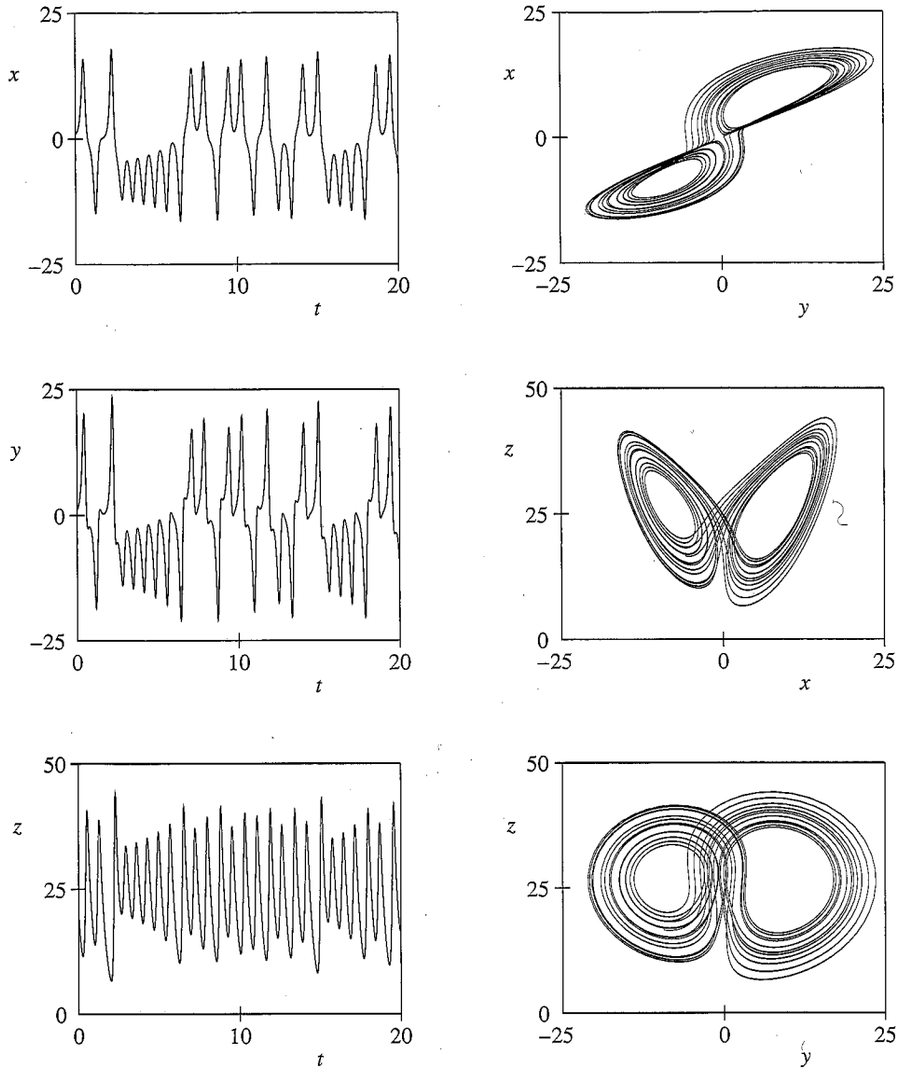

Figure 6.10 A trajectory of the Lorenz model with $\sigma = 10$, $b = 8/3$, and $r = 28$ and the initial condition $x_0 = 1$, $y_0 = 1$, $z_0 = 20$. A time interval of $t = 20$ is shown with points plotted at intervals of 0.01. The fourth-order Runge–Kutta algorithm was used with $\Delta t = 0.0025$.

Listing 6.5 `PoincareApp` plots a phase diagram and a Poincaré map for the damped driven pendulum.

```
package org.opensourcephysics.sip.ch06;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.frames.PlotFrame;
import org.opensourcephysics.numerics.RK4;

public class PoincareApp extends AbstractSimulation {
```



```
   final static double PI = Math.PI; // defined for brevity
   PlotFrame phaseSpace = new PlotFrame("theta", "angular velocity",
                                        "Phase space plot");
   PlotFrame poincare = new PlotFrame("theta", "angular velocity",
                                      "Poincare plot");
   int nstep = 100; // # iterations between Poincare plot
   DampedDrivenPendulum pendulum = new DampedDrivenPendulum();
   RK4 odeMethod = new RK4(pendulum);

   public PoincareApp() {
      // angular frequency of external force equals two and hence
      // period of external force equals pi
      odeMethod.setStepSize(PI/nstep); // dt = PI/nsteps
      phaseSpace.setMarkerShape(0, 6);
      // smaller size gives better resolution
      poincare.setMarkerSize(0, 2);
      poincare.setMarkerColor(0, java.awt.Color.RED);
      phaseSpace.setMessage("t = "+0);
   }

   public void reset() {
      control.setValue("theta", 0.2);
      control.setValue("angular velocity", 0.6);
      control.setValue("gamma", 0.2); // damping constant
      control.setValue("A", 0.85);    // amplitude
   }

   public void doStep() {
      double state[] = pendulum.getState();
      for(int istep = 0;istep<nstep;istep++) {
         odeMethod.step();
         if(state[0]>PI) {
            state[0] = state[0]-2.0*PI;
         } else if(state[0]<-PI) {
            state[0] = state[0]+2*PI;
         }
         phaseSpace.append(0, state[0], state[1]);
      }
      poincare.append(0, state[0], state[1]);
      phaseSpace.setMessage("t = "+decimalFormat.format(state[2]));
      poincare.setMessage("t = "+decimalFormat.format(state[2]));
      if(phaseSpace.isShowing()) {
         phaseSpace.render();
      }
      if(poincare.isShowing()) {
         poincare.render();
      }
   }

   public void initialize() {
      double theta = control.getDouble("theta");   // initial angle
      // initial angular velocity
      double omega = control.getDouble("angular velocity");
      pendulum.gamma = control.getDouble("gamma"); // damping constant
      // amplitude of external force
      pendulum.A = control.getDouble("A");
```