed surface with fixed potential. The random walk algorithm for computing the solution to Laplace's equation can be stated as:

1. Begin at a point $(x, y)$ where the value of the potential is desired, and take a step in a random direction.

2. Continue taking steps until the walker reaches the surface. Record $V_b(i)$, the potential at the boundary site $i$. A typical walk is shown in Figure 10.4.

3. Repeat steps (1) and (2) $n$ times and sum the potential found at the surface each time.

4. The value of the potential at the point $(x, y)$ is estimated by

$$V(x,y) = \frac{1}{n}\sum_{i=1}^{n} V_b(i) \tag{10.25}$$

   where $n$ is the total number of random walkers.

**Problem 10.17.** Random walk solution of Laplace's equation

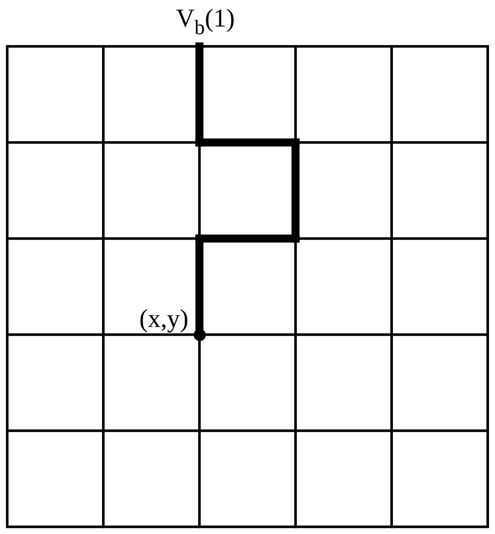


Figure 10.4: A random walk on a $6 \times 6$ grid starting at the point $(x, y) = (3, 3)$ and ending at the boundary site $V_b(3, 6)$ where the potential is recorded.

a. Consider the square region shown in Figure 10.1 and compare the results of the random walk method with the results of the relaxation method (see Problem 10.10c). Try $n = 100$ and $n = 1000$ walkers, and choose a point near the center of the square.

b. Repeat part (a) for other points within the square. Do you need more or less walkers when the potential near the surface is desired? How quickly do your answers converge as a function of $n$?

The disadvantage of the random walk method is that it requires many walkers to obtain a good estimate of the potential at each site. However, if the potential is needed at only a small number of sites, then the random walk method might be more appropriate than the relaxation method which requires the potential to be computed at all points within the region. Another case where the random walk method is appropriate is when the geometry of the boundary is fixed, but the potential in the interior for a variety of different boundary potentials is needed. In this case the quantity of interest is $G(x, y, x_b, y_b)$, the number of times that a walker from the point $(x, y)$ lands at the boundary $(x_b, y_b)$. The random walk algorithm is equivalent to the relation

$$V(x, y) = \frac{1}{n} \sum_{x_b, y_b} G(x, y, x_b, y_b) V(x_b, y_b), \tag{10.26}$$

where the sum is over all sites on the boundary. We can use the same function $G$ for different distributions of the potential on a given boundary. $G$ is an example of a Green's function, a function that you will encounter in advanced treatments of electrodynamics and quantum mechanics (cf. Section 17.9). Of course, if we change the geometry of the boundary, we have to recompute the function $G$.

**Problem 10.18.** Green's function solution of Laplace's equation

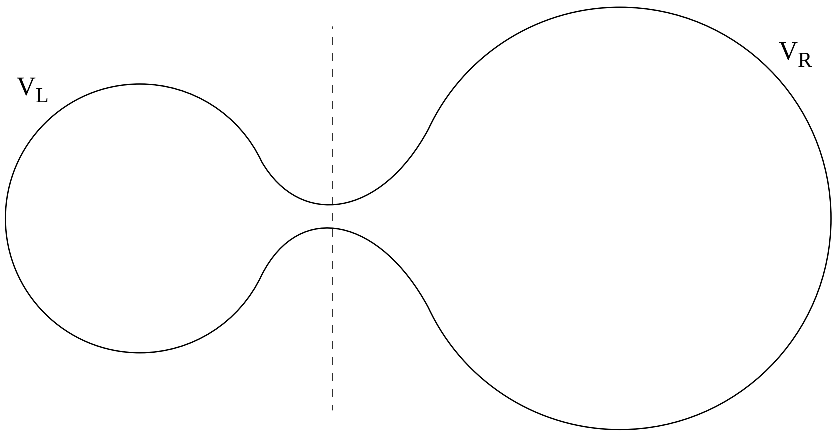


Figure 10.5: Two regions of space connected by a narrow neck. The boundary of the left region has a potential $V_L$, and the boundary of the right region has a potential $V_R$.

a. Compute the Green's function $G(x, y, x_b, y_b)$ for the same geometry considered in Problem 10.17. Use at least 200 walkers at each interior site to estimate $G$. Because of the symmetry of the geometry, you can determine some of the values of $G$ from other values without doing an additional calculation. Store your results for $G$ in a file.

b. Use your results for $G$ found in part (a) to determine the potential at each interior site when the boundary potential is the same as in part (a), except for five boundary sites which are held at $V = 20$. Find the locations of the five boundary sites that maximize the potential at the interior site located at $(3, 5)$. Repeat the calculation to maximize the potential at $(5, 3)$. Use trial and error guided by your physical intuition.

The random walk algorithm can help us gain additional insight into the nature of the solutions of Laplace's equation. Suppose that you have a boundary similar to the one shown in Figure 10.5. The potentials on the left and right boundaries are $V_L$ and $V_R$, respectively. If the neck between the two sides is narrow, it is clear that a random walker starting on the left side has a low probability of reaching the other side. Hence, we can conclude that the potential in the interior of the left side is approximately $V_L$, except very near the neck.

Poisson's equation also can be solved using the random walk method. In this case, the potential is given by

$$V(x, y) = \frac{1}{n}\sum_{\alpha} V(\alpha) + \frac{\pi \Delta x \Delta y}{n}\sum_{i,\alpha} \rho(x_{i,\alpha}, y_{i,\alpha}), \tag{10.27}$$

where $\alpha$ labels the walker, and $i$ labels the site visited by the walker. That is, each time a walker is at site $i$, we add the charge density at that site to the second sum in (10.27).

## 10.7 *Fields Due to Moving Charges

The fact that accelerating charges radiate electromagnetic waves is one of the more important results in the history of physics. In this section we discuss a numerical algorithm for computing the electric and magnetic fields due to the motion of charged particles. The algorithm is very general, but requires some care in its application.

To understand the algorithm, we need a few results that can be found conveniently in Feynman's lectures. We begin with the fact that the scalar potential at the observation point $\mathbf{R}$ due to a stationary particle of charge $q$ is

$$V(\mathbf{R}) = \frac{q}{|\mathbf{R} - \mathbf{r}|}, \tag{10.28}$$

where $\mathbf{r}$ is the position of the charged particle. The electric field is given by

$$\mathbf{E}(\mathbf{R}) = -\frac{\partial V(\mathbf{R})}{\partial \mathbf{R}}, \tag{10.29}$$

where $\partial V(\mathbf{R})/\partial \mathbf{R}$ is the gradient with respect to the coordinates of the observation point. (Note that our notation for the observation point differs from that used in other sections of this chapter.) How do the relations (10.28) and (10.29) change when the particle is moving? We might guess that because it takes a finite time for the disturbance due to a charge to reach the point of observation, we should modify (10.28) by writing

$$V(\mathbf{R}) \overset{?}{=} \frac{q}{r_{\text{ret}}}, \tag{10.30}$$

where

$$r_{\text{ret}} = |\mathbf{R} - \mathbf{r}(t_{\text{ret}})|. \tag{10.31}$$

The quantity $r_{\text{ret}}$ is the separation of the charged particle from the observation point $\mathbf{R}$ at the retarded time $t_{\text{ret}}$. The latter is the time at which the particle was at $\mathbf{r}(t_{\text{ret}})$ such that a disturbance starting at $\mathbf{r}(t_{\text{ret}})$ and traveling at the speed of light would reach $\mathbf{R}$ at time $t$; $t_{\text{ret}}$ is given by the implicit equation

$$t_{\text{ret}} = t - \frac{r_{\text{ret}}(t_{\text{ret}})}{c}, \tag{10.32}$$

where $t$ is the observation time and $c$ is the speed of light.

Although the above reasoning is plausible, the relation (10.30) is not quite correct (cf. Feynman et al. for a derivation of the correct result). We need to take into account that the potential due to the charge is a maximum if the particle is moving toward the observation point and a minimum if it is moving away. The correct result can be written as

$$V(\mathbf{R}, t) = \frac{q}{r_{\text{ret}}\big(1 - \hat{\mathbf{r}}_{\text{ret}} \cdot \mathbf{v}_{\text{ret}}/c\big)}, \tag{10.33}$$

where

$$\mathbf{v}_{\text{ret}} = \frac{d\mathbf{r}(t)}{dt}\Big|_{t=t_{\text{ret}}}. \tag{10.34}$$

and $\hat{\mathbf{r}} = \mathbf{r}/r$.

To find the electric field of a moving charge, we recall that the electric field is related to the time rate of change of the magnetic flux. Hence, we expect that the total electric field at the observation point $\mathbf{R}$ has a contribution due to the magnetic field created by the motion of the charge. We know that the magnetic field due to a moving charge is given by

$$\mathbf{B} = \frac{1}{c}\frac{q\mathbf{v}\times\mathbf{r}}{r^3}. \tag{10.35}$$

If we define the vector potential $\mathbf{A}$ as

$$\mathbf{A} = \frac{q}{r}\frac{\mathbf{v}}{c}, \tag{10.36}$$

we can express $\mathbf{B}$ in terms of $\mathbf{A}$ as

$$\mathbf{B} = \nabla\times\mathbf{A}. \tag{10.37}$$

As we did for the scalar potential $V$, we argue that the correct formula for $\mathbf{A}$ is

$$\mathbf{A}(\mathbf{R},t) = q\frac{\mathbf{v}_{\rm ret}/c}{r_{\rm ret}\big(1-\hat{\mathbf{r}}_{\rm ret}\cdot\mathbf{v}_{\rm ret}/c\big)}. \tag{10.38}$$

Equations (10.33) and (10.38) are known as the Liénard-Wiechert form of the potentials.

The contribution to the electric field $\mathbf{E}$ from $V$ and $\mathbf{A}$ is given by

$$\mathbf{E} = -\nabla V - \frac{1}{c}\frac{\partial\mathbf{A}}{\partial t}. \tag{10.39}$$

The derivatives in (10.39) are with respect to the observation coordinates. The difficulty associated with calculating these derivatives is that the potentials depend on $t_{\rm ret}$, which in turn depends on $\mathbf{R}$, $\mathbf{r}$, and $t$. The result can be expressed as

$$\mathbf{E}(\mathbf{R},t) = \frac{qr_{\rm ret}}{\big(\mathbf{r}_{\rm ret}\cdot\mathbf{u}_{\rm ret}\big)^3}\left[\mathbf{u}_{\rm ret}(c^2-v_{\rm ret}^2) + \mathbf{r}_{\rm ret}\times\big(\mathbf{u}_{\rm ret}\times\mathbf{a}_{\rm ret}\big)\right], \tag{10.40}$$

where

$$\mathbf{u}_{\rm ret} \equiv c\hat{\mathbf{r}}_{\rm ret} - \mathbf{v}_{\rm ret}. \tag{10.41}$$

The acceleration of the particle $\mathbf{a}_{\rm ret} = d\mathbf{v}(t)/dt|_{t=t_{\rm ret}}$. We also can show using (10.37) that the magnetic field $\mathbf{B}$ is given by

$$\mathbf{B} = \hat{\mathbf{r}}_{\rm ret}\times\mathbf{E}. \tag{10.42}$$

The above discussion is not rigorous, but leads to the correct expressions for $\mathbf{E}$ and $\mathbf{B}$. We suggest that you accept (10.40) and (10.42) in the same spirit as you accepted Coulomb's law and the Biot-Savart law. All of classical electrodynamics can be reduced to (10.40) and (10.42) if we assume that the sources of all fields are charges, and all electric currents are due to the motion of charged particles. Note that (10.40) and (10.42) are consistent with the special theory of relativity and reduce to known results in the limit of stationary charges and steady currents.

Although (10.40) and (10.42) are deceptively simple (we do not even have to solve any differential equations), it is difficult to calculate the fields analytically even if the position of a charged particle is an analytic function of time. The difficulty is that we must find the retarded time $t_{\rm ret}$ from (10.32) for each observation position $\mathbf{R}$ and time $t$. For example, consider a charged particle whose motion is sinusoidal, that is, $x(t_{\rm ret}) = A\cos\omega t_{\rm ret}$. To calculate the fields at the position $\mathbf{R} = (X,Y,Z)$ at time $t$, we need to solve the following transcendental equation for $t_{\rm ret}$:

$$t_{\rm ret} = t - \frac{r_{\rm ret}}{c} = t - \frac{1}{c}\sqrt{(X - A\cos^2\omega t_{\rm ret})^2 + Y^2 + Z^2}. \tag{10.43}$$

The solution of (10.43) can be expressed as a root finding problem for which we need to find the zero of the function $f(t_{\text{ret}})$:

$$f(t_{\text{ret}}) = t - t_{\text{ret}} - \frac{r_{\text{ret}}}{c}. \tag{10.44}$$

There are various ways of finding the solution for the retarded time. For example, if the motion of the charges is given by an analytic expression, we can employ *Newton's method* or the *bisection method*. Because we will store the path of the charged particle, we use a simple method that looks for a change in the sign of the function $f(t_{\text{ret}})$ along the path. First find a value $t_a$ such that $f(t_a) > 0$, and another value $t_b$ such that $f(t_b) < 0$. Because $f(t_{\text{ret}})$ is continuous, there is a value of $t_{\text{ret}}$ in the interval $t_a < t_{\text{ret}} < t_b$ such that $f(t_{\text{ret}}) = 0$. This technique is used in the `RadiatingCharge` class shown in Listing 10.6. Note that the particle's path is a sinusoidal oscillation specified in method `evaluate`. The name `evaluate` is used because `RadiatingCharge` implements the `Function` interface, which requires an `evaluate` method. What is the maximum velocity for a particle that moves according to this function?

Listing 10.6: The `RadiatingCharge` class computes the radiating electric and magnetic fields using Liénard-Wiechert potentials.

```java
package org.opensourcephysics.sip.ch10;
import java.awt.Graphics;
import org.opensourcephysics.display.*;
import org.opensourcephysics.numerics.*;

public class RadiatingCharge implements Drawable, Function {
   Circle circle = new Circle(0, 0, 5);
   double t = 0;                                 // time
   double dt = 0.5;                              // time step
   int numPts = 0;                               // current number of points in storage
   double[][] path = new double[3][1024];        // storage for t,x,y
   double[] r = new double[2];
   double[] v = new double[2];
   double[] u = new double[2];
   double[] a = new double[2];
   double vmax;                                  // maximum velocity for charge in units where c = 

   public RadiatingCharge() {
      resetPath();
   }

   private void resizePath() {
      int length = path[0].length;
      if(length>32768) { // drop half the points
         System.arraycopy(path[0], length/2, path[0], 0, length/2);
         System.arraycopy(path[1], length/2, path[1], 0, length/2);
         System.arraycopy(path[2], length/2, path[2], 0, length/2);
         numPts = length/2;
         return;
      }
      double[][] newPath = new double[3][2*length]; // new path
      System.arraycopy(path[0], 0, newPath[0], 0, length);
      System.arraycopy(path[1], 0, newPath[1], 0, length);
      System.arraycopy(path[2], 0, newPath[2], 0, length);
      path = newPath;
   }

   void step() {
      t += dt;
      if(numPts>=path[0].length) {
         resizePath();
      }
      path[0][numPts] = t;
      path[1][numPts] = evaluate(t); // x position of charge
      path[2][numPts] = 0;
      numPts++;
   }

   void resetPath() {
      numPts = 0;
      t = 0;
      path = new double[3][1024];     // storage for t,x,y
      path[0][numPts] = t;
      path[1][numPts] = evaluate(t); // x position of charge
      path[2][numPts] = 0;
      numPts++; // initial position has been added
   }

   void electrostaticField(double x, double y, double[] field) {
      double dx = x-path[1][0];
      double dy = y-path[2][0];
      double r2 = dx*dx+dy*dy;
      double r3 = r2*Math.sqrt(r2);
      double ex = dx/r3;
      double ey = dy/r3;
      field[0] = ex;
      field[1] = ey;
      field[2] = 0; // magnetic field
   }

   double dsSquared(int i, double t, double x, double y) {
      double dt = t-path[0][i];
      double dx = x-path[1][i];
      double dy = y-path[2][i];
      return dx*dx+dy*dy-dt*dt;
   }

   void calculateRetardedField(double x, double y, double[] field) {
      int first = 0;
      int last = numPts-1;
      double ds_first = dsSquared(first, t, x, y);
      if(ds_first>=0) { // field has not yet propagated to the location
         electrostaticField(x, y, field);
         return;
      }
      while((ds_first<0)&&(last-first)>1) {
         int i = first+(last-first)/2; // bisect the interval
         double ds = dsSquared(i, t, x, y);
         if(ds<=0) {
            ds_first = ds;
            first = i;
         } else {
            last = i;
         }
      }
      double t_ret = path[0][first]; // time where ds changes sign
      r[0] = x-evaluate(t_ret);                      // evaluate x at retarded time
      r[1] = y;                                      // evaluate y at retarded time
      v[0] = Derivative.first(this, t_ret, dt);      // derivative of x at retarded time
      v[1] = 0;                                      // derivative of y at retarded time
      a[0] = Derivative.second(this, t_ret, dt);     // acceleration of x at retarded time
      a[1] = 0;                                      // acceleration of y at retarded time
      double rMag = Vector2DMath.mag2D(r); // magnitdue of r
      u[0] = r[0]/rMag-v[0];
      u[1] = r[1]/rMag-v[1];
      double r_dot_u = Vector2DMath.dot2D(r, u);
      double k = rMag/r_dot_u/r_dot_u/r_dot_u;
      double u_cross_a = Vector2DMath.cross2D(u, a); // u cross a is perpendicular to plane
      double[] temp = {r[0], r[1]};
      temp = Vector2DMath.crossZ(temp, u_cross_a); // temp now equals r cross u
      double c2v2 = 1-Vector2DMath.dot2D(v, v); // (c*c - v*v) where c = 1
      double ex = k*(u[0]*c2v2+temp[0]);
      double ey = k*(u[1]*c2v2+temp[1]);
      field[0] = ex;
      field[1] = ey;
      field[2] = k*Vector2DMath.cross2D(temp, r)/rMag;
   }

   public void draw(DrawingPanel panel, Graphics g) {
      circle.setX(evaluate(t));
      circle.draw(panel, g); // draw the charged particle on the screen
   }

   public double evaluate(double t) {
      return 5*Math.cos(t*vmax/5.0);
   }
}
```

The `RadiatingCharge` class computes the electric field due to an oscillating charge using the Liénard-Wiechert potentials. We choose units such that the speed of light $c = 1$. As the charge moves, it stores its $i$th data point in a two-dimensional array `path[3][i]` containing the time, its $x$-position, and its $y$-position. To find the retarded time at the position $(x, y)$, we use the `dsSquared` method to compute the square of the space-time interval between the given location and points along the path. The square of the space-time separation is defined as

$$\Delta s^2 = \Delta x^2 + \Delta y^2 - c^2 \Delta t^2, \tag{10.45}$$

where $\Delta x = x - x_{\text{path}}$, $\Delta y = y - y_{\text{path}}$, and $\Delta t = t - t_{\text{path}}$. The last point on the path contains the current position of the charge so $\Delta s^2$ must be positive because $\Delta t$ is zero (unless the charge is at the observation point $(x, y)$ in which case $\Delta s^2$ is zero and the field is infinite due to the $1/r^2$ dependence). The `calcRetardedField` method evaluates $\Delta s^2$ at the first point in the trajectory to determine if it is negative. We assume the charge was stationary for $t < 0$ and compute the electrostatic field if $\Delta s^2$ is positive at the trajectory's first point where $t = 0$. If $\Delta s^2$ is negative at the trajectory's first point, we repeatedly bisect the path into smaller and smaller segments while checking to see if $\Delta s^2$ remains negative at the beginning of the segment and positive at the end. In this way we can find the retarded time when we have a path segment bounded by two data points. Note that the `RadiatingCharge` class uses the `Vector2DMath` class to perform the necessary vector arithmetic. This helper class, is not listed but is available in ch10 code package.

The `RadiatingEFieldApp` program is shown in Listing 10.7. It displays the electric field in the $x$-$y$ plane using a `Vector2DFrame`. The `calculateFields` method computes the retarded field at every grid point. The simulation's `doStep` method invokes this method after it moves the charge.

Listing 10.7: The `RadiatingEFieldApp` program computes the radiating electric and magnetic fields using Liénard-Wiechert potentials.

```java
package org.opensourcephysics.sip.ch10;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.frames.Vector2DFrame;

public class RadiatingEFieldApp extends AbstractSimulation {
   Vector2DFrame frame = new Vector2DFrame("x", "y", "Electric field");
   RadiatingCharge charge = new RadiatingCharge();
   int gridSize;      // linear dimension of grid used to compute fields
   double[][][] Exy; // x and y components of electric field
   double xmin = -20, xmax = 20, ymin = -20, ymax = 20;

   public RadiatingEFieldApp() {
      frame.setPreferredMinMax(xmin, xmax, ymin, ymax);
      frame.setZRange(false, 0, 0.2);
      frame.addDrawable(charge);
   }

   public void initialize() {
      gridSize = control.getInt("size");
      Exy = new double[2][gridSize][gridSize];
      charge.vmax = control.getDouble("vmax"); // maximum speed of charge
      charge.dt = control.getDouble("dt");
      frame.setAll(Exy);
      initArrays();
   }
```

```
  private void initArrays() {
    charge.resetPath();
    calculateFields();
  }

  private void calculateFields() {
    double[] fields = new double[3]; // Ex, Ey, Bz
    for(int i = 0;i<gridSize;i++) {
      for(int j = 0;j<gridSize;j++) {
        double x = frame.indexToX(i);                // x location where we calculate the
        double y = frame.indexToY(j);                // y location where we calculate the
        charge.calculateRetardedField(x, y, fields); // return the retarded time
        Exy[0][i][j] = fields[0];                    // Ex
        Exy[1][i][j] = fields[1];                    // Ey
      }
    }
    frame.setAll(Exy);
  }

  public void reset() {
    control.setValue("size", 31);
    control.setValue("dt", 0.5);
    control.setValue("vmax", 0.9);
    initialize(); // initialize the model
  }

  protected void doStep() {
    charge.step();
    calculateFields();
  }

  public static void main(String[] args) {
    SimulationControl.createApp(new RadiatingEFieldApp());
  }
}
```

**Problem 10.19.** Field lines from an accelerating charge

a. Read the code for `RadiatingEFieldApp` carefully to understand the correspondence between the program and the analytic results (10.40) and (10.42) discussed in the text.

b. Describe qualitatively the nature of the electric and magnetic fields from an oscillating point charge. How does the electric field differ from that of a static charge at the origin? What happens as the speed increases? The physics breaks down if the maximum speed is greater than $c$. Does the algorithm break down? Explain.

c. Modify the program to show the magnetic field in the $x$-$y$ plane using a `Scalar2DFrame` to show the $B_z$ vector component.

d. Modify the program to observe a charge moving with uniform circular motion about the origin. What happens as the speed of the charge approaches the speed of light?

**Problem 10.20.** Spatial dependence of the radiating fields

a. As waves propagate from an accelerating point source, the total power that passes through a spherical surface of radius $R$ remains constant. Because the surface area is proportional to $R^2$, the power per unit area or intensity is proportional to $1/R^2$. Also, because the intensity is proportional to $E^2$, we expect that $E \propto 1/R$ far from the source. Modify the program to verify this result for a charge that is oscillating along the $x$-axis according to $x(t) = 0.2\cos t$. Plot $|E|$ as a function of the observation time $t$ for a fixed position such as $\mathbf{R} = (10, 10, 0)$. The field should oscillate in time. Find the amplitude of this oscillation. Next double the distance of the observation point from the origin. How does the amplitude depend on $R$?

b. Repeat part (a) for several directions and distances. Generate a polar diagram showing the amplitude as a function of angle in the $x$-$y$ plane. Is the radiation greatest along the line in which the charge oscillates?

**Problem 10.21.** Fields from a charge moving at constant velocity

a. Use `RadiationApp` to calculate **E** due to a charged particle moving at constant velocity toward the origin, for example, $x(t_{\rm ret}) = 1 - 2t_{\rm ret}$. Take a snapshot at $t = 0.5$ and compare the field lines with those you expect from a stationary charge.

b. Modify `RadiationApp` so that $x(t_{\rm ret}) = 1 - 2t_{\rm ret}$ for $t_{\rm ret} < 0.5$ and $x(t_{\rm ret}) = 0$ for $t_{\rm ret} > 0.5$. Describe the field lines for $t > 0.5$. Does the particle accelerate at any time? Is there any radiation?

**Problem 10.22.** Frequency dependence of an oscillating charge

a. The radiated power at any point in space is proportional to $E^2$. Plot $|E|$ versus time at a fixed observation point (for example, $X = 10, Y = Z = 0$), and calculate the frequency dependence of the amplitude of $|E|$ due to a charge oscillating at the frequency $\omega$. It is shown in standard textbooks that the power associated with radiation from an oscillating dipole is proportional to $\omega^4$. How does the $\omega$-dependence that you measured compare to that for dipole radiation? Repeat for a much bigger value of $R$, and explain any differences.

b. Repeat part (a) for a charge moving in a circle. Are there any qualitative differences?

## 10.8 *Maxwell's Equations

In Section 10.7 we found that accelerating charges produce electric and magnetic fields that depend on position and time. We now investigate the direct relation between changes in **E** and **B** given by the differential form of Maxwell's equations:

$$\frac{\partial \mathbf{B}}{\partial t} = -\frac{1}{c}\nabla \times \mathbf{E} \tag{10.46}$$

$$\frac{\partial \mathbf{E}}{\partial t} = c\nabla \times \mathbf{B} - 4\pi\mathbf{j}, \tag{10.47}$$

where $\mathbf{j}$ is the electric current density. We can regard (10.46) and (10.47) as the basis of electrodynamics. In addition to (10.46) and (10.47), we need the relation between $\mathbf{j}$ and the charge density $\rho$ that expresses the conservation of charge:

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot \mathbf{j}. \tag{10.48}$$

A complete description of electrodynamics requires (10.46), (10.47), and (10.48) and the initial values of all currents and fields.

For completeness, we obtain the Maxwell's equations that involve $\nabla \cdot \mathbf{B}$ and $\nabla \cdot \mathbf{E}$ by taking the divergence of (10.46) and (10.47), substituting (10.48) for $\nabla \cdot \mathbf{j}$, and then integrating over time. If the initial fields are zero, we obtain (using the relation $\nabla \cdot (\nabla \times \mathbf{a}) = 0$ for any vector $\mathbf{a}$):

$$\nabla \cdot \mathbf{E} = 4\pi\rho \tag{10.49}$$

$$\nabla \cdot \mathbf{B} = 0. \tag{10.50}$$

If we introduce the electric and magnetic potentials, it is possible to convert the first-order equations (10.46) and (10.47) to second-order differential equations. However, the familiar first-order equations are better suited for numerical analysis. To solve (10.46) and (10.47) numerically, we need to interpret the curl and divergence of a vector. As its name implies, the curl of a vector measures how much the vector twists around a point. A coordinate free definition of the curl of an arbitrary vector $\mathbf{W}$ is

$$(\nabla \times \mathbf{W}) \cdot \hat{\mathbf{S}} = \lim_{S \to 0} \frac{1}{S} \oint_C \mathbf{W} \cdot d\mathbf{l}, \tag{10.51}$$

where $\mathbf{S}$ is the area of any surface bordered by the closed curve $C$, and $\hat{\mathbf{S}}$ is a unit vector normal to the surface $S$.

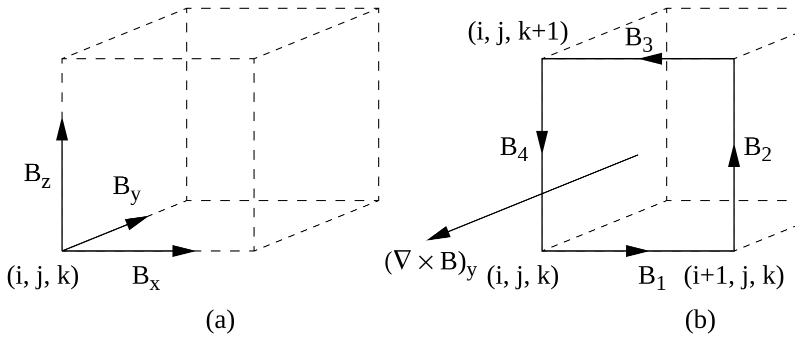
Equation (10.51) gives the component of $\nabla \times \mathbf{W}$ in the direction of $\hat{\mathbf{S}}$ and suggests a way of computing the curl numerically. We divide space into cubes of linear dimension $\Delta l$. The rectangular components of $\mathbf{W}$ can be defined either on the edges or on the faces of the cubes. We compute the curl using both definitions. We first consider a vector $\mathbf{B}$ that is defined on the edges of the cubes so that the curl of $\mathbf{B}$ is defined on the faces. (We use the notation $\mathbf{B}$ because we will find that it is convenient to define the magnetic field in this way.) Associated with each cube is one edge vector and one face vector. We label the cube by the coordinates corresponding to its lower left front corner; the three components of $\mathbf{B}$ associated with this cube are shown in Figure 10.6a. The other edges of the cube are associated with $B$ vectors defined at neighboring cubes.

The discrete version of (10.51) for the component of $\nabla \times \mathbf{B}$ defined on the front face of the cube $(i, j, k)$ is

$$(\nabla \times \mathbf{B}) \cdot \hat{\mathbf{S}} = \frac{1}{(\Delta l)^2} \sum_{i=1}^{4} B_i \Delta l_i, \tag{10.52}$$

where $S = (\Delta l)^2$, and $B_i$ and $l_i$ are shown in Figures 10.6b and 10.6c, respectively. Note that two of the $B_i$ are associated with neighboring cubes.

The components of a vector also can be defined on the faces of the cubes. We call this vector $\mathbf{E}$ because it will be convenient to define the electric field in this way. In Figure 10.7a we show the components of $\mathbf{E}$ associated with the cube $(i, j, k)$. Because $\mathbf{E}$ is normal to a cube face, the



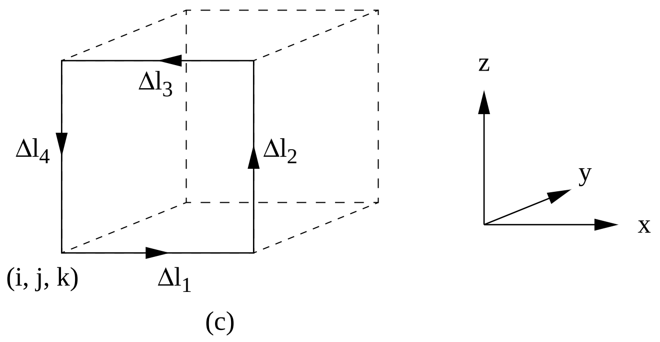


Figure 10.6: Calculation of the curl of **B** defined on the edges of a cube. (a) The edge vector **B** associated with cube $(i, j, k)$. (b) The components $B_i$ along the edges of the front face of the cube. $B_1 = B_x(i, j, k)$, $B_2 = B_z(i+1, j, k)$, $B_3 = -B_x(i, j, k+1)$, and $B_4 = -B_z(i, j, k)$. (c) The vector components $\mathbf{\Delta l}_i$ on the edges of the front face. (The $y$-component of $\nabla \times \mathbf{B}$ defined on the face points in the negative $y$ direction.)

components of $\nabla \times \mathbf{E}$ lie on the edges. The components $E_i$ and $l_i$ are shown in Figures 10.7b and 10.7c respectively. The form of the discrete version of $\nabla \times \mathbf{E}$ is similar to (10.52) with $B_i$ replaced by $E_i$, where $E_i$ and $l_i$ are shown in Figures 10.7b and 10.7c respectively. The $z$-component of $\nabla \times \mathbf{E}$ is along the left edge of the front face.

A coordinate free definition of the divergence of the vector field **W** is

$$\nabla \cdot \mathbf{W} = \lim_{V \to 0} \frac{1}{V} \oint_S \mathbf{W} \cdot d\mathbf{S}, \tag{10.53}$$

where $V$ is the volume enclosed by the closed surface **S**. The divergence measures the average flow of the vector through a closed surface. An example of the discrete version of (10.53) is given in (10.54).

We now discuss where to define the quantities $\rho$, **j**, **E**, and **B** on the grid. It is natural to define the charge density $\rho$ at the center of a cube. From the continuity equation (10.48), we see that this definition leads us to define **j** at the faces of the cube. Hence, each face of a cube has a number associated with it corresponding to the current density flowing parallel to the outward normal to that face. Given the definition of **j** on the grid, we see from (10.47) that the electric field **E** and **j** should be defined at the same places, and hence we define the electric field on the faces of the cubes. Because **E** is defined on the faces, it is natural to define the magnetic field **B** on the edges of the cubes. Our definitions of the vectors **j**, **E**, and **B** on the grid are now complete.

We label the faces of cube $c$ by the symbol $f_c$. If we use the simplest finite difference method with a discrete time step $\Delta t$ and discrete spatial interval $\Delta x = \Delta y = \Delta z \equiv \Delta l$, we can write the continuity equation as:

$$\big[\rho(c, t + \frac{1}{2}\Delta) - \rho(c, t - \frac{1}{2}\Delta t)\big] = -\frac{\Delta t}{\Delta l} \sum_{f_c=1}^{6} j(f_c, t). \tag{10.54}$$

The factor of $1/\Delta l$ comes from the area of a face $(\Delta l)^2$ used in the surface integral in (10.53) divided by the volume $(\Delta l)^3$ of a cube. In the same spirit, the discretization of (10.47) can be written as:

$$E(f, t + \frac{1}{2}\Delta t) - E(f, t - \frac{1}{2}\Delta t) = \Delta t\big[\nabla \times \mathbf{B} - 4\pi j(f, t)\big]. \tag{10.55}$$

Note that **E** in (10.55) and $\rho$ in (10.54) are defined at different times than **j**. As usual, we choose units such that $c = 1$.

We next need to define a square around which we can discretize the curl. If **E** is defined on the faces, it is natural to use the square that is the border of the faces. As we have discussed, this choice implies that we should define the magnetic field on the edges of the cubes. We write (10.55) as:

$$E(f, t + \frac{1}{2}\Delta t) - E(f, t - \frac{1}{2}\Delta t) = \Delta t\big[\frac{1}{\Delta l} \sum_{e_f=1}^{4} B(e_f, t) - 4\pi j(f, t)\big], \tag{10.56}$$

where the sum is over $e_f$, the four edges of the face $f$ (see Figure 10.7b). Note that $B$ is defined at the same time as $j$. In a similar way we can write the discrete form of (10.46) as:

$$B(e, t + \Delta t) - B(e, t) = -\frac{\Delta t}{\Delta l} \sum_{f_e=1}^{4} E(f_e, t + \frac{1}{2}\Delta t), \tag{10.57}$$

where the sum is over $f_e$, the four faces that share the same edge $e$ (see Figure 10.7b).

We now have a well defined algorithm for computing the spatial dependence of the electric and magnetic field, the charge density, and the current density as a function of time. This algorithm was developed by Yee, an electrical engineer, in 1966, and independently by Visscher, a physicist, in 1988 who also showed that all of the integral relations and other theorems that are satisfied by the continuum fields also are satisfied for the discrete fields.

Usually, the most difficult part of this algorithm is specifying the initial conditions because we cannot simply place a charge somewhere. The reason is that the initial fields appropriate for this charge would not be present. Indeed, our rules for updating the fields and the charge densities reflect the fact that the electric and magnetic fields do not appear instantaneously at all positions in space when a charge appears, but instead evolve from the initial appearance of a charge. Of course, charges do not appear out of nowhere, but appear by disassociating from neutral objects.

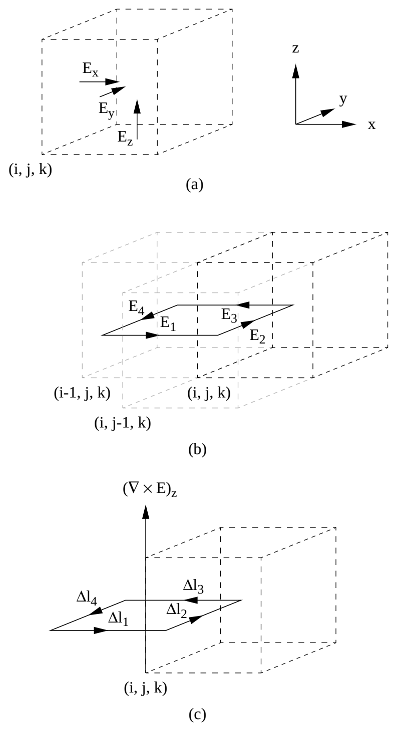


Figure 10.7: Calculation of the curl of the vector **E** defined on the faces of a cube. (a) The face vector **E** associated with the cube $(i, j, k)$. The components associated with the left, front, and bottom faces are $E_x(i, j, k), E_y(i, j, k), E_z(i, j, k)$ respectively. (b) The components $E_i$ on the faces that share the front left edge of the cube $(i, j, k)$. $E_1 = E_x(i, j-1, k), E_2 = E_y(i, j, k), E_3 = -E_x(i, j, k), \text{and} E_4 = -E_y(i-1, j, k)$. The cubes associated with $E_1$ and $E_4$ also are shown. (c) The vector components $\Delta l_i$ on the faces that share the left front edge of the cube. (The $z$-component of the curl of **E** defined on the left edge points in the positive $z$ direction.)

Conceptually, the simplest initial condition corresponds to two charges of opposite sign moving oppositely to each other. This condition corresponds to an initial current on one face. From this current, a charge density and electric field appears using (10.54) and (10.56), respectively, and a magnetic field appears using (10.57).

Because we cannot compute the fields for an infinite lattice, we need to specify the boundary conditions. The easiest method is to use fixed boundary conditions such that the fields vanish at the edges of the lattice. If the lattice is sufficiently large, fixed boundary conditions are a reasonable approximation. However, fixed boundary conditions usually lead to nonphysical reflections off the edges, and a variety of approaches have been used including boundary conditions equivalent to a conducting medium that gradually absorbs the fields. In some cases physically motivated boundary conditions can be employed. For example, in simulations of microwave cavity resonators (see Problem 10.24), the appropriate boundary conditions are that the tangential component of **E** and the normal component of **B** vanish at the boundary.

As we have noted, **E** and $\rho$ are defined at different times than **B** and **j**. This half-step approach leads to well behaved equations that are stable over a range of parameters. An analysis of the stability requirement for the Yee-Visscher algorithm shows that the time step $\Delta t$ must be smaller than the spatial grid $\Delta l$ by:

$$c\Delta t \leq \frac{\Delta l}{\sqrt{3}}. \qquad \text{(stability requirement)} \tag{10.58}$$

The `Maxwell` class implements the Visscher-Yee finite difference algorithm for solving Maxwell's equations. The field and current data are stored in multi-dimensional arrays `E`, `B`, and `J`. The first index determines the vector component. The last three indices represent the three spatial coordinates. The `current` method models a positive current flowing for one time unit. This current flow produces both electric and magnetic fields. Because charge is conserved, the current flow produces an electrostatic dipole. Negative charge remains at the source and a positive charge is deposited at the destination. Note that the `doStep` method invokes a `damping` method that reduces the fields at points near the boundaries, thereby absorbing the emitted radiation and reducing the reflected electromagnetic waves. Your understanding of the Yee-Visscher algorithm for finding solutions to Maxwell's equations will be enhanced by carefully reading the `MaxwellApp` program and the `Maxell` class.

Listing 10.8: The `Maxwell` class implements the Yee-Visscher finite difference approximation to Maxwell's equations.

```
package org.opensourcephysics.sip.ch10;

// revised 12/14/04 JT
public class Maxwell {
   // static variables determine units and time scale
   static final double pi4 = 4*Math.PI;
   static final double dt = 0.03;
   static final double dl = 0.1;
   static final double escale = dl/(4*Math.PI*dt);
   static final double bscale = escale*dl/dt;
   static final double jscale = 1;
   double dampingCoef = 0.1; // damping coefficient near boundaries
   int size;
   double t;                 // time
   double[][][][] E, B, J;

   public Maxwell(int size) {
      this.size = size;
      //3D arrays for electric field, magnetic field, and current
      //last three indices indicate location, first index indicates x, y, or z component
      E = new double[3][size][size][size];
      B = new double[3][size][size][size];
      J = new double[3][size][size][size];
   }

   public void doStep() {
      current(t); // update the current
      computeE(); // step electric field
      computeB(); // step magnetic field
      damping();  // damp transients
      t += dt;
   }

   void current(double t) {
      final int mid = size/2;
      double delta = 1.0;
      for(int i = -3;i<5;i++) {
         J[mid+i][mid][mid][0] = (t<delta) ? +1 : 0;
      }
   }

   void computeE() {
      for(int x = 1;x<size-1;x++) {
         for(int y = 1;y<size-1;y++) {
            for(int z = 1;z<size-1;z++) {
               double curlBx = (B[1][x][y][z]-B[1][x][y][z+1]+B[2][x][y+1][z]-B[2][x][y][z])/
               E[0][x][y][z] += dt*(curlBx-pi4*J[0][x][y][z]);
               double curlBy = (B[2][x][y][z]-B[2][x+1][y][z]+B[0][x][y][z+1]-B[0][x][y][z])/
               E[1][x][y][z] += dt*(curlBy-pi4*J[1][x][y][z]);
               double curlBz = (B[0][x][y][z]-B[0][x][y+1][z]+B[1][x+1][y][z]-B[1][x][y][z])/
               E[2][x][y][z] += dt*(curlBz-pi4*J[2][x][y][z]);
            }
         }
      }
   }

   void computeB() {
      for(int x = 1;x<size-1;x++) {
         for(int y = 1;y<size-1;y++) {
            for(int z = 1;z<size-1;z++) {
               double curlEx = (E[2][x][y][z]-E[2][x][y-1][z]+E[1][x][y][z-1]-E[1][x][y][z])/
               B[0][x][y][z] -= dt*curlEx;
               double curlEy = (E[0][x][y][z]-E[0][x][y][z-1]+E[2][x-1][y][z]-E[2][x][y][z])/
               B[1][x][y][z] -= dt*curlEy;
               double curlEz = (E[1][x][y][z]-E[1][x-1][y][z]+E[0][x][y-1][z]-E[0][x][y][z])/
               B[2][x][y][z] -= dt*curlEz;
            }
         }
      }
   }

   void damping() {
      for(int i = 0;i<size;i++) {
         for(int j = 0;j<size;j++) {
            for(int w = 0;w<4;w++) { // w used to index cell near boundary subject to dampin
               for(int comp = 0;comp<3;comp++) {
                  E[comp][w][i][j] -= dampingCoef*E[comp][w][i][j];
                  E[comp][size-w-1][i][j] -= dampingCoef*E[comp][size-w-1][i][j];
                  E[comp][i][w][j] -= dampingCoef*E[comp][i][w][j];
                  E[comp][i][size-w-1][j] -= dampingCoef*E[comp][i][size-w-1][j];
                  E[comp][i][j][w] -= dampingCoef*E[comp][i][j][w];
                  E[comp][i][j][size-w-1] -= dampingCoef*E[comp][i][j][size-w-1];
                  B[comp][w][i][j] -= dampingCoef*B[comp][w][i][j];
                  B[comp][size-w-1][i][j] -= dampingCoef*B[comp][size-w-1][i][j];
                  B[comp][i][w][j] -= dampingCoef*B[comp][i][w][j];
                  B[comp][i][size-w-1][j] -= dampingCoef*B[comp][i][size-w-1][j];
                  B[comp][i][j][w] -= dampingCoef*B[comp][i][j][w];
                  B[comp][i][j][size-w-1] -= dampingCoef*B[comp][i][j][size-w-1];
               }
            }
         }
      }
   }
}
```

Listing 10.9: The `MaxwellApp` program computes and displays the electric field by solving Maxwell's equations.

```
package org.opensourcephysics.sip.ch10;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.frames.Vector2DFrame;

public class MaxwellApp extends AbstractSimulation {
   Vector2DFrame frame = new Vector2DFrame("x", "y", "EField in XY Plane");
   int size;
   Maxwell maxwell;
   double[][][] Exy; //  x and y components of E for middle plane in z direction

   public MaxwellApp() {
      frame.setZRange(false, 0, 1.0);
   }

   public void reset() {
      control.setValue("size", 31);
      control.setValue("dt", 0.5);
   }

   public void initialize() {
      size = control.getInt("size");
      Exy = new double[2][size][size];
      maxwell = new Maxwell(size);
      frame.setAll(Exy);
      frame.setPreferredMinMax(0, Maxwell.dl*size, Maxwell.dl*size, 0);
      plotField();
   }
   protected void doStep() {
      maxwell.doStep();
      plotField();
      frame.setMessage("t="+decimalFormat.format(maxwell.t));
   }

   void plotField() {
      double[][][][] E = maxwell.E; // electric field
      int mid = size/2;
      for(int i = 0;i<size;i++) {
         for(int j = 0;j<size;j++) {
            Exy[0][i][j] = E[i][j][mid][0]; // Ex
            Exy[1][i][j] = E[i][j][mid][1]; // Ey
         }
      }
      frame.setAll(Exy);
   }

   public static void main(String[] args) {
      SimulationControl.createApp(new MaxwellApp());
   }
}
```

The `MaxwellApp` program shows the electric field in the $x$-$y$ plane. The $x$-$y$ components of the electric field are represented by arrows, whose length is fixed and whose color indicates the field magnitude at each position where the field is defined.

**Problem 10.23.** Fields from a current loop

a. A steady current in the middle of the $x$-$y$ plane is turned on at $t = 0$ and left on for one time unit. Before running the program, predict what you expect to see. Compare your expectations with the results of the simulation. Use $\Delta t = 0.03$, $\Delta l = 0.1$, and take the number of cubes in each direction to be `n(1)` $=$ `n(2)` $=$ `n(3)` $= 8$.

b. Add a plot of the magnetic field. Where should the viewing plane be placed to produce the best visualization? How should the plane be oriented? Predict what you expect to see before you run the simulation.

c. Verify the stability requirement (10.58), by running your program with $\Delta t = 0.1$ and $\Delta l = 0.1$. Then try $\Delta t = 0.05$ and $\Delta l = \Delta t\sqrt{3}$. What happens to the results in part (a) if the stability requirement is not satisfied?

d. Modify the current density in part (a) so that **j** oscillates sinusoidally. What happens to the electric and magnetic field vectors?

e. How much must you change the factor `dampingCoef` in the damping method before you can visually see a difference in the simulation? What problems occur when the damping is removed?

f.* The amplitude of the fields far from the current loop should be characteristic of radiation fields for which the amplitude falls off as $1/r$, where $r$ is the distance from the current loop to the observation point. Do a simulation to detect this dependence if you have sufficient computer resources.

**Problem 10.24.** Microwave cavity resonators

a. Cavity resonators are a practical way of storing energy in the form of oscillating electric and magnetic fields without losing as much energy as would be dissipated in a resonant LC circuit. Consider a cubical resonator of linear dimension $L$ whose walls are made of a perfectly conducting material. The tangential components of **E** and the normal component of **B** vanish at the walls. Standing microwaves can be set up in the box of the form (cf. Reitz et al.)

$$E_x = E_{x0} \cos k_x x \sin k_y y \sin k_z z \, e^{i\omega t} \tag{10.59a}$$

$$E_y = E_{y0} \cos k_y y \sin k_x x \sin k_z z \, e^{i\omega t} \tag{10.59b}$$

$$E_z = E_{z0} \cos k_z z \sin k_x x \sin k_y y \, e^{i\omega t}. \tag{10.59c}$$

The wave vector $\mathbf{k} = (k_x, k_y, k_z) = (m_x\pi/L, m_y\pi/L, m_z\pi/L)$, where $m_x$, $m_y$, and $m_z$ are integers. A particular mode is labeled by the integers $(m_x, m_y, m_z)$. The initial electric field is perpendicular to **k**, and $\omega = ck$. Implement the boundary conditions at $(x = 0, y = 0, z = 0)$ and $(x = L, y = L, z = L)$. Set $\Delta t = 0.05$, $\Delta l = 0.1$, and $L = 1$. At $t = 0$, set $\mathbf{B} = 0$, $\mathbf{j} = 0$ (there are no currents within the cavity), and use (10.59) with $(m_x, m_y, m_z) = (0, 1, 1)$, and $E_{x0} = 1$. Plot the field components at specific positions as a function of $t$ and find the resonant frequency $\omega$. Compare your computed value of $\omega$ with the analytical result. Do the magnetic fields change with time? Are they perpendicular to **k** and **E**?

b. Repeat part (a) for two other modes.

c. Repeat part (a) with a uniform random noise added to the initial field at all positions. Assume the amplitude of the noise is $\delta$ and describe the resulting fields for $\delta = 0.1$. Are they similar to those without noise? What happens for $\delta = 0.5$? More quantitative results can be found by computing the power spectrum $|E(\omega)|^2$ for the electric field at a few positions. What is the order of magnitude of $\delta$ for which the maximum of $|E(\omega)|^2$ at the standing wave frequency is swamped by the noise?

d. Change the shape of the container slightly by removing a $0.1 \times 0.1$ cubical box from each of the corners of the original resonator. Do the standing wave frequencies change? Determine the standing wave frequency by adding noise to the initial fields and looking at the power spectrum. How do the standing wave patterns change?

e. Change the shape of the container slightly by adding a $0.1 \times 0.1$ cubical box at the center of one of the faces of the original resonator. Do the standing wave frequencies change? How do the standing wave patterns change?

f. Cut a $0.2 \times 0.2$ square hole in a face in the $y$-$z$ plane, and double the computational region in the $x$ direction. Begin with a $(0, 1, 1)$ standing wave, and observe how the fields "leak" out of the hole.

**Problem 10.25.** Billiard microwave cavity resonators

a. Repeat Problem 10.24a for $L_x = L_y = 2$, $L_z = 0.2$, $\Delta l = 0.1$, and $\Delta t = 0.05$. Indicate the magnitude of the electric field in the $L_z = 0.1$ plane by a color code. Choose an initial normal mode field distribution and describe the pattern that you obtain. Then repeat your calculation for a random initial field distribution.

b. Place an approximately circular conductor in the middle of the cavity of radius $r = 0.4$. Describe the patterns that you see. Such a geometry leads to chaotic trajectories for particles moving within such a cavity (see Project 6.26). Is there any evidence of chaotic behavior in the field pattern?

c. Repeat part (b) with the circular conductor placed off center.

## 10.9 Projects

Part of the difficulty in understanding electromagnetic phenomena is visualizing its three-dimensional nature. Many interesting problems can be posed based on the simple, but nontrivial question of how three-dimensional electromagnetic fields can best be represented visually in various contexts (cf. Belcher and Olbert). However, we have not suggested projects in this area because of their difficulty.

Many of the techniques used in this chapter, for example, the random walk method and the relaxation method for solving Laplace's equation, have applications in other fields, especially problems in fluid flow and transport. Similarly, the multigrid method, discussed in Project 10.26, has far reaching applications.

**Project 10.26.** Multigrid method

In general, the relaxation method for solving Laplace's equation is very slow even using overrelaxation. The reason is that the local updates of the relaxation method cannot quickly take into account effects at very large length scales. The *multigrid method* greatly improves performance by using relaxation at many length scales. The important idea is to use a relaxation method to find the values of the potential on coarser and coarser grids, and then use the coarse grid values to determine the fine grid values. The fine grid relaxation updates take into account effects at short length scales. If we define the initial grid by a lattice spacing $b = 1$, then the coarser grids are characterized by $b = 2^n$, where $n$ determines the coarseness of the grid and is known as the grid level. We need to decide how to use the fine grid values of the potential to assign values to a coarser grid, and then how to use a coarse grid to assign values to a finer grid. The first step is called *prolongation* and the second step is called *restriction*. There is some flexibility on how to do these two operations. We discuss one approach.

We define the centers of the sites of the coarse grid to be located at the centers of every other site of the fine grid. That is, if the set $\{i, j\}$ represents the positions of the sites of the 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 $4 \times 4$ 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 also are 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 $4 \times 4$ 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 A: Vector Fields

The frames package contains the `Vector2DFrame` class for displaying two-dimensional vector fields. To use this class we instantiate a multi-dimensional 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.

```java
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. The *Open Source Physics: A User's Guide with Examples* manual describes how a vector field visualization can be superimposed on a scalar field visualization.

## References and Suggestions for Further Reading

Forman S. Acton, *Numerical Methods That Work*, Harper & Row (1970); corrected edition, Mathematical Association of America (1990). Chapter 18 discusses solutions to Laplace's equation using the relaxation method and alternative approaches.

John W. Belcher and Stanislaw Olbert, "Field line motion in classical electromagnetism," Am. J. Phys. **71**, 220–228 (2003). The authors discuss some of the difficulties with field lines that change in time, and suggest a procedure where the local direction of motion of a field line is in the same direction as the Poynting vector.

Charles K. Birdsall and A. Bruce Langdon, *Plasma Physics via Computer Simulation*, McGraw-Hill (1985).

D. H. Choi and W. J. R. Hoefer, "The finite-difference-time-domain method and its application to eigenvalue problems," *IEEE Trans. Microwave Theory and Techniques* **34**, 1464–1469 (1986). The authors use Yee's algorithm to model microwave cavity resonators.

David M. Cook, *The Theory of the Electromagnetic Field*, Prentice Hall (1975). One of the first books to introduce numerical methods in the context of electromagnetism.

Robert Ehrlich, Jaroslaw Tuszynski, Lyle Roelofs, and Ronald Stoner, *Electricity and Magnetism Simulations: The Consortium for Upper-Level Physics Software*, John Wiley (1995).

Richard P. Feynman, Robert B. Leighton, and Matthew Sands, *The Feynman Lectures on Physics*, Vol. 2, Addison-Wesley (1963).

T. E. Freeman, "One-, two- or three-dimensional fields?," Am. J. Phys. **63**, 273–274 (1995).

R. L. Gibbs, Charles W. Beason, and James D. Beason, "Solutions to boundary value problems of the potential type by random walk method," Am. J. Phys. **43**, 782–785 (1975).

R. H. Good, "Dipole radiation: Simulation using a microcomputer," Am. J. Phys. **52**, 1150–1151 (1984). The author discusses a graphical simulation of dipole radiation.

David J. Griffiths, *Introduction to Electrodynamics*, third edition, Prentice Hall (1999). A classic undergraduate text on electromagnetism. See also David J. Griffiths and Daniel Z. Uvanovic, "The charge distribution on a conductor for non-coulombic potentials," Am. J. Phys. **69**, 435–440 (2001), O. F. de Alcantara Bonfim and David Griffiths, "Comment on 'Charge density on a thin straight wire, revisited,' by J. D. Jackson [Am. J. Phys. **68** (9), 789-799 (2000)]," Am. J. Phys. **69**, 515–516 (2001); and O. F. de Alcantara Bonfim, David J. Griffiths, and Sasha Hinkley, "Chaotic and hyperchaotic motion of a charged particle in a magnetic dipole field," Int. J. Bifurcation and Chaos**10**, 265–271 (2000).

R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles*, McGraw-Hill (1981).

Steven E. Koonin and Dawn C. Meredith, *Computational Physics*, Addison-Wesley (1990). See Chapter 6 for a discussion of the numerical solution of elliptic partial differential equations of which Laplace's and Poisson's equations are examples.

William M. MacDonald, "Discretization and truncation errors in a numerical solution of Laplace's equation," Am. J. Phys. **62**, 169–173 (1994).

William H. Press and Saul A. Teukolsky, "Multigrid methods for boundary value problems I," Computers in Physics **5** (5), 514 (1991).

Edward M. Purcell, *Electricity and Magnetism*, second edition, Berkeley Physics Course, Vol. 2, McGraw-Hill (1985). A well known text that discusses the relaxation method.

John R. Reitz, Frederick J. Milford, and Robert W. Christy, *Foundations of Electromagnetic Theory*, third edition, Addison-Wesley (1979). This text discusses microwave cavity resonators.

Matthew N. O. Sadiku, *Numerical Techniques in Electromagnetics*, second edition, CRC Press (2001).

A. Taflove and M. E. Brodwin, "Numerical solution of steady state electromagnetic scattering problems using the time dependent Maxwell equations," IEEE Trans. Microwave Theory and Techniques **23**, 623–630 (1975). The authors derive the stability conditions for the Yee algorithm.

P. B. Visscher, *Fields and Electrodynamics*, John Wiley & Sons (1988). An intermediate level text that incorporates computer simulations and analysis into its development.

P. J. Walker and I. D. Johnston, "Computer model clarifies spontaneous charge distribution in conductors," Computers in Physics **9**, 42 (1995).

Gregg Williams, "An introduction to relaxation methods," Byte **12** (1), 111–124 (1987). The author discusses the application of relaxation methods to the solution of the two-dimensional Poisson's equation.

K. S. Yee, "Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media," IEEE Trans. Antennas and Propagation **14**, 302–307 (1966). Yee uses the discretized Maxwell's equations to model the scattering of electromagnetic waves off a perfectly conducting rectangular obstacle.