```
      state[4] = 0;     // time
      Trail trail = new Trail();
      trail.color = Color.red;
      frame.addDrawable(trail);
      double r2 = state[0]*state[0]+state[2]*state[2];
      double count = 0;
      while((count<=1000)
           &&((2*r2)>state[0]*state[0]+state[2]*state[2])) {
         trail.addPoint(state[0], state[2]);
         odeSolver.step();
         count++;
      }
      return count<1000;
   }

   private double force(double r) {
      // Coulomb force law
      return(r==0) ? 0 : (1/r/r); // returns 0 if r = 0
   }

   public void getRate(double[] state, double[] rate) {
      double r = Math.sqrt(state[0]*state[0]+state[2]*state[2]);
      double f = force(r);
      rate[0] = state[1];
      rate[1] = (f*state[0])/r;
      rate[2] = state[3];
      rate[3] = (f*state[2])/r;
      rate[4] = 1;
   }

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

   double getAngle() {
      return Math.atan2(state[3], state[1])/Math.PI;
   }
}
```

The `ScatterAnalysis` class performs the data analysis. This class creates an array of `bins` to sort and accumulate the trajectories according to the scattering angle. The values of the scattering angle between 0° and 180° are divided into bins of width `dtheta`. To compute the number of particles coming from a ring of radius b, we accumulate the value of b associated with each bin or "detector" and write `bins[index] += b` (see the `detectParticle` method), because the number of particles in a ring of radius b is proportional to b. The total number of scattered particles is computed in the same way:

```
totalN += b;
```

You might want to increase the number of bins and the range of angles for better resolution.

Listing 5.8 The `ScatterAnalysis` class accumulates the scattering data and plots the differential cross section.

```
package org.opensourcephysics.sip.ch05;
import org.opensourcephysics.frames.PlotFrame;
```



```
public class ScatterAnalysis {
   int numBins = 20;
   PlotFrame frame = new PlotFrame("angle", "sigma",
                          "differential cross section");
   double[] bins = new double[numBins];
   double dtheta = Math.PI/(numBins-1);
   double totalN = 0; // total number of scattered particles

   void clear() {
      for(int i = 0;i<numBins;i++) {
         bins[i] = 0;
      }
      totalN = 0;
      frame.clearData();
      frame.repaint();
   }

   void detectParticle(double b, double theta) {
      // treats positive and negative angles equally to
      // get better statistics
      theta = Math.abs(theta);
      int index = (int) (theta/dtheta);
      bins[index] += b;
      totalN += b;
   }

   void plotCrossSection(double radius) {
      double targetDensity = 1/Math.PI/radius/radius;
      double delta = (dtheta*180)/Math.PI; // uses degrees for plot
      frame.clearData();
      for(int i = 0;i<numBins;i++) {
         double domega = 2*Math.PI*Math.sin((i+0.5)*dtheta)*dtheta;
         double sigma = bins[i]/totalN/targetDensity/domega;
         frame.append(0, (i+0.5)*delta, sigma);
      }
      frame.setVisible(true);
   }
}
```

Problem 5.12 Total cross section

The total cross section σ_T is defined as

$$\sigma_T = \int \sigma(\theta)\, d\Omega. \tag{5.29}$$

Add code to calculate and display the total cross section in the `plotCrossSection` method. Design a test to verify that the ODE solver in the `Scatter` class has sufficient accuracy. ■

In Problem 5.13, we consider a model of the hydrogen atom for which the force on a beam particle is zero for $r > a$. Because we do not count the beam particles that are not scattered, we set the beam radius equal to a. For forces that are not identically zero, we need to choose a minimum angle for θ such that particles whose scattering angle is less than this minimum are not counted as scattered (see Problem 5.14).