Conservation of Momentum and Energy

7.1 A binary-star system

In the previous chapter we considered the motion of a planet subject to a central force that was always directed from the planet towards the origin where a star was sitting motionless. This is, of course, not quite correct, since the star also experiences a gravitational pull towards the planet and must therefore also move. This motion is usually very small since stars tend to be much heavier than planets, but it is not zero.

Let's consider now the full two-body problem. To give us maximal flexibility in parameters, we can consider the two bodies both stars in a binary-star system. We will creatively call the heavier star "bigstar" and the lighter star "smallstar". We could, of course, proceed exactly like before, modifying the code in Chapter 6 to include the dynamics of the second object (i.e., the sun). But to illustrate another important physical concept, namely that of momentum and momentum conservation, let us incorporate this quantity in the code.

To review briefly, momentum is defined by $\vec{p} = m\vec{v}$. An object has momentum by virtue of having mass and velocity. We can also talk of the total momentum of a system of objects. In that case we simply add the individual momenta of all the objects that make up the system together (as vectors). Now, we can prove that when this system is *isolated*, in other words when nothing outside this system exists that would push or pull on the objects within the system, then the total momentum of the system doesn't change. We say that the total momentum is *conserved*.

Another useful concept in this context is the *center of mass* of a system comprised of discrete objects/particles. We can compute the coordinates of the center of mass using the well-known formula,

$$x_{com} = \frac{m_1 x_1 + m_2 x_2 + ...}{m_1 + m_2 + ...},$$

and similarly for the y-coordinate. Since VPython deals well with vector quantities, we can also dispense with the individual components and refer only the position vectors, $\vec{r}_i$, of the objects themselves:

$$\vec{r}_{com} = \frac{\sum m_i \vec{r}_i}{\sum m_i}. \tag{7.1}$$

You may have learned that whenever the total momentum of a system is conserved, i.e., when there is no net external force acting on the system, then the center of mass cannot experience any acceleration. In our notion, we can write,

$$\vec{a}_{com} = \frac{d^2}{dt^2}(\vec{r}_{com}) = 0. \tag{7.2}$$

The proof of this statement is actually not difficult and well within the reach of an introductory physics student. See if you can work it out yourself.

One of our objectives will now be to "prove" this property about the center of mass numerically by simply computing its location for each time-step and then to observe its motion across the screen. Another will be to show that in this binary-star system, total momentum will be conserved. It should be conserved because the way we have set up the problem, the system is clearly isolated - there are no other objects around at all.

```
GlowScript 2.7 VPython

scene.range=5e11

G = 6.7e-11

bigstar = sphere(pos=vector(-2e11,0,0), radius=2e10, color=color.red,
                 make_trail=False, interval=10)
bigstar.mass = 3e30
bigstar.p = vector(0, 3e3, 0) * bigstar.mass

smallstar = sphere(pos=vector(2e11,0,0), radius=1e10, color=color.yellow,
                 make_trail=False, interval=10)
smallstar.mass = 1e30
smallstar.p = vector(0, -1e4, 0) * smallstar.mass

dt = 1e5

centerofmass =
(bigstar.mass*bigstar.pos+smallstar.mass*smallstar.pos)/(bigstar.mass+smallstar.mass)

COM = cone(pos=centerofmass, axis=vector(0,1e10,0),radius=1e10,color=color.green,
make_trail=True)

while True:
    rate(200)
    r = bigstar.pos - smallstar.pos
    F = G * bigstar.mass * smallstar.mass * r.hat / mag2(r)

    bigstar.p = bigstar.p - F*dt
    smallstar.p = smallstar.p + F*dt

    bigstar.pos = bigstar.pos + (bigstar.p/bigstar.mass) * dt
    smallstar.pos = smallstar.pos + (smallstar.p/smallstar.mass) * dt
    COM.pos = (bigstar.mass*bigstar.pos+smallstar.mass*smallstar.pos)/(bigstar.mass+
    smallstar.mass)

    print ("The total momentum is", bigstar.p + smallstar.p)
```

Figure 7.1: The code for the binary-star system.

Let us again start with a code template - see Figure 7.1. One thing you will notice immediately is that we are now not working in artificial units (as before) and instead use the true value for the gravitational constant. That also forces us to use astronomically realistic values for all other quantities in the problem, such as for the masses and distances involved. Again, as long as all the inputs are in SI-units, so will all the computed outputs.

As you can see, in the first six command lines, we basically set up the properties about the two stars. For instance, the bigger star has a mass of 3×10^{30} kg - three times that of the smaller star. For

comparison, our sun's mass is about 2×10^{30} kg. One of the things you will have a chance to play around with is the mass ratio of the two stars.

The next line (Line 17) defines the computational time step. This might strike you as an incredibly large time step, especially compared to the values we have used before, but remember - everything has to be astronomical, including time. The time step of 10^5 seconds translates to a duration of slightly longer than an earth day. Compared to the period of revolution this is still quite small.

Lines 19 and 20 are recognized from Equation (7.1) as nothing other than $\vec{r}_{com}$. The next two lines are there just so we can visualize the location of center of mass on the screen, here in the aspect of a green *cone* whose trajectory we will also keep track of via the "make_trail" command.

The iterative part of the program starts with Line 25. We define the vector, **r**, that runs from the small star to the big star. We could have also reversed it, but it is important to be consistent. The way it is defined, it will be parallel to the force on the small star and anti-parallel to the force on the big star. By Newton's third law, these two interaction forces must be equal and opposite.

Lines 30 and 31 implement Newton's second law,

$$\vec{F} = \frac{d\vec{p}}{dt}. \tag{7.3}$$

If we solve this for the small change in momentum, we get: $dp = Fdt$; in other words, it takes a force to change the momentum. This is also sometimes referred to as the *impulse-momentum* theorem. Once we have calculated the new momentum, we can also update the position of the corresponding star, since momentum, of course, is intimately related to velocity - we get velocity by dividing the momentum by mass. This way, we do not have to explicitly refer to

velocity at all in the WHILE loop.

7.2 Exercises

- Run the program - does the computed total momentum change over time? Is physics correct? Also examine the code - is the result at all surprising?

- You will also notice that the center of mass does not stay stationary, but that it moves on the screen. We can make that motion cease if we give the two stars initial momenta that add to zero. Show that right now (i.e., in Figure 7.1) they two initial momenta do not add to zero.

- Now change the initial conditions such that the total momentum is in fact zero initially. What do you notice about the center of mass motion on the screen? Also change the viewer's perspective to see the two-body motion from different angles.

- Make an additional change in initial conditions (positions and velocities) and watch the orbits of the two stars around each other. Consider turning on the "trails" on the orbits for better visualization.

- VPython allows you to view the motion from the perspective of different observers (or reference frames). The command is **scene.camera.follow(bigstar)**. In the parenthesis appears the name of the object that you want to make as the reference frame. In the command above we take the big star as the observer's reference frame. To make things less confusing,

turn off all the trails again.

- Now that you have explored the role of initial conditions a bit, let's turn to the mass ratio. Try one case where the ratio is much larger than 3, and then one case for the ratio is close to 1. What do you conclude from those two scenarios.

7.3 Energy in the Spring-Mass System

In Section 6.2, we explored the motion of a mass at the end of a spring. We saw the sinusoidal motion that resulted from the combination of Newton's second law and Hooke's law. Now we can revisit this problem and analyze it through the lens of potential and kinetic energy.

Remember that the potential energy stored in a stretched or compressed spring, also called elastic potential energy, is given by $U = \frac{1}{2}kx^2$. Furthermore, the kinetic energy of the end-mass is given by $K = \frac{1}{2}mv^2$. Finally, the total mechanical energy is simply the sum of the two energies, $E_{mech} = U + K$.

Modify the code, as given in Figure 6.1, in the following manner.

- Add some lines that will compute the three energies, U, K, and E_{mech}.

- Then, instead of plotting position and velocity as a function of time, instead plot the three energies that are now being computed at each time step. What do you observe? Is the total mechanical energy conserved over time? Explain!

7.4 Simulating the Rutherford experiment

Another example that we will see is actually mathematically very similar to the binary-star system is the famous Rutherford scattering experiment. We know that two like charges repel via the Coulomb force. This force has the same structure as the law of universal gravitation. It is given by,

$$\vec{F}_c = k\frac{q_1 q_2}{r^2}\hat{r}, \tag{7.4}$$

where the Coulomb constant $k = 9.0 \times 10^9$ N m^2/C^2 plays the role of the gravitational constant, G. Notice how this force also depends inversely on the square of the distance between the two objects, and that it is again a *central* force (indicated by the direction vector $\hat{r}$).

Ernest Rutherford's experiment was to shoot alpha particles, which are fast Helium nuclei, at a thin gold foil. The idea was to see if and how the alpha particles would be deflected upon hitting the foil. What Rutherford discovered was that every once in a while the deflection angle was very large so that the alpha particle was scattered back to the source. The explanation for these large deflections is that the positive charge in the gold atoms is actually highly concentrated in the nucleus (and not smeared out). In fact, while the size of the gold atom is on the order of 10^{-10} meters, the nucleus is 100,000 times smaller, i.e., on the order of 10^{-15} meters.

So if the positively charged alpha particle hits a gold nucleus close to head-on, since the two repel it should be deflected back. At the same time, the Gold nucleus should also experiences some recoil. (If it didn't, momentum could not be conserved during the collision.) Let us now use VPython to explore the different scenarios that can

occur when we shoot an alpha particle at a gold nucleus.

While Equation (7.4) is very similar in form to Equation (6.7), the length scales of the nuclear-collision problem could not be more different from that of the binary-star problem. In the nuclear problem, a typical distance (or *characteristic* length scale) is on the order of 10^{-15} m, whereas in the gravitational context it was 10^{+11} m - an incredible difference of 26 orders of magnitude. With this change in spatial scale come other difference, such as in characteristic mass and time. In short, we are now dealing with vastly smaller lengths, vastly tinier masses, and vastly shorter times. Luckily for us, VPython can handle all of these changes very well.

Figure 7.2 shows the new setup for the nuclear context.[1] The first thing we notice is the new **scene.range** which catapults us into the microcosm. Next we define the **nucleus** and its properties of mass, charge and momentum.

In Line 13, we introduce the so-called *impact parameter*, b. This will be an important control parameter for us to vary from run to run. This parameter tells us, roughly speaking, by how much the alpha-particle would miss the nucleus if it felt no repulsive force. Thus, $b = 0$ corresponds to a perfectly head-on collision.

Next, we define the alpha particle and its properties of mass, charge and momentum. Notice that we give the alpha particle some initial speed to the right. In fact, this initial speed is quite large, $v_0 = 5 \times 10^7$ m/s, about 17 percent of the speed of light. We need large speeds for the alpha particle to get close to the gold nucleus. This initial speed is another parameter we will be able to adjust later.

Line 22-25 set up arrows for the momenta of the two particles (alpha

[1] This code is loosely based on the code by R. Chabay, "03-particle collision", available in the Matter-and-Interactions's Glowscript-programs folder.

```
1  GlowScript 2.8 VPython
2
3  scene.range = 3.5e-14
4
5  k = 9e9
6
7  nucleus = sphere(pos=vector(0,-1e-14,0), radius=2e-15, color=color.red, make_trail=True)
8  nucleus.mass = 1.32e-25
9  nucleus.q = 79*1.6e-19
10 nucleus.p = vector(0, 0, 0)
11
12 #The impact parameter, b
13 b = 1.5e-15
14
15 alpha = sphere(pos=vector(-2.5e-14,b-1e-14,0), radius=1e-15, color=color.yellow, make_trail=True)
16 alpha.mass = 6.64e-27
17 alpha.q = 4*1.6e-19
18 alpha.p = vector(5e7, 0, 0) * alpha.mass
19
20 dt = 5e-24
21
22 scale=2.5e4
23 p1=arrow(pos=vector(1e-14,1e-14,0),axis=alpha.p*scale, color=color.yellow, shaftwidth=7e-16)
24 p2=arrow(pos=vector(1e-14,1e-14,0),axis=nucleus.p*scale, color=color.red, shaftwidth=7e-16)
25 ptot=arrow(pos=vector(1e-14,1e-14,0),axis=(nucleus.p+alpha.p)*scale, color=color.white, shaftwidth=7e-16
26
27 #Start-Pause Button
28 run = False
29 def Runbutton(r):
30     global run
31     run = not run
32     if run:
33         r.text = "Pause"
34     else:
35         r.text = "Run"
36 button(text='Run', bind=Runbutton)
```

Figure 7.2: Setting up the Rutherford-scattering code

and Gold nucleus) and the total momentum. In order to see these arrows on the screen, they have to be stretched by the scale-factor, **scale**.

Finally, we have added here for the first time a start-pause button, which will allow us to start the code and pause it at any time. The details of this section of code need not concern us here.

The main part of the code, namely the WHILE loop, is shown in Figure 7.3. We should recognize large chunks of it from before. Lines 52 to 55 are new. They will update the momentum vectors throughout the interaction of the two particles. The lengths and direction of the three momenta (alpha, nucleus, and total) are set via the **.axis** assignment. Additionally, we have displaced the *tail* of the momentum vector associated with the nucleus to coincide with the *head* of

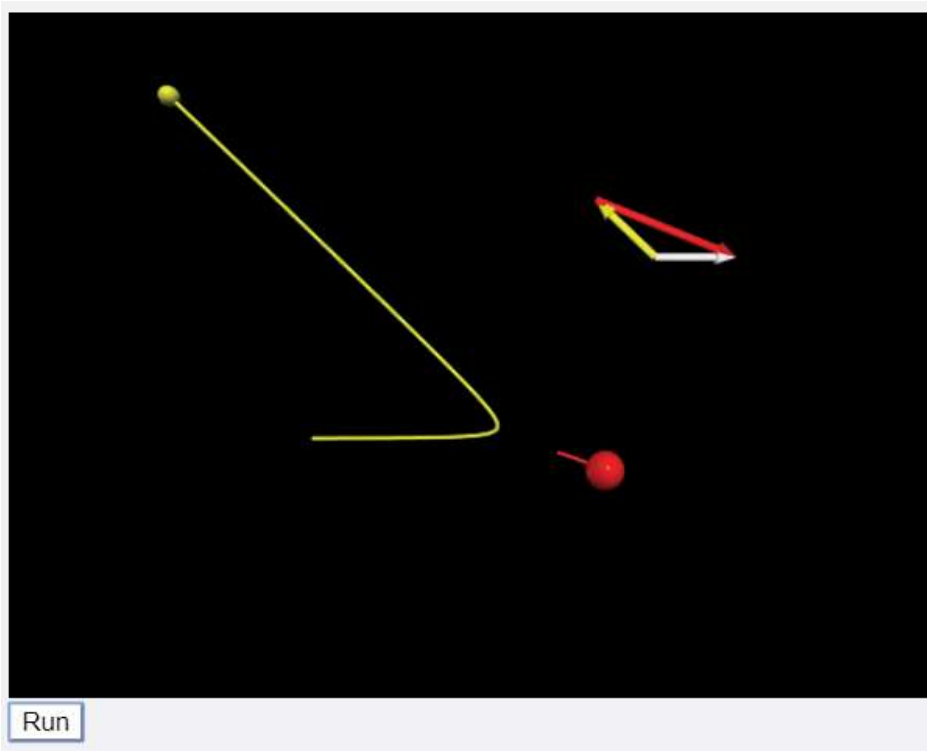
the alpha's momentum vector. This way we can verify visually that the two individual momentum vectors do indeed add to the same total momentum vector (via the head-to tail method of adding vectors).

```
39 while True:
40     rate(50)
41     if not run: continue
42 
43     r = alpha.pos - nucleus.pos
44     F = k * alpha.q * nucleus.q * r.hat / mag2(r)
45 
46     alpha.p = alpha.p + F*dt
47     nucleus.p = nucleus.p - F*dt
48 
49     nucleus.pos = nucleus.pos + (nucleus.p/nucleus.mass) * dt
50     alpha.pos = alpha.pos + (alpha.p/alpha.mass) * dt
51 
52     p1.axis = alpha.p*scale
53     p2.axis = nucleus.p*scale
54     p2.pos = vector(1e-14,1e-14,0) + alpha.p*scale
55     ptot.axis = (alpha.p + nucleus.p)*scale
```

Figure 7.3: The core of the Rutherford-scattering code

7.5 Exercises

- Run the code and observe the trajectories of the two particles. Also observe the particle's momentum vectors in the upper-right corner. You should see something like this:



- Verify that the momentum vectors are in fact tangentially parallel to the trajectory at any given instant of time. Why is the red arrow so large if the gold nucleus seems to move so slowly?

- Given the vector diagrams rendered in the upper-right corner of the screen, is momentum conserved throughout the scattering process? How do you know?

- Adjust the impact parameter, b. Make it progressively smaller (all the way to zero) and observe what happens to the deflection angle of the alpha particle.

- Now make b incrementally bigger than the value in Figure 7.2. What do you see now? Is momentum still conserved?

- You can now explore the role of the initial speed of the alpha particle. How does the angle of deflection seem to depend on this initial speed?