Chapter 5

Few-Body Problems: The Motion of the Planets

©2005 by Harvey Gould, Jan Tobochnik, and Wolfgang Christian
27 May 2005

We apply Newton's laws of motion to planetary motion and other systems of a few particles and explore some of the counter-intuitive consequences of Newton's laws.

5.1 Planetary Motion

Planetary motion is of special significance because it played an important role in the conceptual history of the mechanical view of the universe. Few theories have affected Western civilization as much as Newton's laws of motion and the law of gravitation, which together relate the motion of the heavens to the motion of terrestrial bodies.

Much of our knowledge of planetary motion is summarized by Kepler's three laws, which can be stated as:

1. Each planet moves in an elliptical orbit with the Sun located at one of the foci of the ellipse.
2. The speed of a planet increases as its distance from the Sun decreases such that the line from the Sun to the planet sweeps out equal areas in equal times.
3. The ratio T^2/a^3 is the same for all planets that orbit the Sun, where T is the period of the planet and a is the semimajor axis of the ellipse.

Kepler obtained these laws by a careful analysis of the observational data collected over many years by Tycho Brahe.

Kepler's first and third laws describe the shape of the orbit rather than the time dependence of the position and velocity of a planet. Because it is not possible to obtain this time dependence

in terms of elementary functions, we will obtain the numerical solution of the equations of motion of planets and satellites in orbit. In addition, we will consider the effects of perturbing forces on the orbit and problems that challenge our intuitive understanding of Newton's laws of motion.

5.2 The Equations of Motion

The motion of the Sun and Earth is an example of a *two-body problem.* We can reduce this problem to a one-body problem in one of two ways. The easiest way is to use the fact that the mass of the Sun is much greater than the mass of the Earth. Hence we can assume that, to a good approximation, the Sun is stationary and is a convenient choice of the origin of our coordinate system.

If you are familiar with the concept of a *reduced mass,* you know that the reduction to a one-body problem is more general. That is, the motion of two objects of mass m and M whose total potential energy is a function only of their relative separation can be reduced to an equivalent one-body problem for the motion of an object of reduced mass μ given by

$$\mu = \frac{Mm}{m+M}. \tag{5.1}$$

Because the mass of the Earth, $m = 5.99 \times 10^{24}\,\text{kg}$ is so much smaller than the mass of the Sun, $M = 1.99 \times 10^{30}$ kg, we find that for most practical purposes, the reduced mass of the Sun and the Earth is that of the Earth alone. In the following, we consider the problem of a single particle of mass m moving about a fixed center of force, which we take as the origin of the coordinate system.

Newton's universal law of gravitation states that a particle of mass M attracts another particle of mass m with a force given by

$$\mathbf{F} = -\frac{GMm}{r^2}\hat{\mathbf{r}} = -\frac{GMm}{r^3}\mathbf{r}, \tag{5.2}$$

where the vector $\mathbf{r}$ is directed from M to m (see Figure 5.1). The negative sign in (5.2) implies that the gravitational force is attractive, that is, it tends to decrease the separation r. The gravitational constant G is determined experimentally to be

$$G = 6.67 \times 10^{-11}\,\frac{\text{m}^3}{\text{kg}\cdot\text{s}^2}. \tag{5.3}$$

The force law (5.2) applies to the motion of the center of mass for objects of negligible spatial extent. Newton delayed publication of his law of gravitation for twenty years while he invented integral calculus and showed that (5.2) also applies to any uniform sphere or spherical shell of matter if the distance r is measured from the center of each mass.

The gravitational force has two general properties: its magnitude depends only on the separation of the particles, and its direction is along the line joining the particles. Such a force is called a central force. The assumption of a central force implies that the orbit of the Earth is restricted to a plane (x-y), and the angular momentum $\mathbf{L}$ is conserved and lies in the third (z) direction. We write L_z in the form

$$L_z = (\mathbf{r} \times m\mathbf{v})_z = m(xv_y - yv_x), \tag{5.4}$$

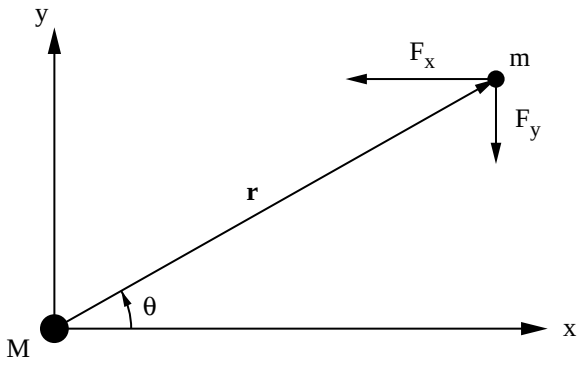

Figure 5.1: An object of mass m moves under the influence of a central force F. Note that $\cos\theta = x/r$ and $\sin\theta = y/r$, which provide useful relations for writing the equations of motion in component form suitable for numerical solutions.

where we have used the cross-product definition $\mathbf{L} = \mathbf{r} \times \mathbf{p}$ and $\mathbf{p} = m\mathbf{v}$. An additional constraint on the motion is that the total energy E is conserved and is given by

$$E = \frac{1}{2}mv^2 - \frac{GMm}{r}. \tag{5.5}$$

If we fix the coordinate system at the mass M, the equation of motion of the particle of mass m is

$$m\frac{d^2\mathbf{r}}{dt^2} = -\frac{GMm}{r^3}\mathbf{r}. \tag{5.6}$$

It is convenient to write the force in Cartesian coordinates (see Figure 5.1):

$$F_x = -\frac{GMm}{r^2}\cos\theta = -\frac{GMm}{r^3}x \tag{5.7a}$$

$$F_y = -\frac{GMm}{r^2}\sin\theta = -\frac{GMm}{r^3}y. \tag{5.7b}$$

Hence, the equations of motion in Cartesian coordinates are

$$\frac{d^2x}{dt^2} = -\frac{GM}{r^3}x \tag{5.8a}$$

$$\frac{d^2y}{dt^2} = -\frac{GM}{r^3}y, \tag{5.8b}$$

where $r^2 = x^2 + y^2$. Equations (5.8a) and (5.8b) are examples of coupled differential equations because each equation contains both x and y.

5.3 Circular and Elliptical Orbits

Because many planetary orbits are nearly circular, it is useful to obtain the condition for a circular orbit. The magnitude of the acceleration a is related to the radius r of the circular orbit by

$$a = \frac{v^2}{r}, \tag{5.9}$$

where v is the speed of the object. The acceleration always is directed toward the center and is due to the gravitational force. Hence we have

$$\frac{mv^2}{r} = \frac{GMm}{r^2}, \tag{5.10}$$

and

$$v = \left(\frac{GM}{r}\right)^{1/2}. \tag{5.11}$$

The relation (5.11) between the radius and the speed is the general condition for a circular orbit.

We also can find the dependence of the period T on the radius of a circular orbit using the relation,

$$T = \frac{2\pi r}{v}, \tag{5.12}$$

in combination with (5.11) to obtain

$$T^2 = \frac{4\pi^2}{GM} r^3. \tag{5.13}$$

The relation (5.13) is a special case of Kepler's third law with the radius r corresponding to the semimajor axis of an ellipse.

A simple geometrical characterization of an elliptical orbit is shown in Figure 5.2. The two *foci* of an ellipse, F_1 and F_2, have the property that for any point P, the distance $F_1P + F_2P$ is a constant. In general, an ellipse has two perpendicular axes of unequal length. The longer axis is the major axis; half of this axis is the semimajor axis a. The shorter axis is the minor axis; the semiminor axis b is half of this distance. It is common to specify an elliptical orbit by a and by the eccentricity e, where e is the ratio of the distance between the foci to the length of the major axis. Because $F_1P + F_2P = 2a$, it is easy to show that

$$e = \sqrt{1 - \frac{b^2}{a^2}} \tag{5.14}$$

with $0 < e < 1$. (Choose the point P at $x = 0, y = b$.) A special case is $b = a$ for which the ellipse reduces to a circle and $e = 0$.

5.4 Astronomical Units

It is convenient to choose a system of units in which the magnitude of the product GM is not too large and not too small. To describe the Earth's orbit, the convention is to choose the length of

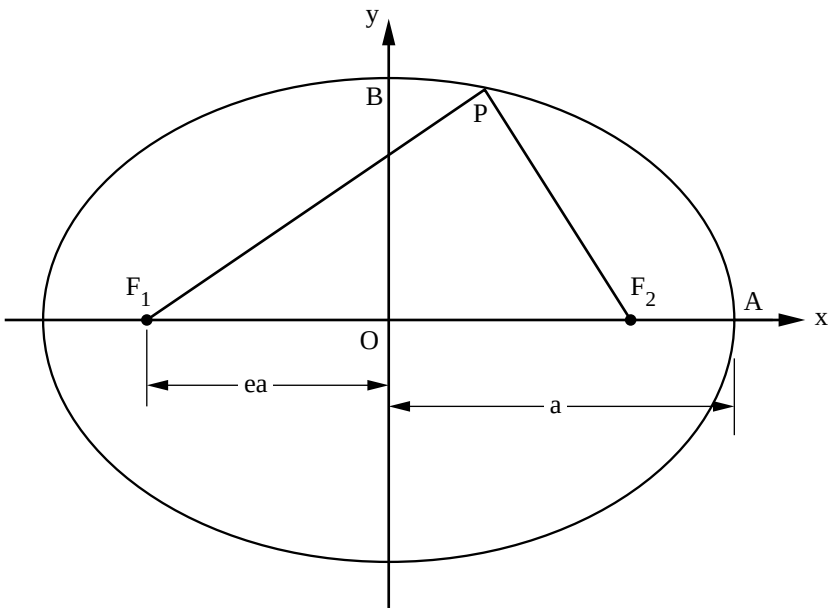

Figure 5.2: The characterization of an ellipse in terms of the semimajor axis a and the eccentricity e. The semiminor axis b is the distance OB. The origin O in Cartesian coordinates is at the center of the ellipse.

the Earth's semimajor axis as the unit of length. This unit of length is called the *astronomical unit* (AU) and is

$$1\,\text{AU} = 1.496 \times 10^{11}\,\text{m}. \tag{5.15}$$

The unit of time is assumed to be one year or 3.15×10^7 s. In these units, the period of the Earth is $T = 1$ years and its semimajor axis is $a = 1$ AU. Hence from (5.13)

$$GM = \frac{4\pi^2 a^3}{T^2} = 4\pi^2 \text{AU}^3/\text{years}^2. \qquad \text{(astronomical units)} \tag{5.16}$$

As an example of the use of astronomical units, a program distance of 1.5 would correspond to $1.5 \times (1.496 \times 10^{11}) = 2.244 \times 10^{11}$ m.

5.5 Log-log and Semilog Plots

The values of T and a for our solar system are given in Table 5.1. We first analyze these values and determine if T and a satisfy a simple mathematical relationship.

Suppose we wish to determine whether two variables y and x satisfy a functional relationship, $y = f(x)$. To simplify the analysis, we ignore possible errors in the measurements of y and x. The simplest relation between y and x is linear, that is, $y = mx + b$. The existence of such a relation can be seen by plotting y versus x and finding if the plot is linear. From Table 5.1 we see that T is not

a linear function of a. For example, an increase in T from 0.24 to 1, an increase of approximately 4, yields an increase in T of approximately 2.5.

For many problems, it is reasonable to assume an exponential relation

$$y = C\, e^{rx}, \tag{5.17}$$

or a power law relation

$$y = C\, x^n, \tag{5.18}$$

where C, r, and n are unknown parameters.

If we assume the exponential form (5.17), we can take the natural logarithm of both sides to find

$$\ln y = \ln C + rx. \tag{5.19}$$

Hence if (5.17) is applicable, a plot of $\ln y$ versus x would yield a straight line with slope r and intercept $\ln C$.

The natural logarithm of both sides of the power law relation (5.18) yields

$$\ln y = \ln C + n \ln x. \tag{5.20}$$

If (5.18) applies, a plot of $\ln y$ versus $\ln x$ yields the exponent n (the slope), which is the usual quantity of physical interest if a power law dependence holds.

planet	T (Earth years)	a (AU)
Mercury	0.241	0.387
Venus	0.615	0.723
Earth	1.0	1.0
Mars	1.88	1.523
Jupiter	11.86	5.202
Saturn	29.5	9.539
Uranus	84.0	19.18
Neptune	165	30.06
Pluto	248	39.44

Table 5.1: The period T and semimajor axis a of the planets. The unit of length is the astronomical unit (AU). The unit of time is one (Earth) year.

We illustrate a simple analysis of the data in Table 5.1. Because we expect that the relation between T and a has the power law form $T = Ca^n$, we plot $\ln T$ versus $\ln a$ (see Figure 5.3). A visual inspection of the plot indicates that a linear relationship between $\ln T$ and $\ln a$ is reasonable and that the slope is approximately 1.50 in agreement with Kepler's second law. In Chapter 8, we will discuss the least squares method for fitting a straight line through a number of data points. With a little practice you can do a visual analysis that is nearly as good.

The `PlotFrame` class contains the axes and titles needed to produce linear, log-log, and semi-log plots. It also contains the methods needed to display data in a table format. This table can be displayed programmatically or by right-clicking (control-clicking) at runtime. Listing 5.1 shows

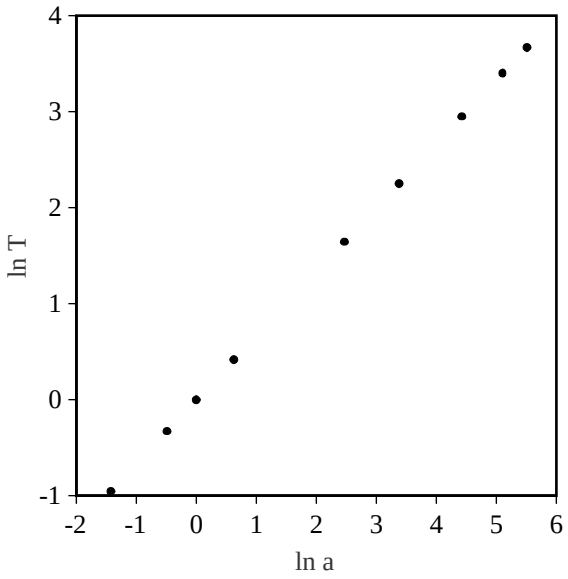

Figure 5.3: Plot of $\ln T$ versus $\ln a$ using the data in Table 5.1. Verify that the slope is 1.50.

a short program that produces the log-log plot of the semi-major axis of the planets versus the orbital period. The arrays, `a` and `T`, contain the semimajor axis of the planets and their periods, respectively. Setting the log scale option causes the `PlotFrame` to transform the data as it is being plotted and causes the axis to change how labels are rendered. Note that the plot automatically adjusts itself to fit the data because the autoscale option is true by default. Also the grid and the tick-labels change as the window is resized.

Listing 5.1: A simple program that producs a log-log plot to demonstrate Kepler's second law.

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

public class SecondLawPlotApp {
   public static void main(String[] args) {
      PlotFrame frame = new PlotFrame("ln(a)", "ln(T)", "Kepler's second law");
      frame.setLogScale(true, true);
      frame.setConnected(false);
      double[] period = {
         0.241, 0.615, 1.0, 1.88, 11.86, 29.50, 84.0, 165, 248
      };
      double[] a = {
         0.387, 0.723, 1.0, 1.523, 5.202, 9.539, 19.18, 30.06, 39.44
      };
      frame.append(0, a, period);
      frame.setVisible(true);
      // defines titles of table columns
      frame.setXYColumnNames(0, "T (years)", "a (AU)");
```

x	$y_1(x)$	$y_2(x)$	$y_3(x)$
0	0.00	0.00	2.00
0.5	0.75	1.59	5.44
1.0	3.00	2.00	14.78
1.5	6.75	2.29	40.17
2.0	12.00	2.52	109.20
2.5	18.75	2.71	296.83

Table 5.2: Determine the functional forms of $y(x)$ for the three sets of data. There are no measurement errors, but there are roundoff errors.

```
        // shows data table; can also be done from frame menu
        frame.showDataTable(true);
        frame.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
    }
}
```

Exercise 5.1. Simple functional forms

a. Run `SecondLawPlotApp` and convince yourself that you understand the syntax.

b. Modify `SecondLawPlotApp` so that the three sets of data shown in Table 5.2 are plotted. Generate linear, semi-log, and log-log plots to determine the functional form of $y(x)$ that best fits each data set.

5.6 Simulation of the Orbit

We now develop a program to simulate the Earth's orbit about the Sun. The `PlanetApp` class shown in Listing 5.2 organizes the startup process and creates the visualization. Because this class extends `AbstractSimulation`, it is sufficient to know that the superclass invokes the `doStep` method periodically when the thread is running or once each time the Step button is clicked. The preferred scale and the aspect ratio for the plot frame is set in the constructor. The statement `frame.setSquareAspect(true)` ensures that a unit of distance will equal the same number of pixels in both the horizontal and vertical directions; the statement `planet.initialize(new double[]{x, vx, y, vy, 0})` in the `initialize` method is used to create an array on the fly as the argument to another method.

Listing 5.2: PlanetApp.

```
package org.opensourcephysics.sip.ch05;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.frames.*;

public class PlanetApp extends AbstractSimulation {
   PlotFrame frame = new PlotFrame("x (AU)", "y (AU)", "Planet Simulation");
```

```
   Planet planet = new Planet();

   public PlanetApp() {
      frame.addDrawable(planet);
      frame.setPreferredMinMax(-5, 5, -5, 5);
      frame.setSquareAspect(true);
   }

   public void doStep() {
      for(int i = 0;i<5;i++) { // do 5 steps between screen draws
         planet.doStep();      // advances time
      }
      frame.setMessage("t = "+decimalFormat.format(planet.state[4]));
   }

   public void initialize() {
      planet.odeSolver.setStepSize(control.getDouble("dt"));
      double x = control.getDouble("x");
      double vx = control.getDouble("vx");
      double y = control.getDouble("y");
      double vy = control.getDouble("vy");
      // create an array on the fly as the argument to another method
      planet.initialize(new double[] {x, vx, y, vy, 0});
      frame.setMessage("t = 0");
   }

   public void reset() {
      control.setValue("x", 1);
      control.setValue("vx", 0);
      control.setValue("y", 0);
      control.setValue("vy", 6.28);
      control.setValue("dt", 0.01);
      initialize();
   }

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

The `Planet` class in Listing 5.3 defines the physics and instantiates the numerical method. The latter is the Euler algorithm, which will be replaced in Problem 5.2. Note how the argument to the `initialize` method is used. The `System.arraycopy(array1,index1,array2,index2,length` method in the core Java API copies blocks of memory such as arrays and is optimized for particular operating systems. This method copies `length` elements of `array1` staring at `index1` into `array2` starting at `index2`. In most applications `index1` and `index2` will be set equal to 0.

Listing 5.3: A class that models the rate equation for a planet acted on by an inverse square law force.

```
package org.opensourcephysics.sip.ch05;
```

```
import java.awt.*;
import org.opensourcephysics.display.*;
import org.opensourcephysics.numerics.*;

public class Planet implements Drawable, ODE {
   // GM in units of (AU)^3/(yr)^2
   final static double GM = 4*Math.PI*Math.PI;
   Circle circle = new Circle();
   Trail trail = new Trail();
   double[] state = new double[5];     // {x,vx,y,vy,t}
   Euler odeSolver = new Euler(this); // creates numerical method

   public void doStep() {
      odeSolver.step();                    // advances time
      trail.addPoint(state[0], state[2]); // x,y
   }

   void initialize(double[] initState) {
      System.arraycopy(initState, 0, state, 0, initState.length);
      // reinitializes the solver in case the solver accesses data from previous steps
      odeSolver.initialize(odeSolver.getStepSize());
      trail.clear();
   }

   public void getRate(double[] state, double[] rate) {
      // state[]: x, vx, y, vy, t
      double r2 = (state[0]*state[0])+(state[2]*state[2]); // r squared
      double r3 = r2*Math.sqrt(r2);                       // r cubed
      rate[0] = state[1];          // x rate
      rate[1] = (-GM*state[0])/r3; // vx rate
      rate[2] = state[3];          // y rate
      rate[3] = (-GM*state[2])/r3; // vy rate
      rate[4] = 1;                 // time rate
   }

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

   public void draw(DrawingPanel panel, Graphics g) {
      circle.setXY(state[0], state[2]);
      circle.draw(panel, g);
      trail.draw(panel, g);
   }
}
```

The `Planet` class implements the `Drawable` interface and defines the `draw` method as described in Section 3.3. In this case we did not use graphics primitives such as `fillOval` to perform the drawing. Instead, the method calls the methods `circle.draw` and `trail.draw` to draw the planet and its trajectory, respectively.

Invoking a method in another object that has the desired functionality is known as *forwarding* or *delegating* the method. One advantage of forwarding is that we can change the implementation of the drawing within the `Planet` class at any time and still be assured that the `planet` object is drawable. We could, for example, replace the circle by an image of the Earth. Note that we have created a composite object by combining the properties of the simpler `circle` and `trace` objects. These techniques of encapsulation and composition are common in object oriented programming.

Problem 5.2. Verification of `Planet` and `PlanetApp` for circular orbits

a. Verify `Planet` and `PlanetApp` by considering the special case of a circular orbit. For example, choose (in astronomical units) $x(t=0)=1$, $y(t=0)=0$, and $v_x(t=0)=0$. Use the relation (5.11) to find the value of $v_y(t=0)$ that yields a circular orbit. How small a value of Δt is needed so that a circular orbit is repeated over many periods? Your answer will depend on your choice of differential equation solver. Find the largest value of Δt that yields an orbit that repeats for many revolutions using the Euler, Euler-Cromer, Verlet, and RK4 algorithms. Is it possible to choose a smaller value of Δt, or are some algorithms, such as the Euler method, simply not stable for this dynamical system?

b. Write a method to compute the total energy (see (5.5)) and compute it at regular intervals as the system evolves. (It is sufficient to calculate the energy per unit mass, E/m.) For a given value of Δt, which algorithm conserves the total energy best? Is it possible to choose a value of Δt that conserves the energy exactly? What is the significance of the negative sign for the total energy?

c. Write a separate method to determine the numerical value of the period. (See Problem 3.9c for a discussion of a similar condition.) Choose different sets of values of $x(t=0)$ and $v_y(t=0)$, consistent with the condition for a circular orbit. For each orbit, determine the radius and the period and verify Kepler's third law.

Problem 5.3. Verification of Kepler's second and third law

a. Set $y(t=0)=0$ and $v_x(t=0)=0$ and find by trial and error several values of $x(t=0)$ and $v_y(t=0)$ that yield elliptical orbits of a convenient size. Choose a suitable algorithm and plot the speed of the planet as the orbit evolves. Where is the speed a maximum (minimum)?

b. Use the same initial conditions as in part (a) and compute the total energy, angular momentum, semimajor and semiminor axes, eccentricity, and period for each orbit. Plot your data for the dependence of the period T on the semimajor axis a and verify Kepler's third law. Given the ratio of T^2/a^3 that you found, determine the numerical value of this ratio in SI units for our solar system.

c. The force center is at $(x,y)=(0,0)$ and is one focus. Find the second focus by symmetry. Compute the sum of the distances from each point on the orbit to the two foci and verify that the orbit is an ellipse.

d. According to Kepler's second law, the orbiting object sweeps out equal areas in equal times. If we use an algorithm with a fixed time step Δt, it is sufficient to compute the area of the triangle

swept in each time step. This area equals one-half the base of the triangle times its height, or $\frac{1}{2}\Delta t\,(\mathbf{r}\times\mathbf{v}) = \frac{1}{2}\Delta t(xv_y - yv_x)$. Is this area a constant? This constant corresponds to what physical quantity?

e.* Show that algorithms with a fixed value of Δt break down if the "planet" is too close to the sun. What is the cause of the failure of the method? What advantage might there be to using a variable time step? What are the possible disadvantages? (See Project 5.19 for an example where a variable time step is very useful.)

Problem 5.4. Non-inverse square forces

a. Consider the dynamical effects of a small change in the attractive inverse-square force law, for example, let the magnitude of the force equal $Cm/r^{2+\delta}$, where $\delta << 1$. For simplicity, take the numerical value of the constant C to be $4\pi^2$ as before. Consider the initial conditions $x(t=0)=1$, $y(t=0)=0$, $v_x(t=0)=0$, and $v_y(t=0)=5$. Choose $\delta = 0.05$ and determine the nature of the orbit. Does the orbit of the planet retrace itself? Verify that your result is not due to your choice of Δt. Does the planet spiral away from or toward the sun? The path of the planet can be described as an elliptical orbit that slowly rotates or *precesses* in the same sense as the motion of the planet. A convenient measure of the precession is the angle between successive orientations of the semimajor axis of the ellipse. This angle is the rate of precession per revolution. Estimate the magnitude of this angle for your choice of δ. What is the effect of decreasing the semimajor axis for fixed δ? What is the effect of changing δ for fixed semimajor axis?

b. Einstein's theory of gravitation (the general theory of relativity) predicts a correction to the force on a planet that varies as $1/r^4$ due to a weak gravitational field. The result is that the equation of motion for the trajectory of a particle can be written as

$$\frac{d^2\mathbf{r}}{dt^2} = -\frac{GM}{r^2}\left[1+\alpha\Big(\frac{GM}{c^2}\Big)^2\frac{1}{r^2}\right]\hat{\mathbf{r}}, \tag{5.21}$$

where the parameter α is dimensionless. Take $GM = 4\pi^2$ and assume $\alpha = 10^{-3}$. Determine the nature of the orbit for this potential. (For our solar system the constant α is a maximum for the planet Mercury, but is much smaller than 10^{-3}.)

c. Suppose that the attractive gravitational force law depends on the inverse-cube of the distance, Cm/r^3. What are the units of C? For simplicity, take the numerical value of C to be $4\pi^2$. Consider the initial condition $x(t=0)=1$, $y(t=0)=0$, $v_x(t=0)=0$, and determine analytically the value of $v_y(t=0)$ required for a circular orbit. How small a value of Δt is needed so that the simulation yields a circular orbit over several periods? How does this value of Δt compare with the value needed for the inverse-square force law?

d. Vary $v_y(t=0)$ by approximately 2% from the circular orbit condition that you determined in part (c). What is the nature of the new orbit? What is the sign of the total energy? Is the orbit bound? Is it closed? Are all bound orbits closed?

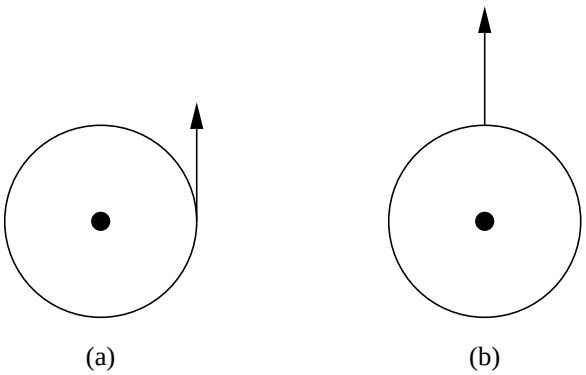

Figure 5.4: (a) An impulse applied in the tangential direction. (b) An impulse applied in the radial direction.

Problem 5.5. Effect of drag resistance on a satellite orbit

Consider a satellite in orbit about the Earth. In this case it is convenient to measure distances in terms of the radius of the Earth, $R = 6.37 \times 10^6$ m, and the time in terms of hours. Because the force on the satellite is proportional to Gm, where $m = 5.99 \times 10^{24}$ kg is the mass of the Earth, we need to evaluate the product Gm in Earth units (EU). In these units the value of Gm is given by

$$\begin{aligned} Gm &= 6.67 \times 10^{-11} \frac{\mathrm{m}^3}{\mathrm{kg} \cdot \mathrm{s}^2} \left(\frac{1\,\mathrm{EU}}{6.37 \times 10^6\,\mathrm{m}} \right)^3 \left(3.6 \times 10^3\,\mathrm{s/h} \right)^2 \left(5.99 \times 10^{24}\,\mathrm{kg} \right) \\ &= 20.0\,\mathrm{EU}^3/\mathrm{h}^2. \qquad \text{(Earth units)} \end{aligned} \tag{5.22}$$

Modify the `Planet` class to incorporate the effects of drag resistance on the motion of an orbiting Earth satellite. Assume that the drag force is proportional to the square of the speed of the satellite. To be able to observe the effects of air resistance in a reasonable time, take the magnitude of the drag force to be approximately one-tenth of the magnitude of the gravitational force. Choose initial conditions such that a circular orbit would be obtained in the absence of drag resistance and allow at least one revolution before "switching on" the drag resistance. Describe the qualitative change of the orbit due to drag resistance. How does the total energy and the speed of the satellite change with time?

5.7 Impulsive Forces

What happens to the orbit of an Earth satellite when it is hit by space debris? We now discuss the modifications we need to make in `Planet` and `PlanetApp` so that we can apply an impulsive force (a kick) by a mouse click. If we apply a vertical kick when the position of the satellite is as shown in Figure 5.4a, the impulse would be tangential to the orbit. A radial kick can be applied when the satellite is as shown in Figure 5.4b.

User actions, such as mouse clicks or keyboard entries, are passed from the operating system to Java *event listeners*. Although this standard Java framework is straightforward, we have simplified

it to respond to mouse actions within the Open Source Physics panels and frames.[1] In order for an Open Source Physics program to respond to mouse actions, the program implements the `InteractiveMouseHandler` interface and then registers its ability to process mouse actions with the `PlotFrame`. This procedure is demonstrated in the following test program. You can copy the `handleMouseAction` code into your program and replace the print statements with useful methods. Other mouse actions, such as `MOUSE_CLICKED`, `MOUSE_MOVED`, and `MOUSE_ENTERED` are defined in the `InteractivePanel` class.

Listing 5.4: A test program that implements the `InteractiveMouseHandler` interface.

```
package org.opensourcephysics.sip.ch05;
import java.awt.event.*;
import javax.swing.*;
import org.opensourcephysics.display.*;
import org.opensourcephysics.frames.*;

public class MouseApp implements InteractiveMouseHandler {
   PlotFrame frame = new PlotFrame("x", "y", "Interactive Handler");

   public MouseApp() {
      frame.setInteractiveMouseHandler(this);
      frame.setVisible(true);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }

   public void handleMouseAction(InteractivePanel panel, MouseEvent evt) {
      switch(panel.getMouseAction()) {
      case InteractivePanel.MOUSE_DRAGGED :
         panel.setMessage("Dragged");
         break;
      case InteractivePanel.MOUSE_PRESSED :
         panel.setMessage("Pressed");
         break;
      case InteractivePanel.MOUSE_RELEASED :
         panel.setMessage(null);
         break;
      }
   }

   public static void main(String[] args) {
      new MouseApp();
   }
}
```

The `switch` statement is used in Listing 5.4 instead of a chain of `if` statements. The panel's `getMouseAction` method returns an integer. If this integer matches one of the named constants following the `case` label, then the statements following that constant are executed until a `break` statement is encountered. If a `case` does not include a `break`, then the execution continues with

[1] See the *Open Source Physics User's Guide* for an extensive discussion of interactive drawing panels.

the next `case`. The equivalent of the `else` construct in an `if` statement is `default` followed by statements that are executed if none of the explicit cases occur.

We now challenge your intuitive understanding of Newton's laws of motion by considering several perturbations of the motion of an orbiting object. Modify your planet program to simulate the effects of the perturbations in Problem 5.6. In each case answer the questions before doing the simulation.

Problem 5.6. Tangential and radial perturbations

a. Suppose that a small tangential "kick" or impulsive force is applied to a satellite in a circular orbit about the Earth (see Figure 5.4a.) Choose Earth units so that the numerical value of the product Gm is given by (5.22). Apply the impulsive force by stopping the program after the satellite has made several revolutions and click the mouse to apply the force. Recall that the impulse changes the momentum in the desired direction directly. In what direction does the orbit change? Is the orbit stable, for example, does a small impulse lead to a small change in the orbit? Does the orbit retrace itself indefinitely if no further perturbations are applied? Describe the shape of the perturbed orbit.

b. How does the change in the orbit depend on the strength of the kick and its duration?

c. Determine if the angular momentum and the total energy are changed by the perturbation.

d. Apply a radial kick to the satellite as in Figure 5.4b and answer the same questions as in parts (a)–(c).

e. Determine the stability of the inverse-cube force law (see Problem 5.4) to radial and tangential perturbations.

Mouse actions are not the only possible way to affect the simulation. We also can add *custom buttons* to the control. These buttons are added when the program is instantiated in the `main` method.

```
public static void main(String[] args) {
   // OSPControl is a superclass of SimulationControl
   OSPControl control = SimulationControl.createApp(new PlanetApp());
   control.addButton("doRadialKick", "Kick!", "Perform a radial kick");
}
```

Note that `SimulationControl` (and `CalculationControl`) extend the `OSPControl` superclass and therefore support the `addButton` method where this method is defined. We assign the variable returned by the static `createApp` method to a variable of type `OSPControl` to highlight the object oriented structure of the Open Source Physics library.

The first parameter in the `addButton` method specifies the method that will be invoked when the button is clicked, the second parameter specifies the text label that will appear on the button, and the third parameter specifies the *tool tip* that will appear when the mouse hovers over the button. Custom buttons can be used for just about anything, but the corresponding method must be defined.

Exercise 5.7. Custom buttons

Use a custom button in Problem 5.6 rather than a mouse click to apply an impulsive force to the planet.

5.8 Velocity Space

In Problem 5.6 your intuition might have been incorrect. For example, you might have thought that the orbit would elongate in the direction of the kick. In fact the orbit does elongate, but in a direction perpendicular to the kick. Do not worry, you are in good company! Few students have a good qualitative understanding of Newton's law of motion, even after taking an introductory course in physics. A qualitative way of stating Newton's second law is

Forces act on the trajectories of particles by changing velocity not position.

If we fail to take into account this property of Newton's second law, we will encounter physical situations that appear counterintuitive.

Because force acts to change velocity, it is reasonable to consider both velocity and position on an equal basis. In fact position and momentum are treated in such a manner in advanced formulations of classical mechanics and in quantum mechanics.

In Problem 5.8 we explore some of the properties of orbits in velocity space in the context of the bound motion of a particle in an inverse-square force. Modify your program so that the path in velocity space of the Earth is plotted. That is, plot the point (v_x, v_y) the same way you plotted the point (x, y). The path in velocity space is a series of successive values of the object's velocity vector. If the position space orbit is an ellipse, what is the shape of the orbit in velocity space?

Problem 5.8. Properties of velocity space orbits

a. Modify your program to display the orbit in position space and in velocity space at the same time. Verify that the velocity space orbit is a circle, even if the orbit in position space is an ellipse. Does the center of this circle coincide with the origin $(v_x, v_y) = (0, 0)$ in velocity space? Choose the same initial conditions that you considered in Problems 5.2 and 5.3.

b.* Let $\mathbf{u}$ denote the radius vector of a point on the velocity circle, and $\mathbf{w}$ denote the vector from the origin in velocity space to the center of the velocity circle (see Figure 5.5). Then the velocity of the particle can be written as

$$\mathbf{v} = \mathbf{u} + \mathbf{w}. \tag{5.23}$$

Compute $\mathbf{u}$ and verify that its magnitude is given by

$$u = GMm/L, \tag{5.24}$$

where L is the magnitude of the angular momentum. Note that L is proportional to m so that it is not necessary to know the magnitude of m.

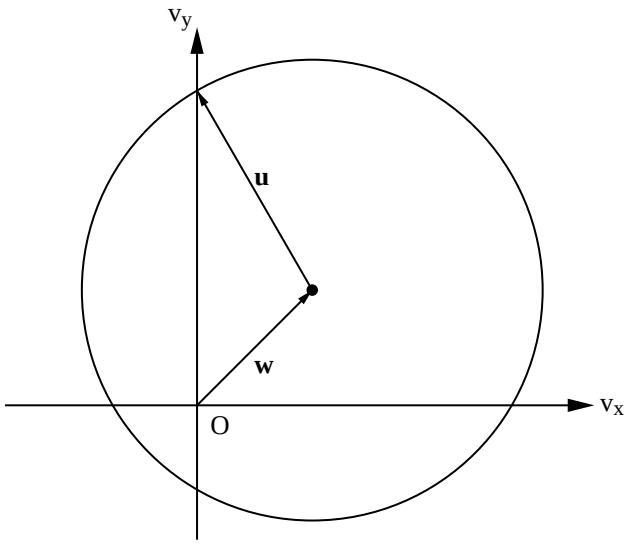

Figure 5.5: The orbit of a particle in velocity space. The vector **w** points from the origin in velocity space to the center of the circular orbit. The vector **u** points from the center of the orbit to the point (v_x, v_y).

c.* Verify that at each moment in time, the planet's position vector **r** is perpendicular to **u**. Explain why this relation holds.

Problem 5.9. Effect of impulses in velocity space

How does the velocity space orbit change when an impulsive kick is applied in the tangential or in the radial direction? How does the magnitude and direction of **w** change? From the observed change in the velocity orbit and the above considerations, explain the observed change of the orbit in position space.

5.9 A Mini-Solar System

So far our study of planetary orbits has been restricted to two-body central forces. However, the solar system is not a two-body system, because the planets exert gravitational forces on one another. Although the interplanetary forces are small in magnitude in comparison to the gravitational force of the Sun, they can produce measurable effects. For example, the existence of Neptune was conjectured on the basis of a discrepancy between the experimentally measured orbit of Uranus and the predicted orbit calculated from the known forces.

The presence of other planets implies that the total force on a given planet is not a central force. Furthermore, because the orbits of the planets are not exactly in the same plane, an analysis of the solar system must be extended to three dimensions if accurate calculations are required. However

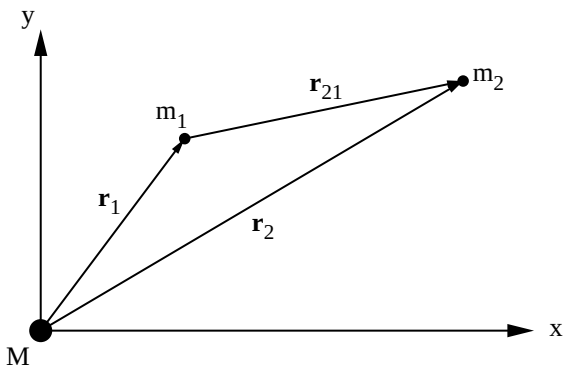

Figure 5.6: The coordinate system used in (5.25). Planets of mass m_1 and m_2 orbit a sun of mass M.

for simplicity, we will consider a model of a two-dimensional solar system with two planets in orbit about a fixed sun.

The equations of motion of two planets of mass m_1 and mass m_2 can be written in vector form as (see Figure 5.6)

$$m_1 \frac{d^2\mathbf{r}_1}{dt^2} = -\frac{GMm_1}{{r_1}^3}\mathbf{r}_1 + \frac{Gm_1m_2}{{r_{21}}^3}\mathbf{r}_{21} \tag{5.25a}$$

$$m_2 \frac{d^2\mathbf{r}_2}{dt^2} = -\frac{GMm_2}{{r_2}^3}\mathbf{r}_2 - \frac{Gm_1m_2}{{r_{21}}^3}\mathbf{r}_{21}, \tag{5.25b}$$

where $\mathbf{r}_1$ and $\mathbf{r}_2$ are directed from the sun to planets 1 and 2 respectively, and $\mathbf{r}_{21} = \mathbf{r}_2 - \mathbf{r}_1$ is the vector from planet 1 to planet 2. It is convenient to divide (5.25a) by m_1 and (5.25b) by m_2 and to write the equations of motion as

$$\frac{d^2\mathbf{r}_1}{dt^2} = -\frac{GM}{{r_1}^3}\mathbf{r}_1 + \frac{Gm_2}{{r_{21}}^3}\mathbf{r}_{21} \tag{5.26a}$$

$$\frac{d^2\mathbf{r}_2}{dt^2} = -\frac{GM}{{r_2}^3}\mathbf{r}_2 - \frac{Gm_1}{{r_{21}}^3}\mathbf{r}_{21}. \tag{5.26b}$$

A numerical solution of (5.26) can be obtained by the straightforward extension of the `Planet` class as shown in Listing 5.5. To simplify the drawing of the particle trajectories, the `Planet2` class defines an *inner class*, `Mass`, which extends `Circle` and contains a `Trail`. Whenever a planet moves, a point is added to the trail so that its location and path are shown on the plot. Inner classes are an organizational convenience that save us the trouble of having to create another file, which in this case would be named `Mass.java`. When we compile the `Planet2` class, we will produce a bytecode file named `Planet2$Mass.class` in addition to the file `Planet2.class`. Inner classes are most effective as short helper classes which work in conjuction with the containing class because they have access to all the data (including private variables) in the containing class.

Listing 5.5: A class that implements the rate equation for two interacting planets acted on by an inverse square law force.

```
package org.opensourcephysics.sip.ch05;
import java.awt.*;
import org.opensourcephysics.display.*;
import org.opensourcephysics.numerics.*;

public class Planet2 implements Drawable, ODE {
   // GM in units of (AU)^3/(yr)^2
   final static double GM = 4*Math.PI*Math.PI;
   final static double GM1 = 0.04*GM;
   final static double GM2 = 0.001*GM;
   double[] state = new double[9];
   ODESolver odeSolver = new RK45MultiStep(this);
   Mass mass1 = new Mass(), mass2 = new Mass();

   public void doStep() {
      odeSolver.step();
      mass1.setXY(state[0], state[2]);
      mass2.setXY(state[4], state[6]);
   }

   public void draw(DrawingPanel panel, Graphics g) {
      mass1.draw(panel, g);
      mass2.draw(panel, g);
   }

   void initialize(double[] initState) {
      System.arraycopy(initState, 0, state, 0, initState.length);
      mass1.clear(); // clears data from the old trail
      mass2.clear();
      mass1.setXY(state[0], state[2]);
      mass2.setXY(state[4], state[6]);
   }

   public void getRate(double[] state, double[] rate) {
      // state[]: x1, vx1, y1, vy1, x2, vx2, y2, vy2, t
      double r1Squared = (state[0]*state[0])+(state[2]*state[2]); // r1 squared
      double r1Cubed = r1Squared*Math.sqrt(r1Squared);           // r1 cubed
      double r2Squared = (state[4]*state[4])+(state[6]*state[6]); // r2 squared
      double r2Cubed = r2Squared*Math.sqrt(r2Squared);           // r2 cubed
      double dx = state[4]-state[0];                             // x12 separation
      double dy = state[6]-state[2];                             // y12 separation
      double dr2 = (dx*dx)+(dy*dy);                              // r12 squared
      double dr3 = Math.sqrt(dr2)*dr2;                           // r12 cubed
      rate[0] = state[1];                                  // x1 rate
      rate[2] = state[3];                                  // y1 rate
      rate[4] = state[5];                                  // x2 rate
      rate[6] = state[7];                                  // y2 rate
```

```
      rate[1] = ((-GM*state[0])/r1Cubed)+((GM1*dx)/dr3); // vx1 rate
      rate[3] = ((-GM*state[2])/r1Cubed)+((GM1*dy)/dr3); // vy1 rate
      rate[5] = ((-GM*state[4])/r2Cubed)-((GM2*dx)/dr3); // vx2 rate
      rate[7] = ((-GM*state[6])/r2Cubed)-((GM2*dy)/dr3); // vy2 rate
      rate[8] = 1;                                       // time rate
   }

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

   class Mass extends Circle {
      Trail trail = new Trail();

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

      void clear() {
         trail.clear();
      }

      public void setXY(double x, double y) {
         super.setXY(x, y);
         trail.addPoint(x, y);
      }
   }
}
```

The target application, `Planet2App`, extends `AbstractSimulation` in the usual way. Because it is almost identical to Listing 5.2, it is not shown here. The complete program is available in the ch05 package.

Problem 5.10. Planetary perturbations

Use `Planet2App` with the initial conditions given in the program. For illustrative purposes, we have adopted the numerial values $m_1/M = 10^{-3}$ and $m_2/M = 4\times10^{-2}$ and hence `GM1` $= (m_2/M)GM =$ 0.04`GM` and `GM2` $= (m_1/M)GM =$ 0.001`GM`. What would be the shape of the orbits and the periods of the two planets if they did not mutually interact? What is the qualitative effect of their mutual interaction? Describe the shape of the two orbits. Why is one planet affected more by their mutual interaction than the other? Is the angular momentum and the total energy of planet one conserved? Is the total energy and total angular momentum of the two planets conserved? A related, but more time consuming problem is given in Project 5.18.

Problem 5.11. Double stars

Another interesting dynamical system consists of one planet orbiting about two fixed stars of equal mass. In this case there are no closed orbits, but the orbits can be classified as either stable or unstable. Stable orbits may be open loops that encircle both stars, figure eights, or orbits that

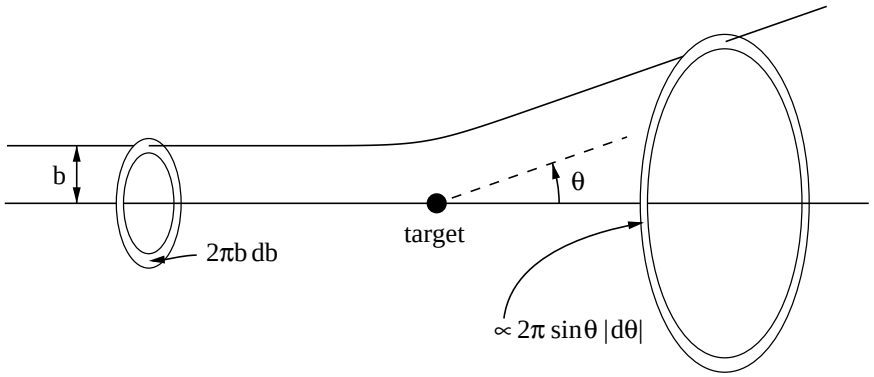

Figure 5.7: The coordinate system used to define the differential scattering cross section. Particles passing through the beam area $2\pi b\,db$ are scattered into the solid angle $d\Omega$.

encircle only one star. Unstable orbits will eventually collide with one of the stars. Modify `Planet2` to simulate the double star system, with the first star located at $(-1, 0)$ and the second star of equal mass located at $(1, 0)$. Place the planet at $(0.1, 1)$ and systematically vary the x and y components of the velocity to obtain different types of orbits. Then try other initial positions.

5.10 Two-Body Scattering

Much of our understanding of the structure of matter comes from scattering experiments. In this section we explore one of the more difficult concepts in the theory of scattering, the *differential cross section*.

A typical scattering experiment involves a beam with many incident particles all with the same kinetic energy. The coordinate system is shown in Figure 5.7. The incident particles come from the left with an initial velocity $\mathbf{v}$ in the $+x$ direction. We take the center of the beam and the center of the target to be on the x axis. The *impact parameter* b is the perpendicular distance from the initial trajectory to a parallel line through the center of the target (see Figure 5.7). We assume that the width of the beam is larger than the size of the target. The target contains many scattering centers, but for calculational purposes we may consider scattering off only one particle if the target is sufficiently thin.

When an incident particle comes close to the target, it is deflected. In a typical experiment, the scattered particles are counted in a detector that is far from the target. The final velocity of the scattered particles is $\mathbf{v}'$, and the angle between $\mathbf{v}$ and $\mathbf{v}'$ is the scattering angle θ.

Let us assume that the scattering is elastic and that the target is much more massive than the beam particles so that the target can be considered to be fixed. (The latter condition can be relaxed by using center of mass coordinates.) We also assume that no incident particle is scattered more than once. These considerations imply that the initial speed and final speed of the incident particles are equal. The functional dependence of θ on b depends on the force on the beam particles due to the target. In a typical experiment the number of particles in an angular region between θ and $\theta + d\theta$ is detected for many values of θ. These detectors measure the number of particles

scattered into the solid angle $d\Omega = \sin\theta d\theta d\phi$ centered about θ. The *differential cross section* $\sigma(\theta)$ is defined by the relation

$$\frac{dN}{N} = n\sigma(\theta)d\Omega, \tag{5.27}$$

where dN is the number of particles scattered into the solid angle $d\Omega$ centered about θ and the azimuthal angle ϕ, N is the total number of particles in the beam, and n is the target density defined as the number of targets per unit area.

The interpretation of (5.27) is that the fraction of particles scattered into the solid angle $d\Omega$ is proportional to $d\Omega$ and the density of the target. From (5.27) we see that $\sigma(\theta)$ can be interpreted as the effective area of a target particle for the scattering of an incident particle into the element of solid angle $d\Omega$. Particles that are not scattered are ignored. Another way of thinking about $\sigma(\theta)$ is that it is the ratio of the area $b\,db\,d\phi$ to the solid angle $d\Omega = \sin\theta d\theta\, d\phi$, where $b\,db\,d\phi$ is the infinitesimal cross sectional area of the beam that scatters into the solid angle defined by θ to $\theta + d\theta$ and ϕ to $\phi + d\phi$. The alternative notation for the differential cross section, $d\sigma/d\Omega$, comes from this interpretation.

To do an analytical calculation of $\sigma(\theta)$, we write

$$\sigma(\theta) = \frac{d\sigma}{d\Omega} = \frac{b}{\sin\theta}\left|\frac{db}{d\theta}\right|. \tag{5.28}$$

We see from (5.28) that the analytical calculation of $\sigma(\theta)$ involves b as a function of θ, or more precisely, how b changes to give scattering through an infinitesimally larger angle $\theta + d\theta$.

In a scattering experiment particles enter from the left (see Figure 5.7) with random values of the impact parameter b and azimuthal angle ϕ and the number of particles scattered into the various detectors is measured. In our simulation we know the value of b, and we can integrate Newton's equations of motion to find the angle at which the incident particle is scattered. Hence, in contrast to the analytical calculation, a simulation naturally yields θ as a function of b.

Because the differential cross section is usually independent of ϕ, we need to consider only beam particles at $\phi = 0$. We have to take into account the fact that in a real beam, there are more particles at some values of b than at others. That is, the number of particles in a real beam is proportional to $2\pi b\Delta b$, the area of the ring between b and $b + \Delta b$, where we have integrated over the values of ϕ to obtain the factor of 2π. Here Δb is the interval between the values of b used in the program. Because there is only one target in the beam, the target density is $n = 1/(\pi R^2)$.

The scattering program requires the `Scatter`, `ScatterAnalysis`, and `ScatterApp` classes. The `ScatterApp` class in Listing 5.6 organizes the startup process and creates the visualizations. As usual, it extends `AbstractSimulation` by overriding the `doStep` method. However, in this case a single step is not a time step. A step calculates a trajectory and scattering angle for the given impact parameter. After a trajectory is calculated, the impact parameter is incremented and the panel is repainted. If necessary, you can eliminate this visualization to increase the computational speed. If the new impact parameter exceeds the beam radius `bmax`, the animation is stopped and the accumulated data is analyzed. Note that the `calculateTrajectory` method returns `true` if the calculation succeeded and that an error message is printed if the calculation fails. Including a failsafe mechanism to stop a computation is good programming practice.

Listing 5.6: A program that calculates the scattering trajectories and computes the differential cross section.

```
package org.opensourcephysics.sip.ch05;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.frames.*;

public class ScatterApp extends AbstractSimulation {
   PlotFrame frame = new PlotFrame("x", "y", "Trajectories");
   ScatterAnalysis analysis = new ScatterAnalysis();
   Scatter trajectory = new Scatter();
   double vx;     // speed of the incident particle
   double b, db; // impact parameter and increment
   double bmax;  // maximum impact parameter

   public ScatterApp() {
      frame.setPreferredMinMax(-5, 5, -5, 5);
      frame.setSquareAspect(true);
   }

   public void doStep() {
      if(trajectory.calculateTrajectory(frame, b, vx)) {
         analysis.detectParticle(b, trajectory.getAngle());
      } else {
         control.println("Trajectory did not converge at b = "+b);
      }
      frame.setMessage("b = "+decimalFormat.format(b));
      b += db; // increases the impact parameter
      frame.repaint();
      if(b>bmax) {
         control.calculationDone("Maximum impact parameter reached");
         analysis.plotCrossSection(b);
      }
   }

   public void initialize() {
      vx = control.getDouble("vx");
      bmax = control.getDouble("bmax");
      db = control.getDouble("db");
      b = db/2; // starts b at average value of first interval 0->db
      // b will increment to 3*db/2, 5*db/2, 7*db/2, ...
      frame.setMessage("b = 0");
      frame.clearDrawables(); // removes old trajectories
      analysis.clear();
   }

   public void reset() {
      control.setValue("vx", 3);
      control.setValue("bmax", 0.25);
      control.setValue("db", 0.01);
```

```
      initialize();
   }

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

The `Scatter` class shown in Listing 5.7 calculates the trajectories by expressing the equation of motion as a rate equation. The most important method is `calculateTrajectory`, which calculates a trajectory by stepping the differential equation solver and adding the resulting data to a trail to display the path. Because the beam source is far away, we stop the calculation when the distance of the scattered particle from the target exceeds the initial distance. Note the use of the ternary `?:` operator. This very efficient and compact, operator uses three expressions. The first expression evaluates to a boolean. If this expression is true, then the statement after the `?` is executed. If this expression is false, then the statement after the `:` is executed. However, because some potentials may trap particles for long periods of time, we also stop the calculation after a predetermined number of time steps.

Listing 5.7: A class that models particle scattering using a central force law.

```
package org.opensourcephysics.sip.ch05;
import java.awt.*;
import org.opensourcephysics.display.*;
import org.opensourcephysics.frames.*;
import org.opensourcephysics.numerics.*;

public class Scatter implements ODE {
   double[] state = new double[5];
   RK4 odeSolver = new RK4(this);

   public Scatter() {
      odeSolver.setStepSize(0.05);
   }

   boolean calculateTrajectory(PlotFrame frame, double b, double vx) {
      state[0] = -5.0; // x
      state[1] = vx;   // vx
      state[2] = b;    // y
      state[3] = 0;    // vy
      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) {
      theta = Math.abs(theta); // treats positive and negative angles equally to get better
      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).

Problem 5.13. Scattering from a model hydrogen atom

a. Consider a model of the hydrogen atom for which a positively charged nucleus of charge $+e$ is surrounded by a uniformly distributed negative charge of equal magnitude. The spherically

symmetric negative charge distribution is contained within a sphere of radius a. It is straightforward to show that the force between a positron of charge $+e$ and this model hydrogen atom is given by

$$f(r) = \begin{cases} 1/r^2 - r/a^3, & r \le a \\ 0. & r > a \end{cases} \tag{5.30}$$

We have chosen units such that $e^2/(4\pi\epsilon_0) = 1$, and the mass of the positron is unity. What is the ionization energy in these units? Modify the `Scatter` class to incorporate this force. Is the force on the positron from the model hydrogen atom purely repulsive? Choose $a = 1$ and set the beam radius `bmax` $= 1$. Use $E = 0.125$ and $\Delta t = 0.01$. Compute the trajectories for $b = 0.25$, 0.5, and 0.75 and describe the qualitative nature of the trajectories.

b. Determine the cross section for $E = 0.125$. Choose nine bins so that the angular width of a detector is `delta` $= 20^\circ$ and let `db` $= 0.1$, 0.01, and 0.002. How does the accuracy of your results depend on the number of bins? Determine the differential cross section for different energies and explain its qualitative energy dependence.

c. What is the value of σ_T for $E = 0.125$? Does σ_T depend on E? The total cross section has units of area, but a point charge does not have an area. To what area does it refer? What would you expect the total cross section to be for scattering from a hard sphere?

d. Change the sign of the force so that it corresponds to electron scattering. How do the trajectories change? Discuss the change in $\sigma(\theta)$.

Problem 5.14. Rutherford scattering

a. One of the most famous scattering experiments was performed by Geiger and Marsden who scattered a beam of alpha particles on a thin gold foil. Based on these experiments, Rutherford deduced that the positive charge of the atom is concentrated in a small region at the center of the atom rather than distributed uniformly over the entire atom. Use a $1/r^2$ force in class `Scatter` and compute the trajectories for $b = 0.25$, 0.5, and 0.75 and describe the trajectories. Choose $E = 5$ and $\Delta t = 0.01$. The default value of x_0, the initial x-coordinate of the beam, is $x_0 = -5$. Is this value reasonable?

b. For $E = 5$ determine the cross section with `numberOfBins` $= 18$. Choose the beam width `bmax` $= 2$. Then vary `db` (or `numberOfBins`) and compare the accuracy of your results to the analytical result for which $\sigma(\theta)$ varies as $[\sin(\theta/2)]^{-4}$. How do your computed results compare with this dependence on θ? If necessary, decrease `db`. Are your results better or worse at small angles, intermediate angles, or large angles near 180°? Explain.

c. Because the Coulomb force is long range, there is scattering at all impact parameters. Increase the beam radius and determine if your results for $\sigma(\theta)$ change. What happens to the total cross section as you increase the beam width?

d. Compute $\sigma(\theta)$ for different values of E and estimate the dependence of $\sigma(\theta)$ on E.

Problem 5.15. Scattering by other potentials

a. A simple phenomenological form for the effective interaction between electrons in metals is the screened Coulomb (or Thomas-Fermi) potential given by

$$V(r) = \frac{e^2}{4\pi\epsilon_0 r} e^{-r/a}. \tag{5.31}$$

The range of the interaction a depends on the density and temperature of the electrons. The form (5.31) is known as the Yukawa potential in the context of the interaction between nuclear particles and as the Debye potential in the context of classical plasmas. Choose units such that $a = 1$ and $e^2/(4\pi\epsilon_0) = 1$. Recall that the force is given by $f(r) = -dV/dr$. Incorporate this force law into class `Scatter` and compute the dependence of $\sigma(\theta)$ on the energy of the incident particle. Choose the beam width equal to 3. Compare your results for $\sigma(\theta)$ with your results from the Coulomb potential.

b. Modify the force law in `Scatter` so that $f(r) = 24(2/r^{13} - 1/r^7)$. This form for $f(r)$ is used to describe the interactions between simple molecules (see Chapter 9). Describe some typical trajectories and compute the differential cross section for several different energies. Let `bmax` = 2. What is the total cross section? How do your results change if you vary `bmax`? Choose a small angle as the minimum scattering angle. How sensitive is the total cross section to this minimum angle? Does the differential cross section vary for any other angles beside the smallest scattering angle?

5.11 Three-body problems

Poincaré showed that it is impossible to obtain an analytical solution for the unrestricted motion of three or more objects interacting under the influence of gravity. However solutions are known for a few special cases, and it is instructive to study the properties of these solutions.

The `ThreeBody` class computes the trajectories of three particles of equal mass moving in a plane and interacting under the influence of gravity. Both the physics and the drawing are implemented in the `ThreeBody` class shown in Listing 5.9. Note that the `getRate` and `computeForce` methods compute trajectories for an arbitrary number of masses. Note how the `computeForce` method uses the `arraycopy` method to quickly zero the arrays. To simplify the drawing of the particle trajectories, the `ThreeBody` class uses an inner class that extends a `Circle` and contains a `Trail`.

Listing 5.9: A class that models the dynamics of the three-body problem.

```
package org.opensourcephysics.sip.ch05;
import java.awt.*;
import org.opensourcephysics.display.*;
import org.opensourcephysics.numerics.*;

public class ThreeBody implements Drawable, ODE {
   int n = 3; // number of interacting bodies
   // state= {x1, vx1, y1, vy1, x2, vx2, y2, vy2, x3, vx3, y3, vy3, t}

   double[] state = new double[4*n+1];
```

```
double[]
   force = new double[2*n], zeros = new double[2*n];
ODESolver odeSolver = new RK45MultiStep(this);
Mass mass1 = new Mass(), mass2 = new Mass(), mass3 = new Mass();

public void draw(DrawingPanel panel, Graphics g) {
   mass1.draw(panel, g);
   mass2.draw(panel, g);
   mass3.draw(panel, g);
}

public void doStep() {
   odeSolver.step();
   mass1.setXY(state[0], state[2]);
   mass2.setXY(state[4], state[6]);
   mass3.setXY(state[8], state[10]);
}

void initialize(double[] initState) {
   System.arraycopy(initState, 0, state, 0, 13); // copies initstate to state
   mass1.clear();                                // clears data from old trail
   mass2.clear();
   mass3.clear();
   mass1.setXY(state[0], state[2]);
   mass2.setXY(state[4], state[6]);
   mass3.setXY(state[8], state[10]);
}

void computeForce(double[] state) {
   System.arraycopy(zeros, 0, force, 0, force.length); // sets force array elements to 0
   for(int i = 0;i<n;i++) {
      for(int j = i+1;j<n;j++) {
         double dx = state[4*i]-state[4*j];
         double dy = state[4*i+2]-state[4*j+2];
         double r2 = dx*dx+dy*dy;
         double r3 = r2*Math.sqrt(r2);
         double fx = dx/r3;
         double fy = dy/r3;
         force[2*i] -= fx;
         force[2*i+1] -= fy;
         force[2*j] += fx;
         force[2*j+1] += fy;
      }
   }
}

public void getRate(double[] state, double[] rate) {
   computeForce(state); // force array alternates fx and fy
   for(int i = 0;i<n;i++) {
      int i4 = 4*i;
```

```
            rate[i4] = state[i4+1];     // x rate is vx
            rate[i4+1] = force[2*i];    // vx rate is fx
            rate[i4+2] = state[i4+3];   // y rate is vy
            rate[i4+3] = force[2*i+1]; // vy rate is fy
        }
        rate[state.length-1] = 1; // time rate is last
    }

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

    class Mass extends Circle {
        Trail trail = new Trail();
        // Draws the mass.

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

        // Clears trail
        void clear() {
            trail.clear();
        }

        // Sets postion and adds to trail
        public void setXY(double x, double y) {
            super.setXY(x, y);
            trail.addPoint(x, y);
        }
    }
}
```

The initial conditions for our examples are contained in the `ThreeBodyInitialConditions` class. This file is available in the ch05 package but is not listed here because it contains mostly numeric data.

In 1765 Euler discovered an analytical solution in which three masses start on a line and rotate so that the central mass stays fixed. The `EULER` array in `ThreeBodyInitialConditions` initializes the model to produce this type of solution. The first mass is placed at the center and the other two masses are placed on opposite sides with velocities that are equal but opposite. Because of the symmetry, the trajectories are ellipses with a common focus at the center.

A second analytic solution to the unrestricted three-body problem was found by Lagrange in 1772. This solution starts with three masses at the corners of an equilateral triangle. Each mass moves in an ellipse in such a way that the triangle formed by the messes remains equilateral. The `LAGRANGE` array initializes this solution.

A spectacular new solution that adds to the sparse list of analytic three-body solutions was first discovered numerically by Moore and proven to be stable by Chenciner and Montgomery. The

`MONTGOMERY` array contains the initial conditions for this solution.

The `ThreeBodyApp` class in Listing 5.10 is the target class for the three-body program. The `doStep` method merely increments the model's differential equations solver and repaints the view.

Listing 5.10: A program that displays the trajectories of three bodies interacting via gravitational forces.

```
package org.opensourcephysics.sip.ch05;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.frames.*;

public class ThreeBodyApp extends AbstractSimulation {
   PlotFrame frame = new PlotFrame("x", "y", "Three-Body Orbits");
   ThreeBody trajectory = new ThreeBody();

   public ThreeBodyApp() {
      frame.addDrawable(trajectory);
      frame.setSquareAspect(true);
      frame.setSize(450, 450);
   }

   public void initialize() {
      trajectory.odeSolver.setStepSize(control.getDouble("dt"));
      trajectory.initialize(ThreeBodyInitialConditions.MONTGOMERY);
      frame.setPreferredMinMax(-1.5, 1.5, -1.5, 1.5);
   }

   public void reset() {
      control.setValue("dt", 0.1);
      enableStepsPerDisplay(true);
      initialize();
   }

   protected void doStep() {
      trajectory.doStep();
      frame.setMessage("t="+decimalFormat.format(trajectory.state[4]));
   }

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

Problem 5.16. Stability of solutions to the three-body problem

Examine the stability of the three solutions to the three-body problem by slightly varying the initial velocity of one of the masses. Before passing your new initial state to `trajectory.initialize`, calculate the center of mass velocity and subtract this velocity from every object. Show that any instability is due to the physics and not to the numerical differential equation solver. Which of the three analytic solutions is stable? Check conservation of the total energy and angular momentum.

5.12 Projects

Project 5.17. Effect of a "solar wind"

a. Assume that a satellite is affected not only by the Earth's gravitational force, but also by a weak uniform "solar wind" of magnitude W acting in the horizontal direction. The equations of motion can be written as

$$\frac{d^2x}{dt^2} = -\frac{GMx}{r^3} + W \tag{5.32a}$$

$$\frac{d^2y}{dt^2} = -\frac{GMy}{r^3}. \tag{5.32b}$$

Choose initial conditions so that a circular orbit would be obtained for $W = 0$. Then choose a value of W whose magnitude is about 3% of the acceleration due to the gravitational field and compute the orbit. How does the orbit change?

b. Determine the change in the velocity space orbit when the solar wind (5.32) is applied. How does the total angular momentum and energy change? Explain in simple terms the previously observed change in the position space orbit. See Luehrmann for further discussion of this problem.

Project 5.18. Resonances and the asteroid belt

a. A histogram of the number of asteroids versus their distance from the Sun shows some distinct gaps. These gaps, called the *Kirkwood gaps*, are due to resonance effects. That is, if asteroids were in these gaps, their periods would be simple fractions of the period of Jupiter. Modify class `Planet2` so that planet two has the mass of Jupiter by setting `GM1 = 0.001*GM`. Because the asteroid masses are very small compared to that of Jupiter, the gravitational force on Jupiter due to the asteroids can be neglected. The initial conditions listed in `Planet2` are approximately correct for Jupiter. The initial conditions for the asteroid (planet one in `Planet2`) correspond to the 1/3 resonance (the period of the asteroid is one third that of Jupiter). Run the program with these changes and describe the orbit of the asteroid.

b. Use Kepler's third law, $T^2/a^3 =$ constant, to determine the values of a, the asteroid's semimajor axis, such that the ratio of its period of revolution about the Sun to that of Jupiter is 1/2, 3/7, 2/5, and 2/3. Set the initial value of `x(1)` equal to a for each of these ratios and choose the initial value of `vy(1)` so that the asteroid would have a circular orbit if Jupiter were not present. Describe the orbits that you obtain.

c. It is instructive to plot a as a function of time. However, because it is not straightforward to measure a directly in the simulation, it is more convenient to plot the quantity $-2GMm/E$, where E is the total energy of the asteroid and m is the mass of the asteroid. Because E is proportional to m, the quantity $-2GMm/E$ is independent of m. If the interaction of the asteroid with Jupiter is ignored, it can be shown that $a = -2GMm/E$, where E is the asteroid kinetic energy plus the asteroid-Sun potential energy. Derive this result for circular orbits. Plot the quantity $-2GMm/E$ versus time for about thirty revolutions for the initial conditions in Problem 5.18b.

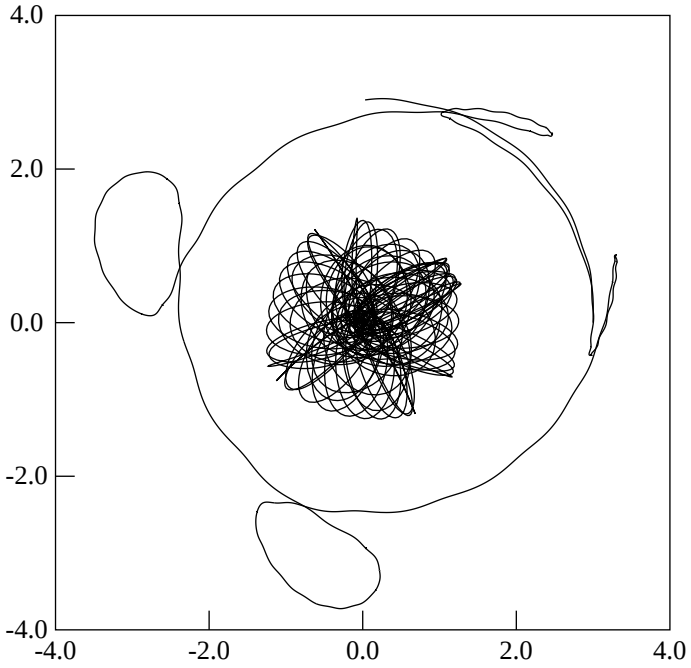

Figure 5.8: Orbits of the two electrons in the classical helium atom with the initial condition $\mathbf{r}_1 = (3, 0), \mathbf{r}_2 = (1, 0), \mathbf{v}_1 = (0, 0.4)$, and $\mathbf{v}_2 = (0, -1)$ (see Project 5.19c).

d. Compute the time dependence of $-2GMm/E$ for asteroid orbits whose initial position `x(1)` ranges from 2.0 to 5.0 in steps of 0.2. Choose the initial values of `vy(1)` so that circular orbits would be obtained in the absence of Jupiter. Are there any values of `x(1)` for which the time dependence of a is unusual?

e. Make a histogram of the number of asteroids versus the value of $-2GMm/E$ at $t = 2000$. (You can use the `HistogramFrame` class described on page 227 if you wish.) Assume that the initial value of `x(1)` ranges from 2.0 to 5.0 in steps of 0.02 and choose the initial values of `vy(1)` as before. Use a histogram bin width of 0.1. If you have time, repeat for $t = 5000$, and compare the histogram with your previous results. Is there any evidence for Kirkwood gaps? A resonance occurs when the periods of the asteroid and Jupiter are related by simple fractions. We expect the number of asteroids with values of a corresponding to resonances to be small.

f. Repeat part (e) with initial velocities that vary from their values for a circular orbit by 1, 3, and 5%.

Project 5.19. The classical helium atom

The classical helium atom is a relatively simple example of a three-body problem and is similar to the gravitational three-body problem of a heavy sun and two light planets. The important difference is that the two electrons repel one another, unlike the planetary case where the intraplanetary

interaction is attractive. If we ignore the small motion of the heavy nucleus, the equations of motion for the two electrons can be written as

$$\mathbf{a}_1 = -2\frac{\mathbf{r}_1}{r_1^3} + \frac{\mathbf{r}_1 - \mathbf{r}_2}{r_{12}^3} \tag{5.33a}$$

$$\mathbf{a}_2 = -2\frac{\mathbf{r}_2}{r_2^3} + \frac{\mathbf{r}_2 - \mathbf{r}_1}{r_{12}^3}, \tag{5.33b}$$

where $\mathbf{r}_1$ and $\mathbf{r}_2$ are measured from the fixed nucleus at the origin, and r_{12} is the distance between the two electrons. We have chosen units such that the mass and charge of the electron are both unity. The charge of the helium nucleus is two in these units. Because the electrons are sometimes very close to the nucleus, their acceleration can become very large, and a very small time step Δt is required. It is not efficient to use the same small time step throughout the simulation and instead a variable time step or an *adaptive* step size algorithm is suggested. An adaptive step size algorithm can be used with any standard numerical algorithm for solving differential equations. The RK45 algorithm described in Appendix 3A is adaptive and is a good all-around choice for these types of problems.

a. For simplicity, we restrict our atom to two dimensions. Modify `Planet2` to simulate the classical helium atom. Choose units such that the electron mass is one and the other constants are absorbed into the unit of charge so that the force between two electrons is

$$|F| = \frac{1}{r^2}. \tag{5.34}$$

Choose the initial value of the time step to be $\Delta t = 0.001$. Some of the possible orbits are similar to those we have seen in our mini-solar system. For example, try the initial condition $\mathbf{r}_1 = (2, 0), \mathbf{r}_2 = (-1, 0), \mathbf{v}_1 = (0, 0.95)$, and $\mathbf{v}_2 = (0, -1)$.

b. Most initial conditions result in unstable orbits in which one electron eventually leaves the atom (autoionization). The initial condition $\mathbf{r}_1 = (1.4, 0), \mathbf{r}_2 = (-1, 0), \mathbf{v}_1 = (0, 0.86)$, and $\mathbf{v}_2 = (0, -1)$ gives "braiding" orbits. Make small changes in this initial condition to observe autoionization.

c. The classical helium atom is capable of very complex orbits (see Figure 5.8). Investigate the motion for the initial condition $\mathbf{r}_1 = (3, 0), \mathbf{r}_2 = (1, 0), \mathbf{v}_1 = (0, 0.4)$, and $\mathbf{v}_2 = (0, -1)$. Does the motion conserve the total angular momentum? Also try $\mathbf{r}_1 = (2.5, 0), \mathbf{r}_2 = (1, 0), \mathbf{v}_1 = (0, 0.4)$, and $\mathbf{v}_2 = (0, -1)$.

d. Choose the initial condition $\mathbf{r}_1 = (2, 0), \mathbf{r}_2 = (-1, 0)$, and $\mathbf{v}_2 = (0, -1)$. Then vary the initial value of $\mathbf{v}_1$ from $(0.6, 0)$ to $(1.3, 0)$ in steps of $\Delta v = 0.02$. For each set of initial conditions calculate the time it takes for autoionization. Assume that ionization occurs when either electron exceeds a distance of six from the nucleus. Run each simulation for a maximum time of 2000. Plot the ionization time versus v_{1x}. Repeat for a smaller interval of Δv centered about one of the longer ionization times. These calculations require much computer resources. Do the two plots look similar? If so, such behavior is called "self-similar" and is characteristic of chaotic systems and the geometry of fractals (see Chapters 7 and 14). More discussion on the nature of the orbits can be found in Yamamoto and Kaneko.

References and Suggestions for Further Reading

Harold Abelson, Andrea diSessa, and Lee Rudolph, "Velocity space and the geometry of planetary orbits," Am. J. Phys. **43**, 579–589 (1975). See also Andrea diSessa, "Orbit: a mini-environment for exploring orbital mechanics," in O. Lecarme and R. Lewis, editors, *Computers in Education*, 359, North-Holland (1975). Detailed geometrical rather than calculus-based arguments on the origin of closed orbits for inverse-square forces are presented. Sections 5.7 and 5.8 are based on these papers.

Ralph Baierlein, *Newtonian Dynamics*, McGraw-Hill (1983). An intermediate level text on mechanics. Of particular interest are the discussions on the stability of circular orbits and the effects of an oblate sun.

John J. Brehm and William J. Mullin, *Introduction to the Structure of Matter*, John Wiley & Sons (1989). See Section 3-4 for a discussion of Rutherford scattering.

Alain Chenciner and Richard Montgomery, "A remarkable periodic solution of the three-body problem in the case of equal masses," Annals of Mathematics **152**, 881–901 (2000).

J. M. A. Danby, *Computer Modeling: From Sports To Spaceflight . . . From Order To Chaos*, William-Bell (1997). See Chapter 11 for a discussion of orbits including an excellent treatment of the Lagrange points.

R. P. Feynman, R. B. Leighton, M. Sands, *The Feynman Lectures in Physics, Vol. 1*, Addison-Wesley (1963). See Chapter 9.

A. P. French, *Newtonian Mechanics*, W. W. Norton & Company (1971). An introductory level text with more than a cursory treatment of planetary motion.

Ian R. Gatland, "Numerical integration of Newton's equations including velocity-dependent forces," Am J. Phys. **62**, 259 (1994). The author chooses a variable time step based on the difference in the calculation of the positions rather than the energy as we did in Project 5.19.

Herbert Goldstein, Charles P. Poole, and John L. Safko, *Classical Mechanics*, third edition, Addison-Wesley (2002). Chapter 3 has an excellent discussion of the Kepler problem and the conditions for a closed orbit.

Myron Lecar and Fred A. Franklin, "On the original distribution of the asteroids. I," Icarus **20**, 422–436 (1973). The authors use simulations of the motions of asteroids and discuss the Kirkwood gaps.

Arthur W. Luehrmann, "Orbits in the solar wind – a mini-research problem," Am. J. Phys. **42**, 361 (1974). Luehrmann emphasizes the desirability of student problems requiring inductive rather than deductive reasoning.

Jerry B. Marion and Stephen T. Thornton, *Classical Dynamics*, fifth edition, Harcourt (2004). Chapter 8 discusses central force motion, the precession of the Mercury, and the stability of circular orbits.

Michael McCloskey, "Intuitive physics," Sci. Am. **248** (4), 122–130 (1983). A discussion of the counterintuitive nature of Newton's laws.

John R. Merrill and Richard A. Morrow, "An introductory scattering experiment by simulation," Am. J. Phys. **38**, 1104–1107 (1970).

C. Moore, "Braids in classical gravity," Phys. Rev. Lett. **70**, 3675–3679 (1993).

Bernard Schutz, *Gravity from the ground up*, Cambridge University Press (2003). The associated Web site, `<http://www.gravityfromthegroundup.org/>`, has many Java programs including a simulation of the orbit of a planet around a black hole or neutron star, using the equation of motion appropriate for general relativity.

Tomomyuki Yamamoto and Kunihiko Kaneko, "Helium atom as a classical three-body problem," Phys. Rev. Lett. **70**, 1928–1931 (1993).