7. MOMENTUM AND COLLISIONS

I'm reminding myself that this is a book about numerical calculations in physics using python. It is NOT a physics textbook. So, I don't have to go over ALL the physics stuff about collisions. However, I'm going to say something anyway because I can't help myself.

Remember, we have the momentum principle.

$$\vec{F}_{\text{net}} = \frac{\Delta \vec{p}}{\Delta t}$$

Where $\vec{p}$ is the momentum. That's cool. Now imagine that we have two objects interacting. I'm going to call them object A and B. From Newton's 3rd law, the force that A exerts on B is equal and opposite to the force that B exerts on A.

$$\vec{F}_{A-B} = -\vec{F}_{B-A}$$

Since the two objects interact for the same time interval (Δt), the change in momentum for object A must be the opposite of the change in momentum for B. Or, we could just say the total change in momentum is the zero vector. Or we could say the initial momentum is equal to the final momentum. Or we could say momentum is conserved.

$$\Delta \vec{p}_A = -\Delta \vec{p}_B$$

$$\Delta \vec{p}_{\text{total}} = \vec{0}$$

$$\vec{p}_i = \vec{p}_f$$

Note that this only works if the only forces on A and B are due to the interaction between them. If there's a rocket on object A then the net force is not just the force from B.

When we talk about collisions, there's really three kinds:

- Inelastic. Here two objects collide and stick together. This means that momentum is conserved AND the two objects have the same final velocity.
- Elastic. In this case, the objects "bounce". Not only is momentum conserved, but the kinetic energy is also conserved.

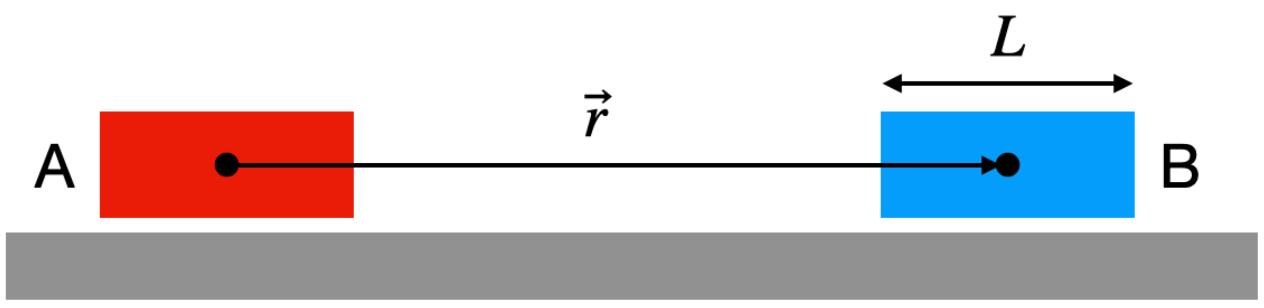
- Plain. A plain collision is somewhere between elastic and inelastic. The two objects collide but don't stick together. Momentum is conserved, but kinetic energy is not conserved.

OK, physics lesson over. Let's model this thing.

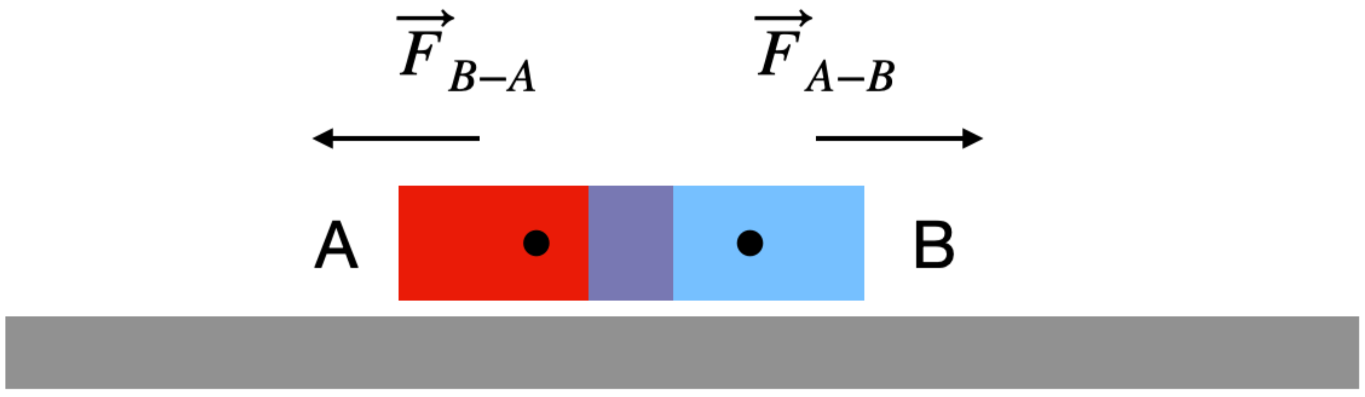
1D ELASTIC COLLISIONS

It's a classic experiment in the physics lab. You get two of those low friction carts (from either PASCO or Vernier) and then crash them together. Most of these carts either have a springy bumper or repelling magnets in one end. This means you can get a nice elastic collision.

But how do we model this kind of interaction? The answer is with a spring. Imagine the two cars have a length L and move towards each other. We can then calculate the vector distance between the center of the carts.



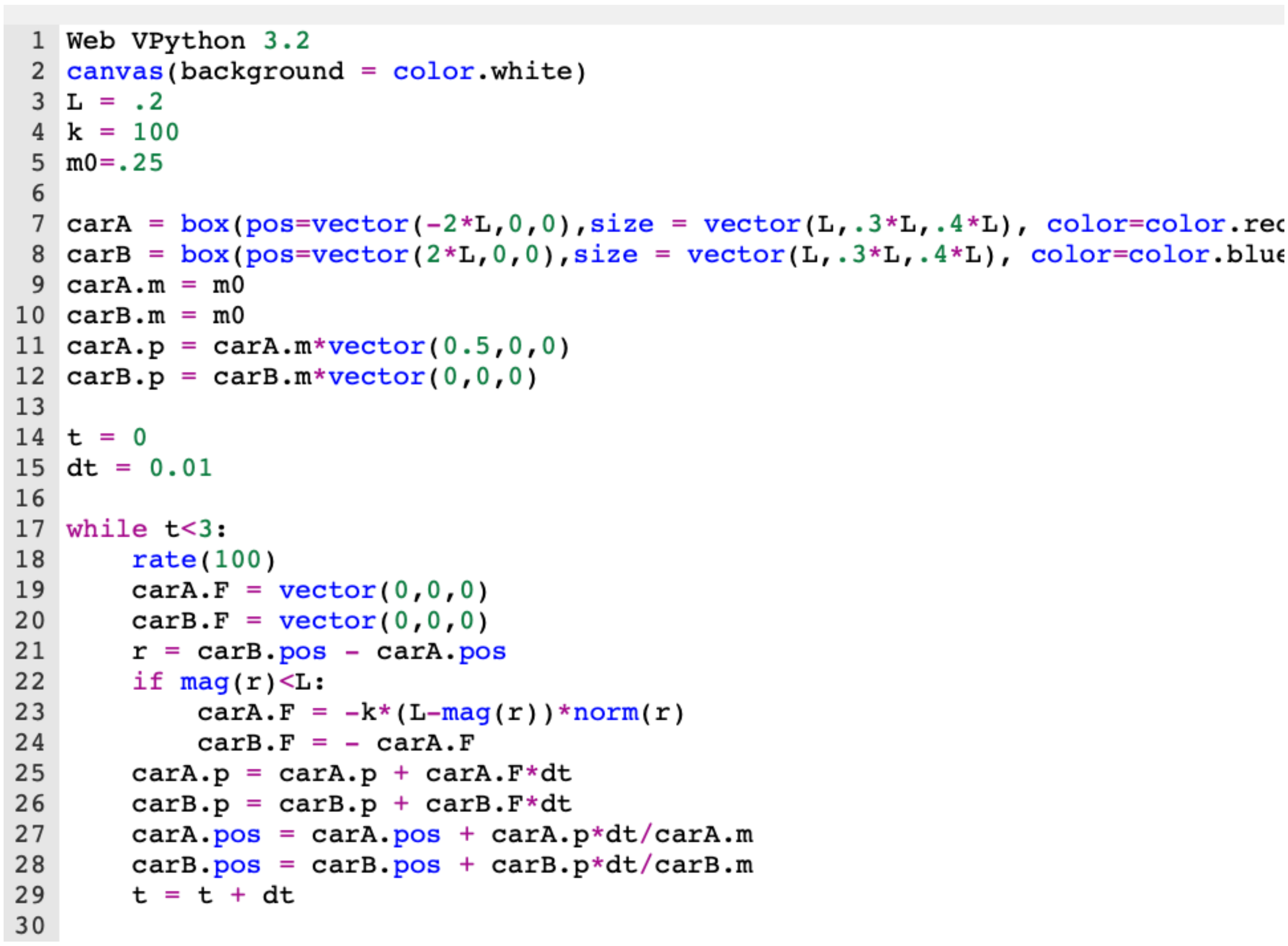
What happens when the distance between them is less than the value of L? We call that a collision. If we want to use our normal numerical methods, then we need the net force on each car. What if we assume that there's a spring force pushing the two cars apart when the objects overlap? That happens when $|\vec{r}| < L$.



If we determine the vector value (r) between the carts (from A to B), then the "impact" force will be:

$$\vec{F}_{B-A} = -k(L - |\vec{r}|)\hat{r} \qquad \vec{F}_{A-B} = k(L - |\vec{r}|)\hat{r}$$

Where k is the effective spring constant. Let's just do this in python. Here's the full code.



```
 1 Web VPython 3.2
 2 canvas(background = color.white)
 3 L = .2
 4 k = 100
 5 m0=.25
 6
 7 carA = box(pos=vector(-2*L,0,0),size = vector(L,.3*L,.4*L), color=color.re
 8 carB = box(pos=vector(2*L,0,0),size = vector(L,.3*L,.4*L), color=color.blue
 9 carA.m = m0
10 carB.m = m0
11 carA.p = carA.m*vector(0.5,0,0)
12 carB.p = carB.m*vector(0,0,0)
13
14 t = 0
15 dt = 0.01
16
17 while t<3:
18     rate(100)
19     carA.F = vector(0,0,0)
20     carB.F = vector(0,0,0)
21     r = carB.pos - carA.pos
22     if mag(r)<L:
23         carA.F = -k*(L-mag(r))*norm(r)
24         carB.F = - carA.F
25     carA.p = carA.p + carA.F*dt
26     carB.p = carB.p + carB.F*dt
27     carA.pos = carA.pos + carA.p*dt/carA.m
28     carB.pos = carB.pos + carB.p*dt/carB.m
29     t = t + dt
30
```

Comments:

- I don't have to comment on normal stuff like momentum and mass. Right?

- Have we used the `box()` object previously? I can't remember. Well, it should be straight forward. The `pos` is the vector position of the center of the box and `size` is a vector used for the three dimensions of the box. The only issue here is if you want the box on top of a track, then you need to make sure the track (which would also be a box) is not at the same y-position.

- Lines 19-20. At the start of the loop, I set the net force on each object equal to the zero vector. This would be the case if they are not currently in a collision.

- Line 21: Calculate the vector from the car A to car B.

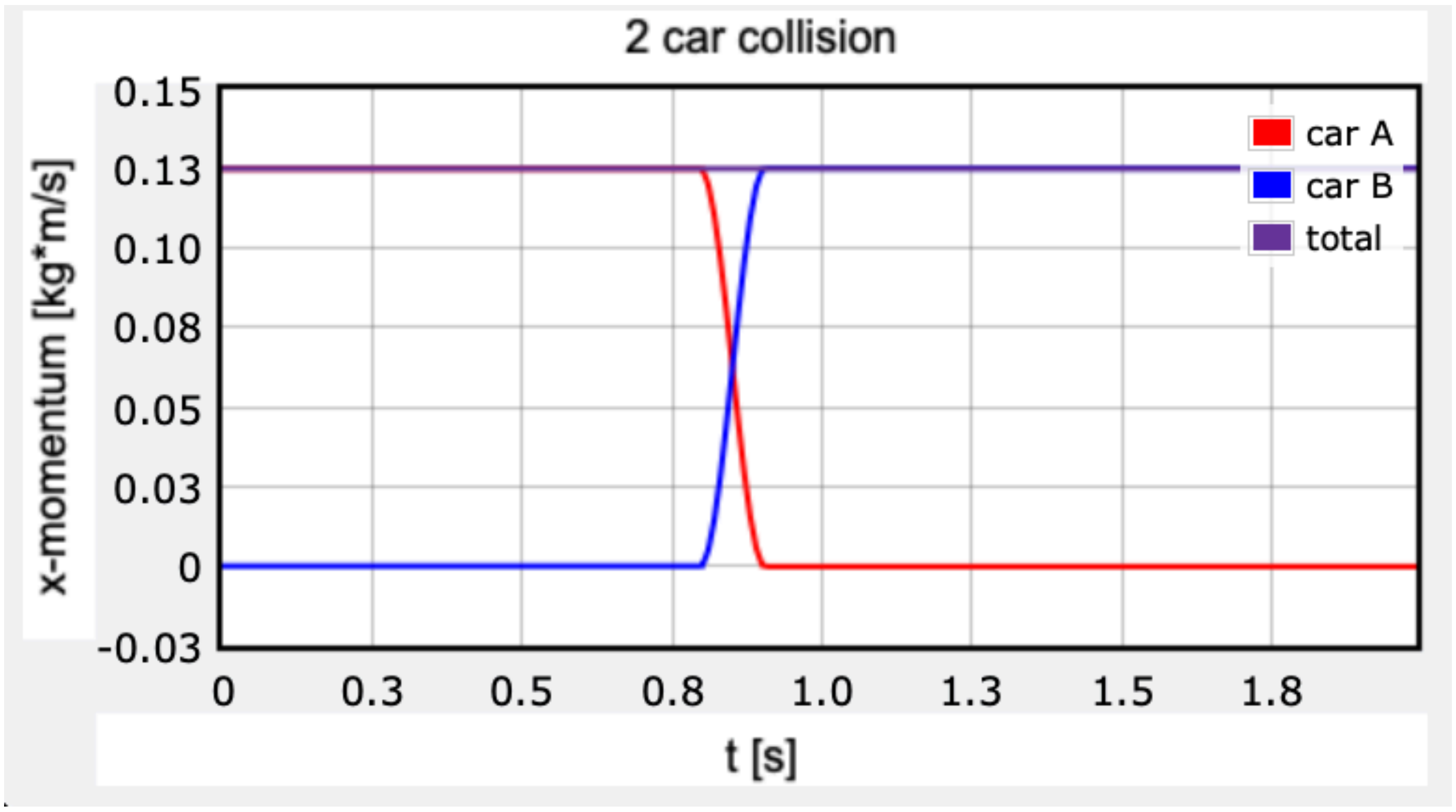
- Line 22-24: Check if there is a collision. If yes, then calculate the forces on the two cars. Notice that the force on A is the negative of the force on B as it should be.

- The rest is just normal stuff.

Here's a screenshot of the motion (but you can't see the animation—which is sad because it looks cool).

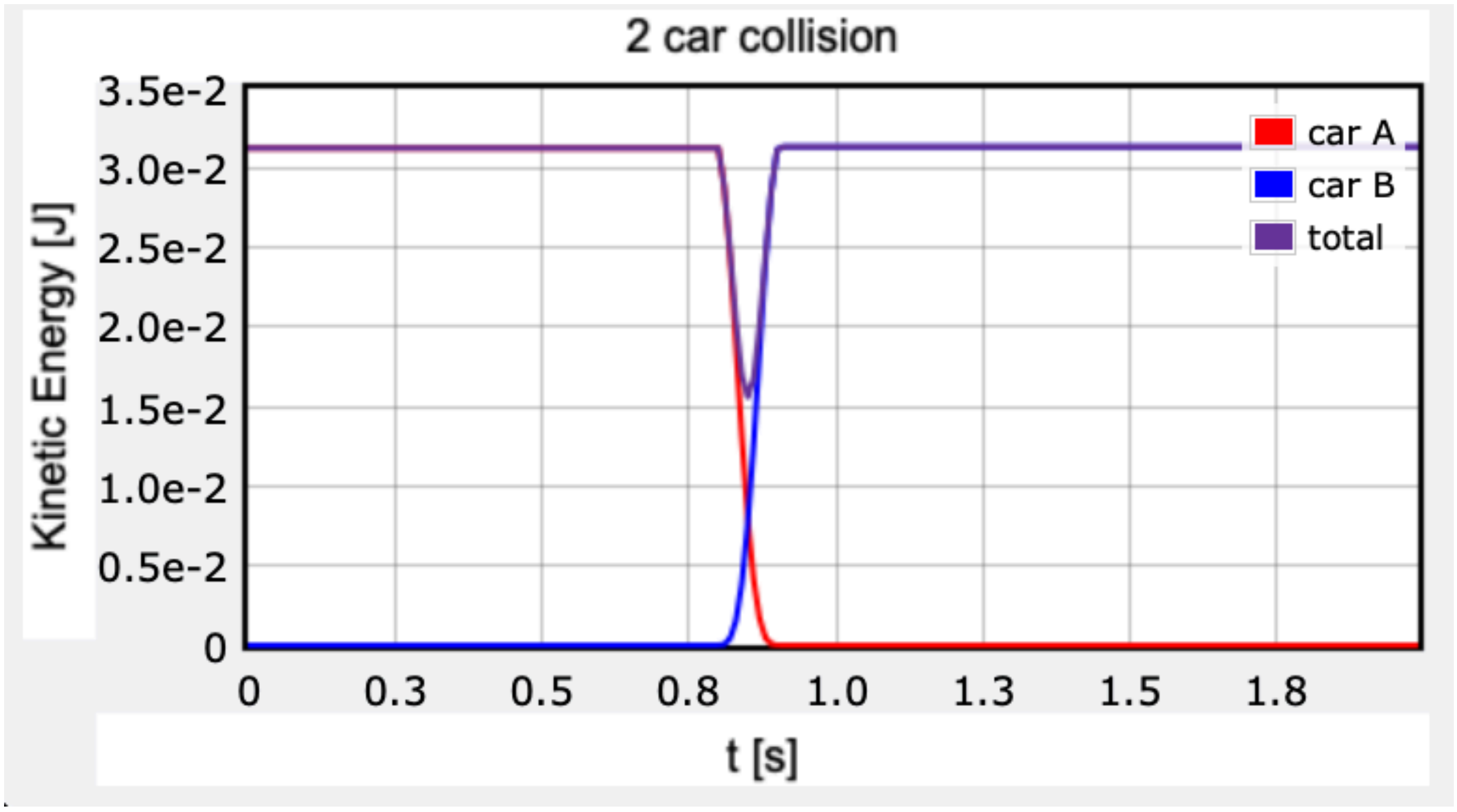
If you want to make it MORE awesome, you can add another box for the track. Also, it's not a bad idea to add some marker points below the cars (just spheres). This will help see how the two objects move as the camera zooms out to show the two cars.

Of course, the most important thing is a plot of the momentum as a function of time. That's pretty easy to add (you know how to add a graph—don't pretend like you don't). Here's that plot.



For a collision between cars of equal mass, the launched car stops and the target car then moves with that same speed. You can see that in the plot above as the momentum of car A goes to zero after the collision. Another cool little fact is that you can see the time interval of the collision. This happens as the momentum of car A is decreasing and car B is increasing.

Is kinetic energy conserved? Well, let's find out. Here's a plot.



Oh! It's NOT conserved. I mean, it sort of looks conserved, but notice that purple plot isn't just a straight line. It dips down during the collision. Yes, kinetic energy is not conserved during this time, but the total energy IS conserved. As the two cars overlap, the fake spring is compressed and there is an increase in elastic potential energy. I just think it's cool.

Homework

1. What happens if you make the "spring constant" much smaller? Hint, it's possible to have a collision where the two objects pass through each other.

2. What happens if the spring constant is super huge? Note: you might need to change the time interval to something smaller than 0.01 seconds.

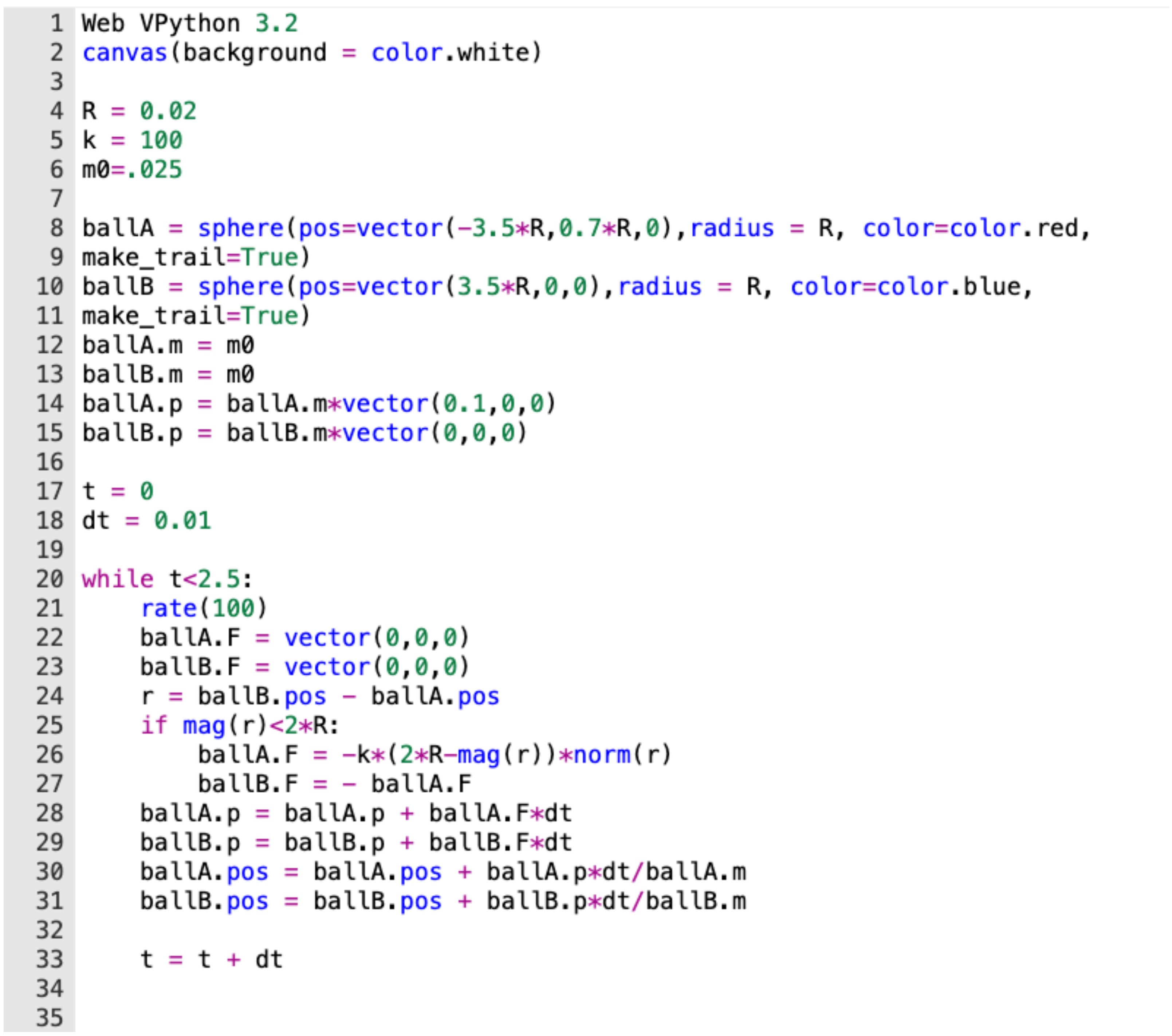
3. Change the mass and initial velocity of car B. Are momentum and kinetic energy still conserved?

4. Physics problem. Car A mass = 432 grams with an initial velocity of 0.5 m/s in the positive x-direction. This collides with car B which has a mass of 654 grams moving with a velocity of -0.2 m/s (in the negative x-direction). Use physics and calculate the final velocity for both cars. Check your results with a python model

2D ELASTIC COLLISION

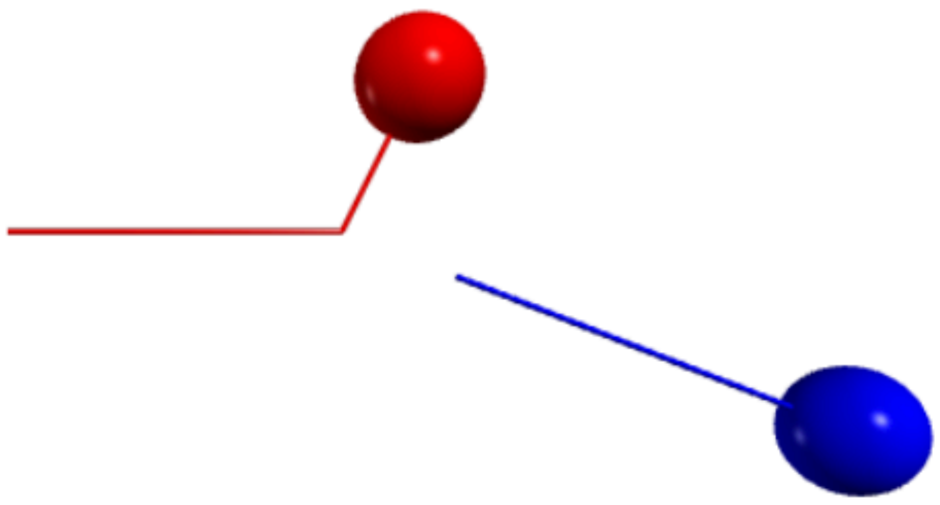
What if you change the two cars into balls? That's pretty simple—it really doesn't change anything except the way the animation looks. The physics is the same.

But wait! Try this code.



```
Web VPython 3.2
canvas(background = color.white)

R = 0.02
k = 100
m0=.025

ballA = sphere(pos=vector(-3.5*R,0.7*R,0),radius = R, color=color.red,
make_trail=True)
ballB = sphere(pos=vector(3.5*R,0,0),radius = R, color=color.blue,
make_trail=True)
ballA.m = m0
ballB.m = m0
ballA.p = ballA.m*vector(0.1,0,0)
ballB.p = ballB.m*vector(0,0,0)

t = 0
dt = 0.01

while t<2.5:
    rate(100)
    ballA.F = vector(0,0,0)
    ballB.F = vector(0,0,0)
    r = ballB.pos - ballA.pos
    if mag(r)<2*R:
        ballA.F = -k*(2*R-mag(r))*norm(r)
        ballB.F = - ballA.F
    ballA.p = ballA.p + ballA.F*dt
    ballB.p = ballB.p + ballB.F*dt
    ballA.pos = ballA.pos + ballA.p*dt/ballA.m
    ballB.pos = ballB.pos + ballB.p*dt/ballB.m

    t = t + dt


```

This is basically the same code as before with just a couple of changes. First, I changed the cars to balls (no big deal there). Second, the starting position of ballA is moved up (in the y-direction) slightly. Oh, also I added trails to the balls. But now we have an off-center collision. This is what happens.



Boom. There's your 2D elastic collision. Why does this work? Well, remember when we calculate the vector from ball A to ball B? It's a vector. It doesn't matter that the two balls aren't both on the x-axis. Now when we calculate the impact force, we have that $\hat{r}$ term in there to give the direction of the two forces. If $\vec{r}$ is at an angle, then the force is at an angle.

But is this actually a 2D elastic collision? Oh, that sounds like some homework.

Homework

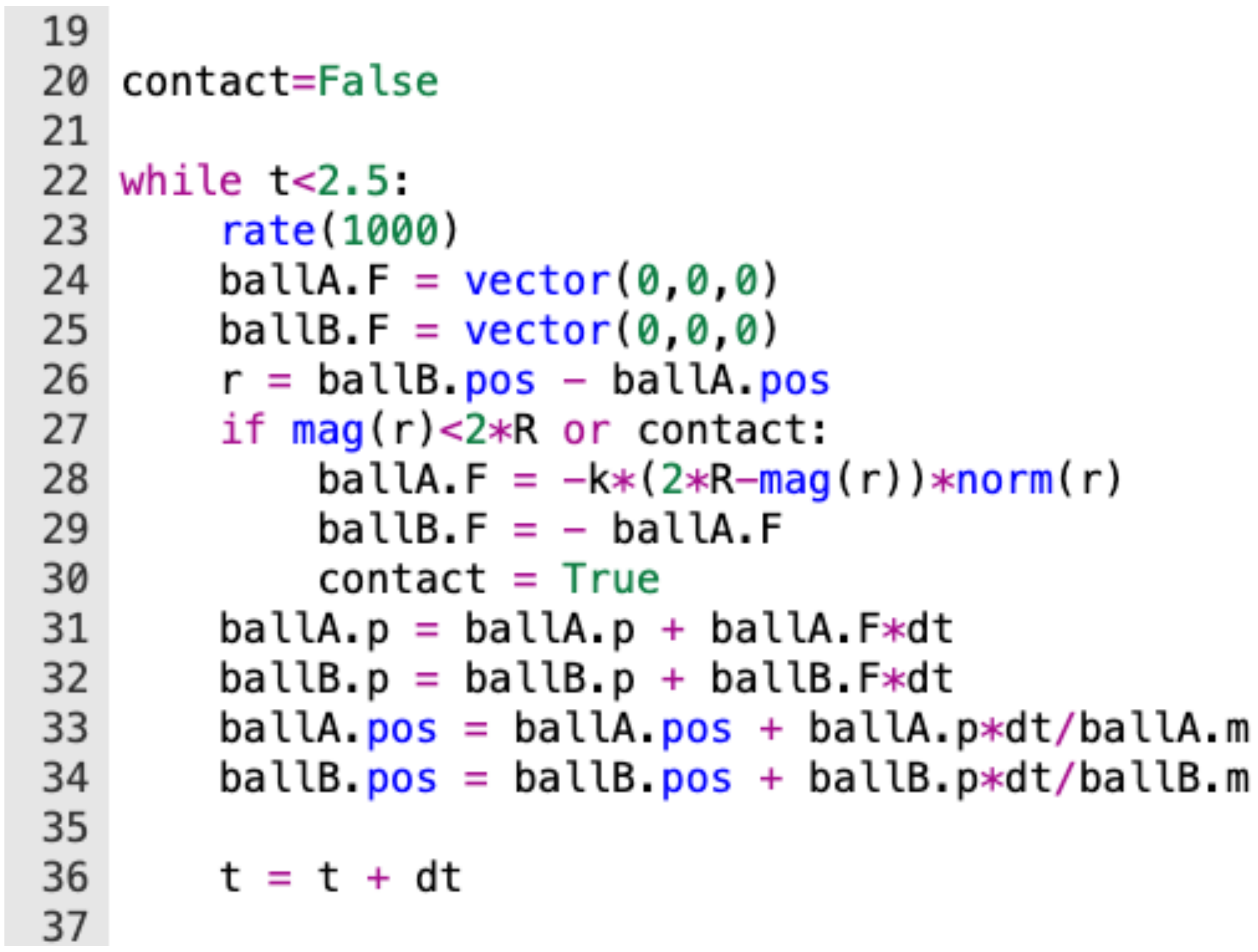
1. Is momentum conserved in both the x and y-directions? You might want to make some graphs.
2. Is kinetic energy conserved?
3. If this is a collision between equal mass object, the angle between their final velocity vectors should be 90 degrees. Is this true? Hint: $\vec{p}_A \cdot \vec{p}_B = |\vec{p}_A||\vec{p}_B|\cos\theta$

INELASTIC COLLISIONS

Now that we know the trick for 2D collisions, I don't need to start with 1D collisions—they are just the same as 2D (and 3D for that matter). But how do you make two objects "stick" together? Yes, we are going to use springs.

Here's how this will work. In the elastic collision, there was only an impact force if the two objects overlapped. Now we want that spring force to pull them back together so that they stay in contact.

We can do this with just a couple of simple changes to our previous code. Here's the part that changed.



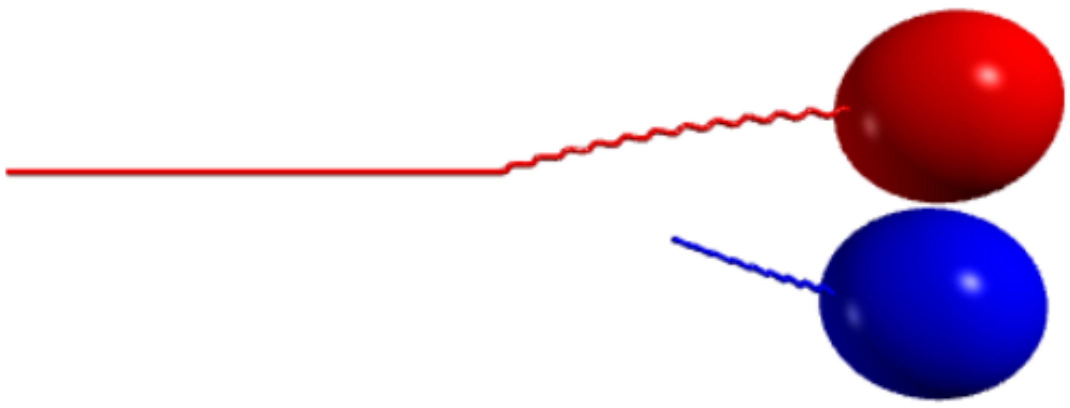
```
19
20 contact=False
21
22 while t<2.5:
23     rate(1000)
24     ballA.F = vector(0,0,0)
25     ballB.F = vector(0,0,0)
26     r = ballB.pos - ballA.pos
27     if mag(r)<2*R or contact:
28         ballA.F = -k*(2*R-mag(r))*norm(r)
29         ballB.F = - ballA.F
30         contact = True
31     ballA.p = ballA.p + ballA.F*dt
32     ballB.p = ballB.p + ballB.F*dt
33     ballA.pos = ballA.pos + ballA.p*dt/ballA.m
34     ballB.pos = ballB.pos + ballB.p*dt/ballB.m
35
36     t = t + dt
37
```

There are only three things that changed. Can you spot them? Don't worry, I'm going to tell you.

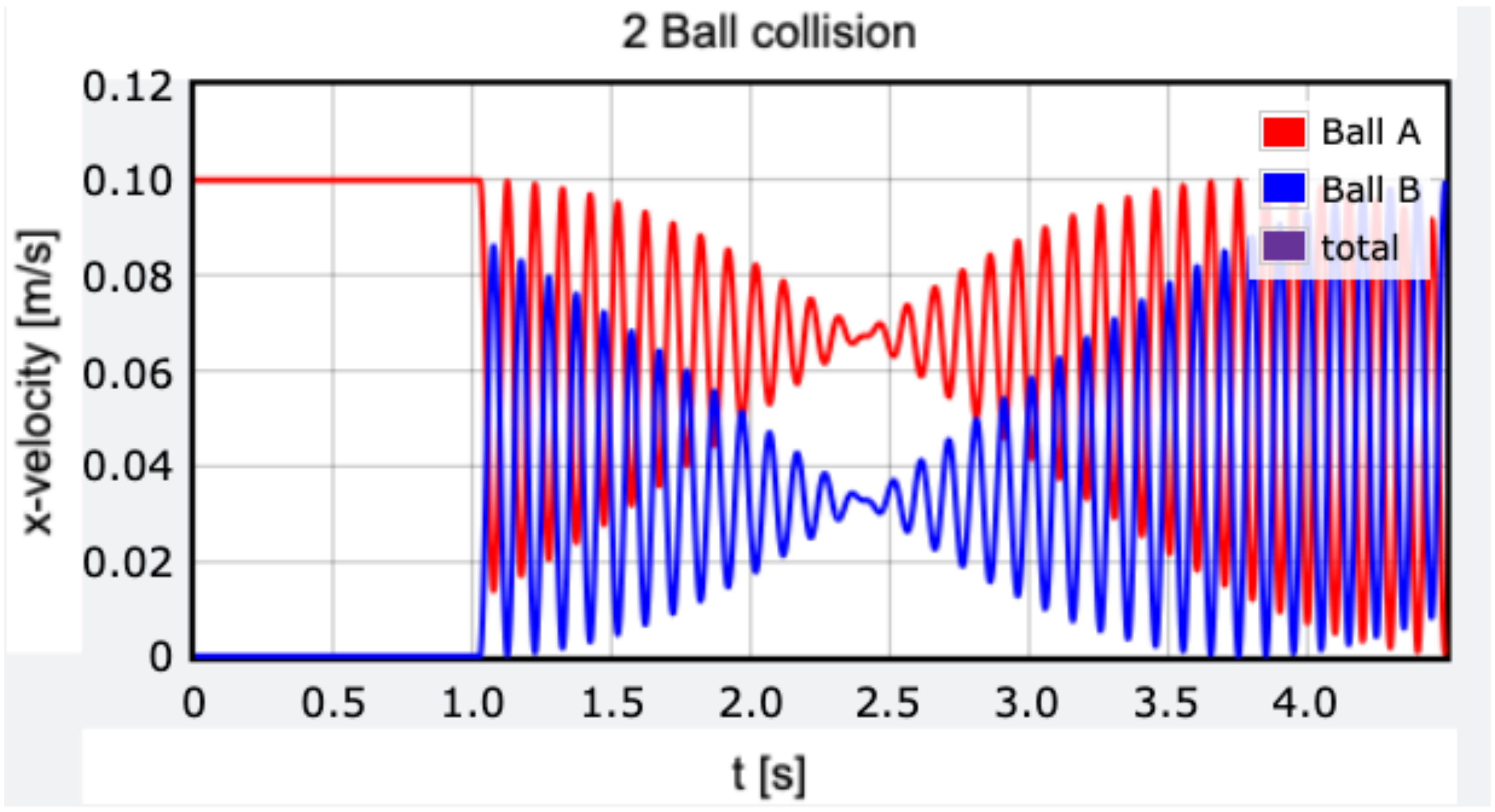
- There's a new variable. It's a boolean (meaning it can only be either `True` or `False`). It's in line 20 and I'm calling it `contact` and it's initially set to `False`.
- In line 27, I've changed the conditional statement for the collision. Now, we are going to calculate the impact force if either of the following two things are true: the balls overlap OR that contact variable is `True`. This means that after the object have collided and `contact = True`, there will still be a force between the balls. But since they will be farther apart than 2R, the force will pull them together rather than pull them apart.

- Finally, in line 30 inside the if loop the value of contact is set to `True`. Since this is past the if statement, it won't get changed to True until there's a collision.

Cool, right? Here's what a collision looks like. Oh, I made the spring constant smaller so that we can see something neat.

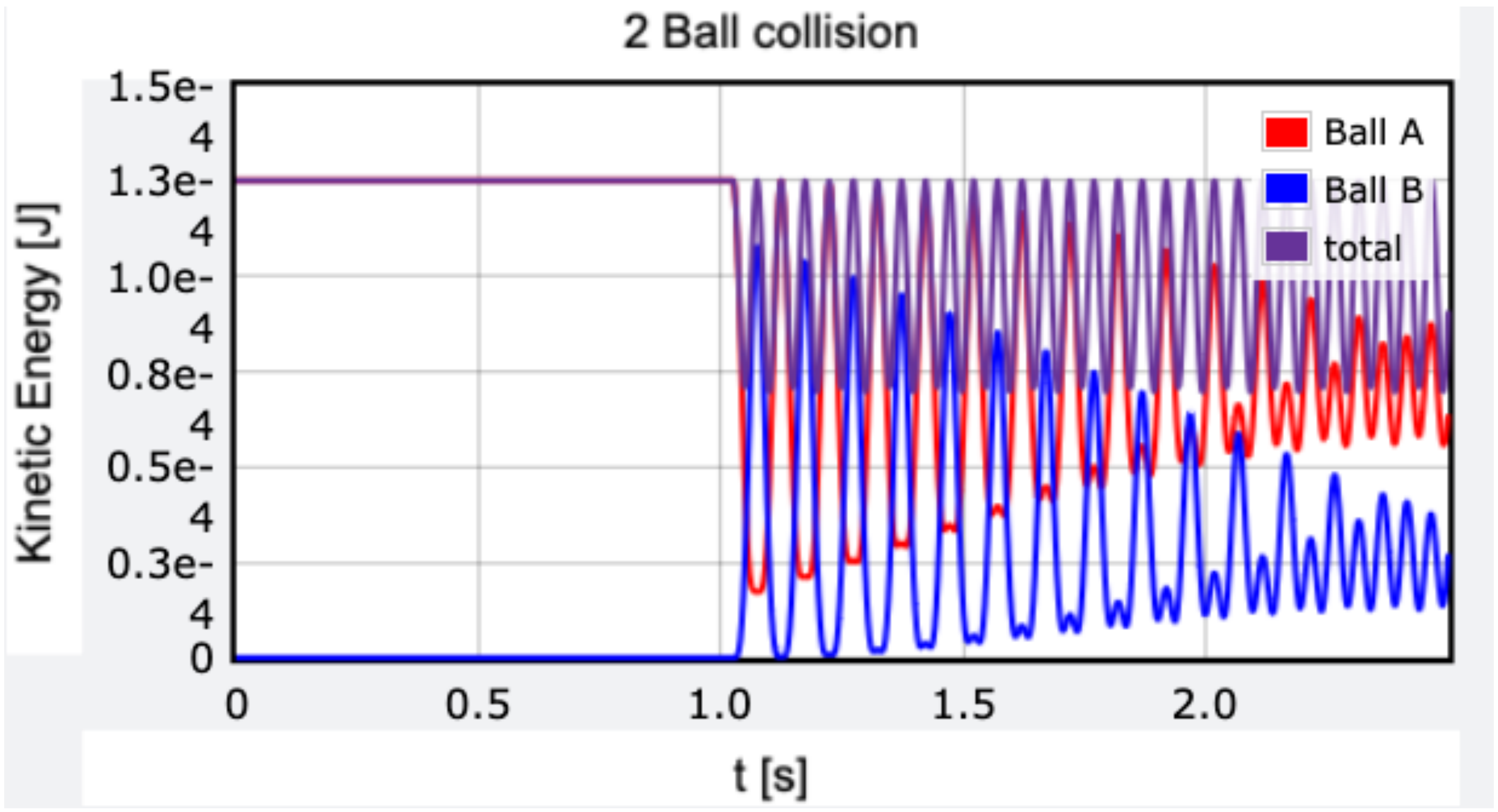


You can see the squiggles (not the Wiggles band) in the trails. That's because these two balls are shaking after they collided due to that spring force that pulls them back together. Do you know what else is awesome? This is the same initial conditions as the previous "glancing" collision. In the elastic version, the two balls went off in different directions but in this case they move together in the x-direction after the collision. Instead of plotting the momentum (which we know is conserved, let's plot the x-component of the velocity for these two balls.



They don't quite have the same x-velocity after the collision. That's because they are still colliding after impact. Not only are they shaking back and forth, but the the combination of the two balls is now rotating. This is way cooler than you think. We will get back to this rotation when we look at angular momentum (later).

But what about the kinetic energy? Here is a plot of the kinetic energy for this inelastic collision.



There's still that drop in the total kinetic energy and it oscillates as energy is stored in the spring. But it's obviously not an elastic collision now.

OK, maybe that's not good enough for you. You want THE BEST inelastic collisions. Well, we can make this better. Suppose I calculate the relative velocity between ball A and B.

$$\vec{v}_{\text{rel}} = \vec{v}_B - \vec{v}_A$$

I can then use that relative velocity to add a relative drag force between the two balls.

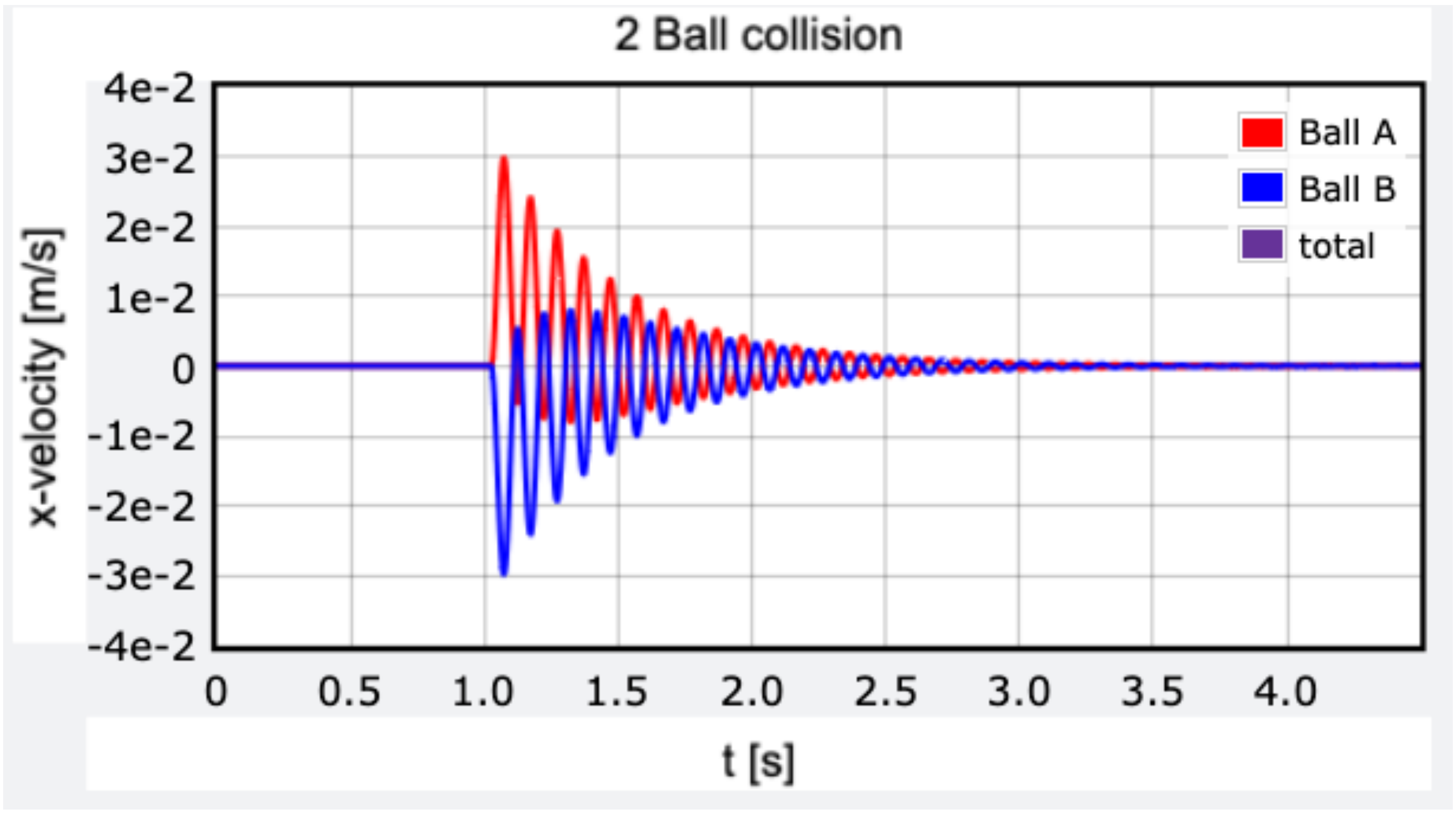
$$\vec{F}_{\text{rel-drag}} = -c\vec{v}_{\text{rel}}$$

This relative drag force will then act in the opposite direction of the relative motion to slow them down until the velocity of the two balls is the same. Oh wait, won't this destroy the conservation of momentum? Nope. As long as the force on A is equal and opposite to the force on B momentum will be conserved. I'm just going to pick a value for the drag coefficient at something like 0.05. Here's the added part of the code.

```
30      r = ballB.pos - ballA.pos
31      if mag(r)<2*R or contact:
32          vrel=ballB.p/ballB.m - ballA.p/ballA.m
33          ballA.F = -k*(2*R-mag(r))*norm(r)+c*vrel
34          ballB.F = - ballA.F
35          contact = True
36
```

- Line 32 calculates the relative velocity (using the momentums).
- Line 33 adds this relative drag force to the net force for ball A. Notice that this drag force is in the "if" loop so that there is only a drag force after the collision.
- Ball B is just the opposite force on A.

Just to show you that this works, here's a plot of the velocities of the two balls in the x-direction.



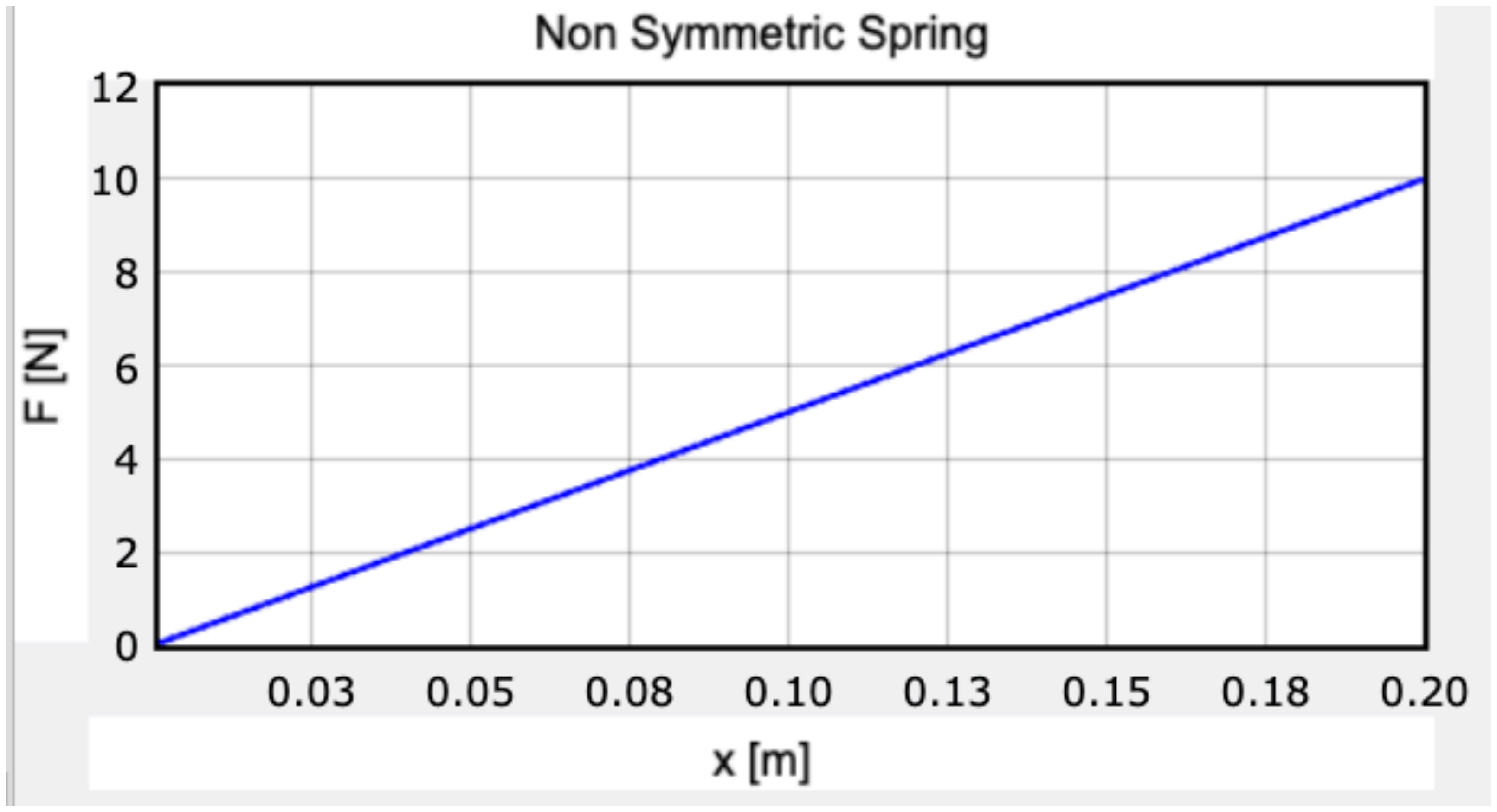
I like it. The two balls end up with the same final velocity. That's exactly what we want from an inelastic collision.

PLAIN OLD COLLISIONS

So, we can model the collision between two bouncy balls and we can model two sticky balls. What about normal plain balls that aren't bouncy or sticky? You know—like a real ball collision?

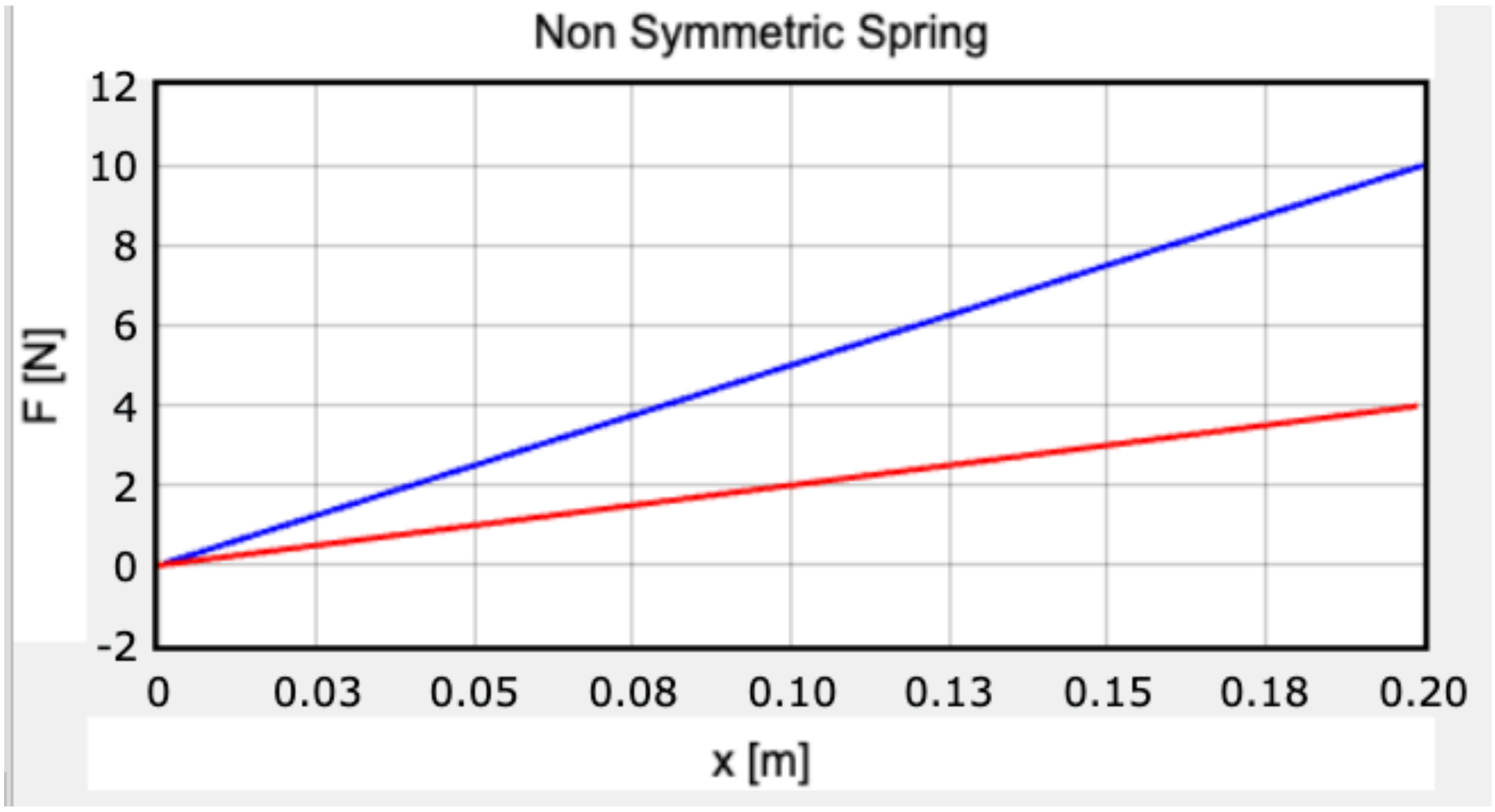
Let's just think. The springs are nice for collisions, but they conserve kinetic energy. If I use the pulling springs like before then the balls will just stick together. I need some type of interaction force that conserves momentum but not kinetic energy.

Here's the trick. Non-symmetrical springs. Suppose I have a spring with a spring constant k_1 when it's compressed. Here is a plot of the force vs. position for a ball running into this spring.



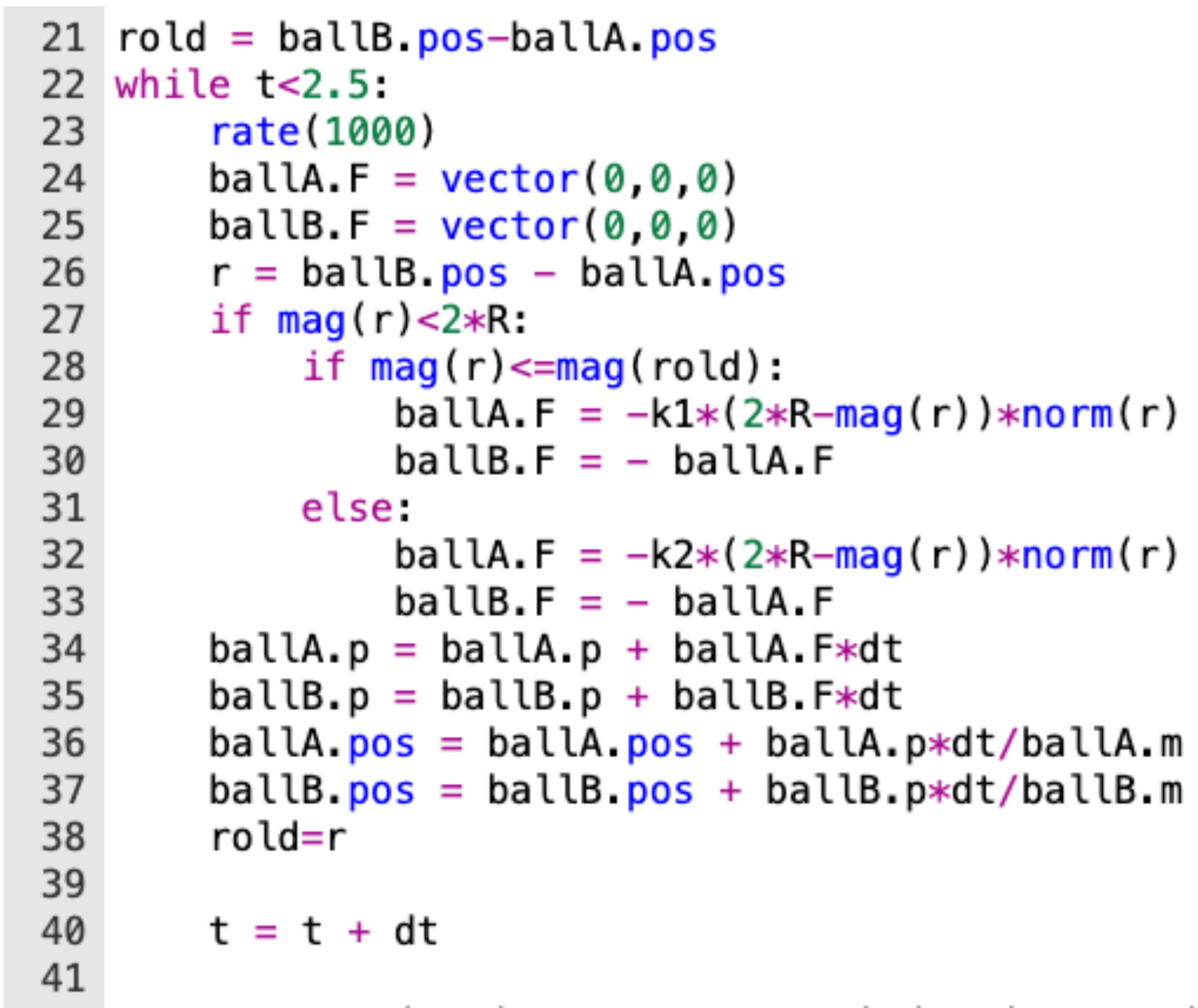
This shows a ball moving in the negative x direction (starting at 0.2 meters). Since the force is in the positive direction and the displacement in the negative direction, this force does negative work. That work is the area under this curve.

Now, as the spring rebounds, it has a smaller spring constant (k_2). We can again look at the work done by this spring.



Now the force and the displacement are in the same direction so that the spring does positive work. However, with a lower spring constant, the area under the curve has a lower value than during compression so that the object will rebound with a lower velocity. This is the key to plain collisions.

How do we add this into our numerical collision model? Check this out.

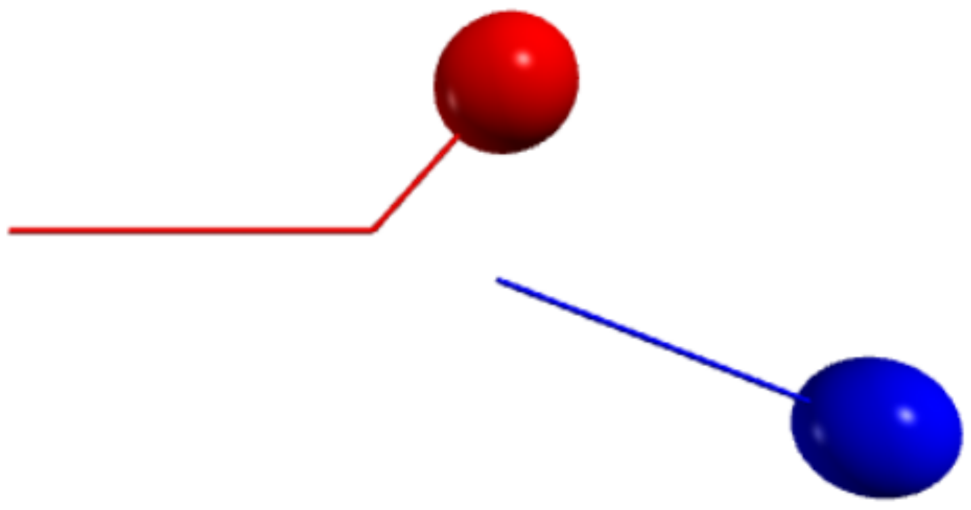


```
rold = ballB.pos-ballA.pos
while t<2.5:
    rate(1000)
    ballA.F = vector(0,0,0)
    ballB.F = vector(0,0,0)
    r = ballB.pos - ballA.pos
    if mag(r)<2*R:
        if mag(r)<=mag(rold):
            ballA.F = -k1*(2*R-mag(r))*norm(r)
            ballB.F = - ballA.F
        else:
            ballA.F = -k2*(2*R-mag(r))*norm(r)
            ballB.F = - ballA.F
    ballA.p = ballA.p + ballA.F*dt
    ballB.p = ballB.p + ballB.F*dt
    ballA.pos = ballA.pos + ballA.p*dt/ballA.m
    ballB.pos = ballB.pos + ballB.p*dt/ballB.m
    rold=r

    t = t + dt

```

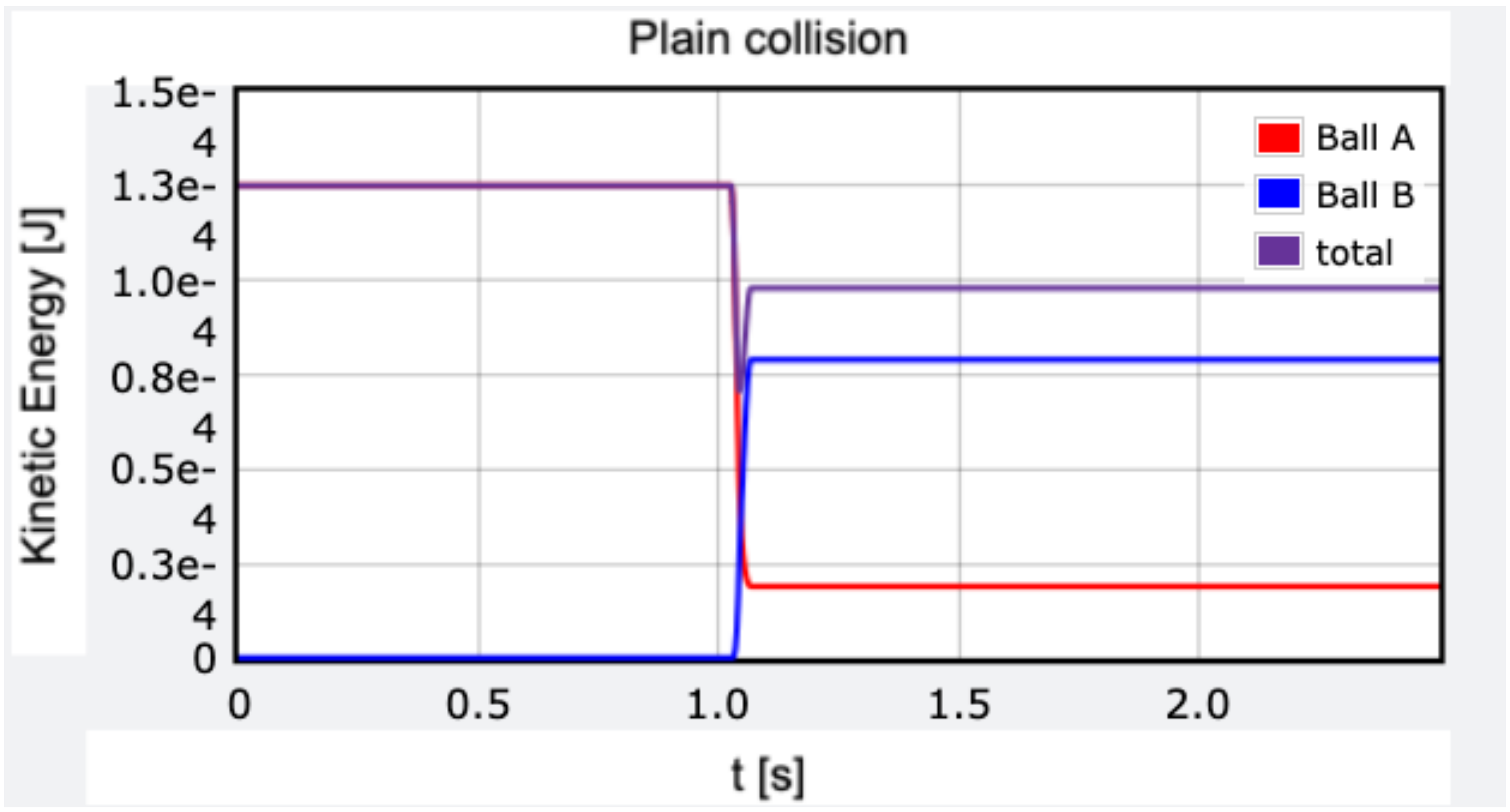
Comments.

- Line 21: The key here is to know when the objects are moving TOWARD each other and when they are moving AWAY. One way to do this is to introduce a new variable—`rold`. This is sort of a place holder. When the vector from one object to the other is calculated, we can compare it to the old distance. If the new value is greater than the old value, then the objects are moving away from each other.
- Line 28: Here we have another conditional statement to see if the balls are moving towards each other. If they are, then we calculate the force using the spring constant k1. Otherwise, we use the constant k2 (smaller value).
- Line 38: Don't forget to update the previous value of `rold` to the new value (so we can use it again).

But does it work? Here is what a 2D collision looks like.



This might look the same as the 2D elastic collision, but the angle between the final ball velocities is NOT 90 degrees. It's different. Is momentum conserved? Yup. Remember, momentum is conserved as long as the forces on the objects are equal and opposite (they are). Is kinetic energy conserved? Let's check. Here's a plot.



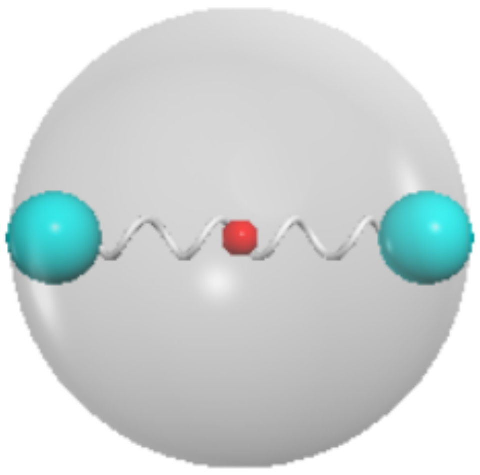
That's a win.

HOW CAN A COLLISION CONSERVE MOMENTUM AND NOT KINETIC ENERGY?

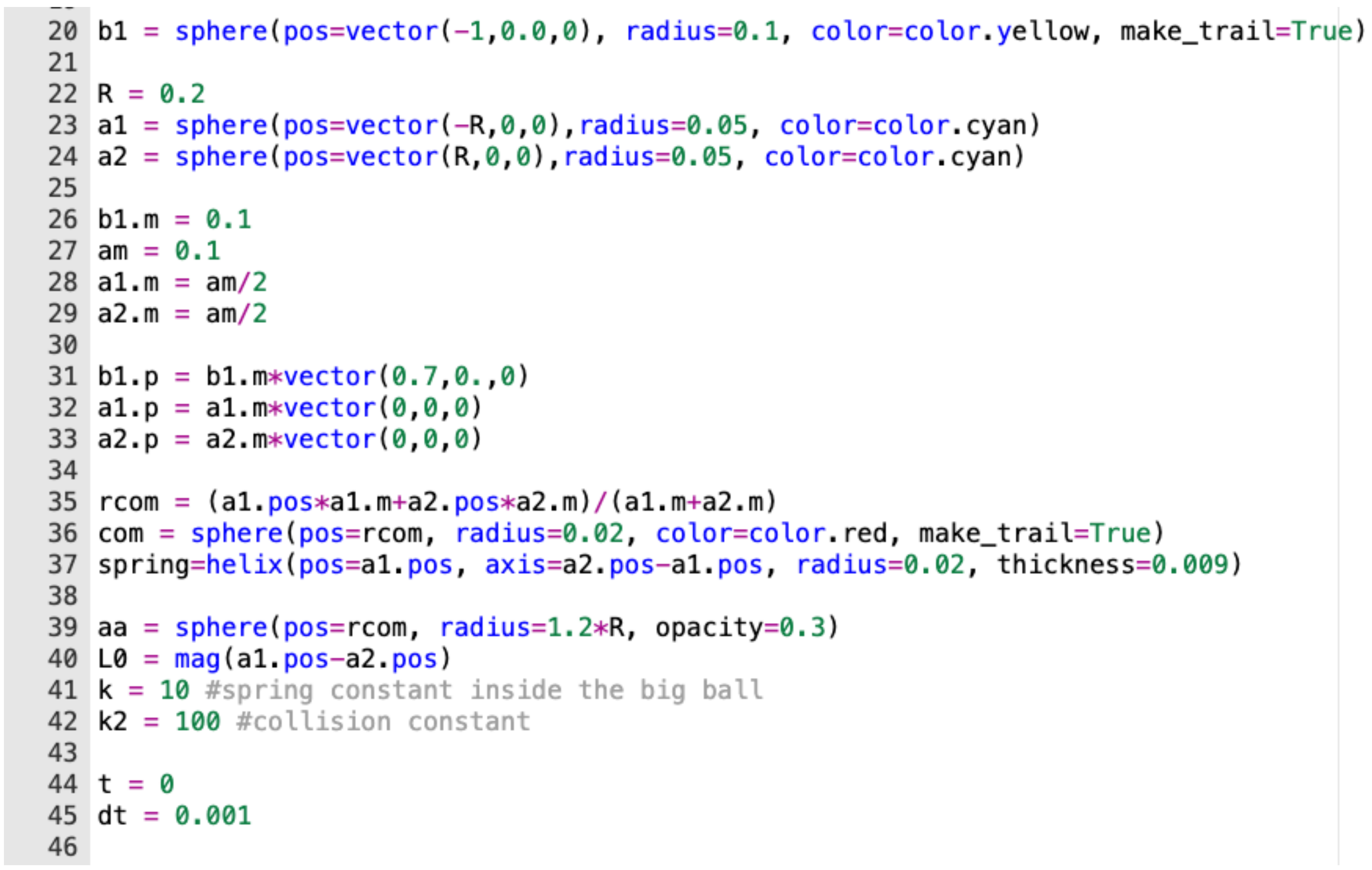
This is a great question. I mean, both the momentum and the kinetic energy depend on the mass and velocity. So it sort of seems like if momentum is conserved then the kinetic energy MUST also be conserved. It makes sense—even if it's not actually true.

Of course we already did this problem—but maybe it could be considered cheating. Let's build a cooler collision model that really shows what's going on with an non-elastic (plain) collision.

I hope you aren't surprised, but I'm going to use springs. But this won't be the same model that we had before, it's going to have even MORE springs. I'm going to replace the target ball with a ball that has two masses connected by springs. It's going to look like this (then I will show you how to make it).



This ball is made of two masses connected by a spring. Just for effect, I'm also showing the center of mass (the red sphere) and the structure of the ball (the clear outer shell). There's a bunch of work here to set this up, so let me just jump to the code.



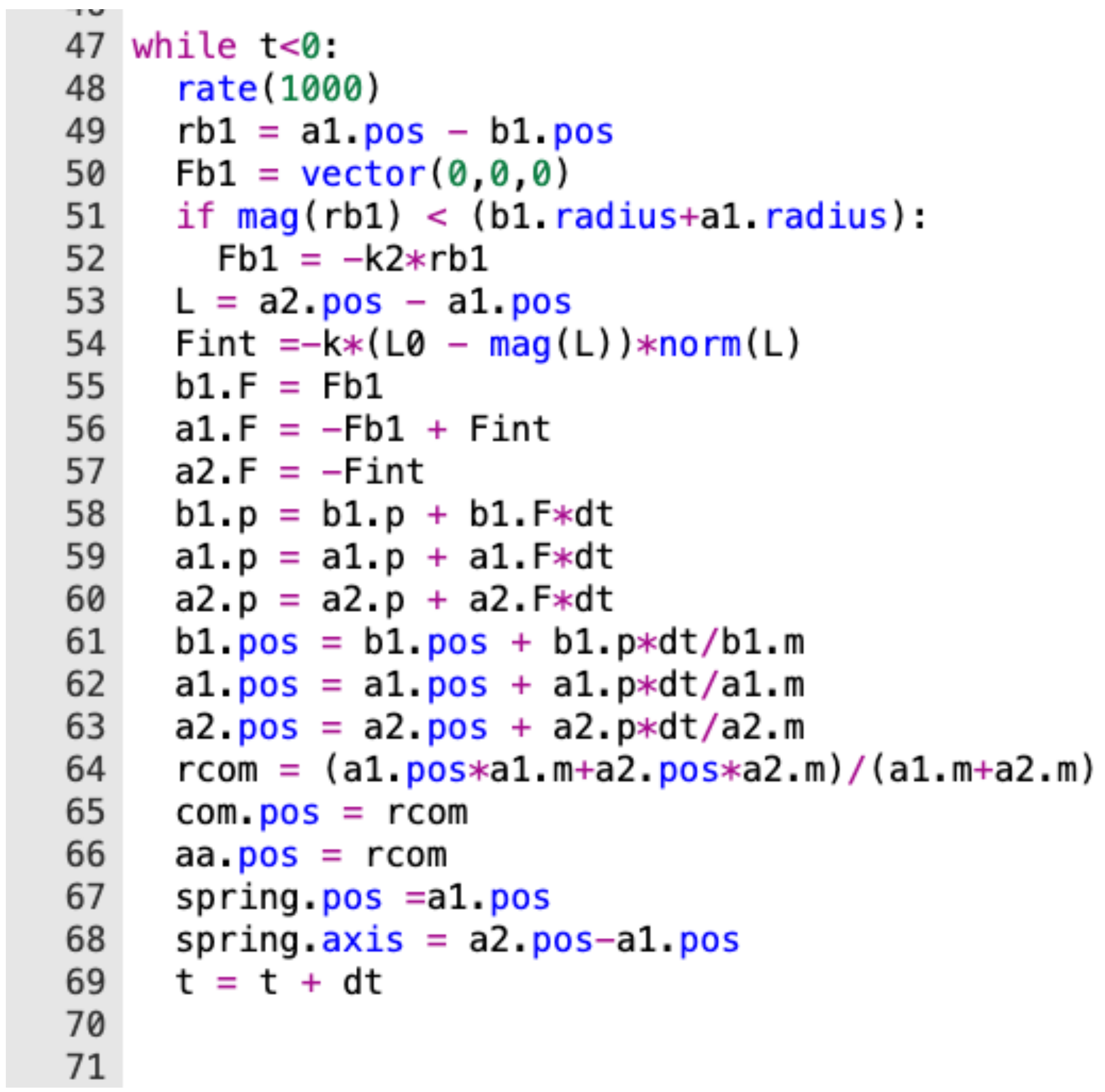
```
b1 = sphere(pos=vector(-1,0.0,0), radius=0.1, color=color.yellow, make_trail=True)

R = 0.2
a1 = sphere(pos=vector(-R,0,0),radius=0.05, color=color.cyan)
a2 = sphere(pos=vector(R,0,0),radius=0.05, color=color.cyan)

b1.m = 0.1
am = 0.1
a1.m = am/2
a2.m = am/2

b1.p = b1.m*vector(0.7,0.,0)
a1.p = a1.m*vector(0,0,0)
a2.p = a2.m*vector(0,0,0)

rcom = (a1.pos*a1.m+a2.pos*a2.m)/(a1.m+a2.m)
com = sphere(pos=rcom, radius=0.02, color=color.red, make_trail=True)
spring=helix(pos=a1.pos, axis=a2.pos-a1.pos, radius=0.02, thickness=0.009)

aa = sphere(pos=rcom, radius=1.2*R, opacity=0.3)
L0 = mag(a1.pos-a2.pos)
k = 10 #spring constant inside the big ball
k2 = 100 #collision constant

t = 0
dt = 0.001

```

Comments:

- There are three objects that interact. There the launched ball (b1) and then the two balls inside the other ball a1 and a2. Of course they have masses and stuff. The other things are the center of mass (com), the shell (aa) and the spring (spring).

- The internal spring has a spring constant (k) and unstretched length (L0). There is also a spring constant for the interaction between the three physical balls (k2).

Now for the motion.

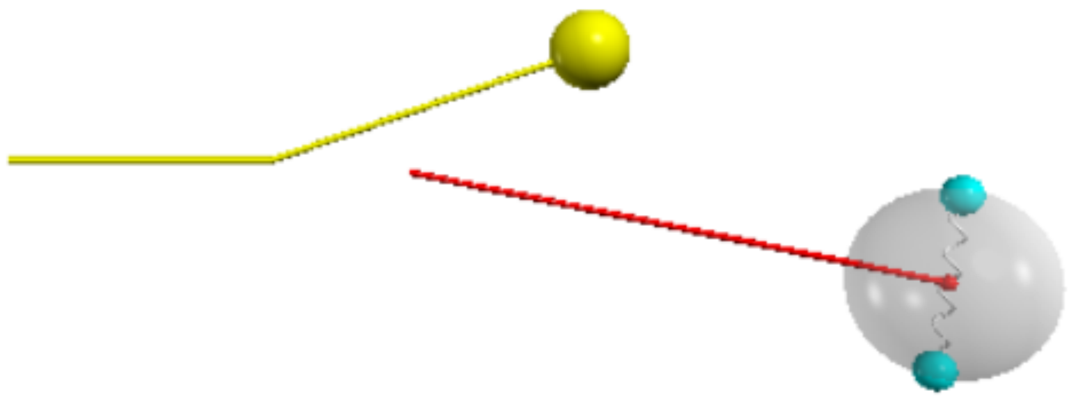


```
47 while t<0:
48   rate(1000)
49   rb1 = a1.pos - b1.pos
50   Fb1 = vector(0,0,0)
51   if mag(rb1) < (b1.radius+a1.radius):
52     Fb1 = -k2*rb1
53   L = a2.pos - a1.pos
54   Fint =-k*(L0 - mag(L))*norm(L)
55   b1.F = Fb1
56   a1.F = -Fb1 + Fint
57   a2.F = -Fint
58   b1.p = b1.p + b1.F*dt
59   a1.p = a1.p + a1.F*dt
60   a2.p = a2.p + a2.F*dt
61   b1.pos = b1.pos + b1.p*dt/b1.m
62   a1.pos = a1.pos + a1.p*dt/a1.m
63   a2.pos = a2.pos + a2.p*dt/a2.m
64   rcom = (a1.pos*a1.m+a2.pos*a2.m)/(a1.m+a2.m)
65   com.pos = rcom
66   aa.pos = rcom
67   spring.pos =a1.pos
68   spring.axis = a2.pos-a1.pos
69   t = t + dt
70
71
```

This is really just a combination of the collision code and the two masses with springs. Let me point out some of the important stuff.

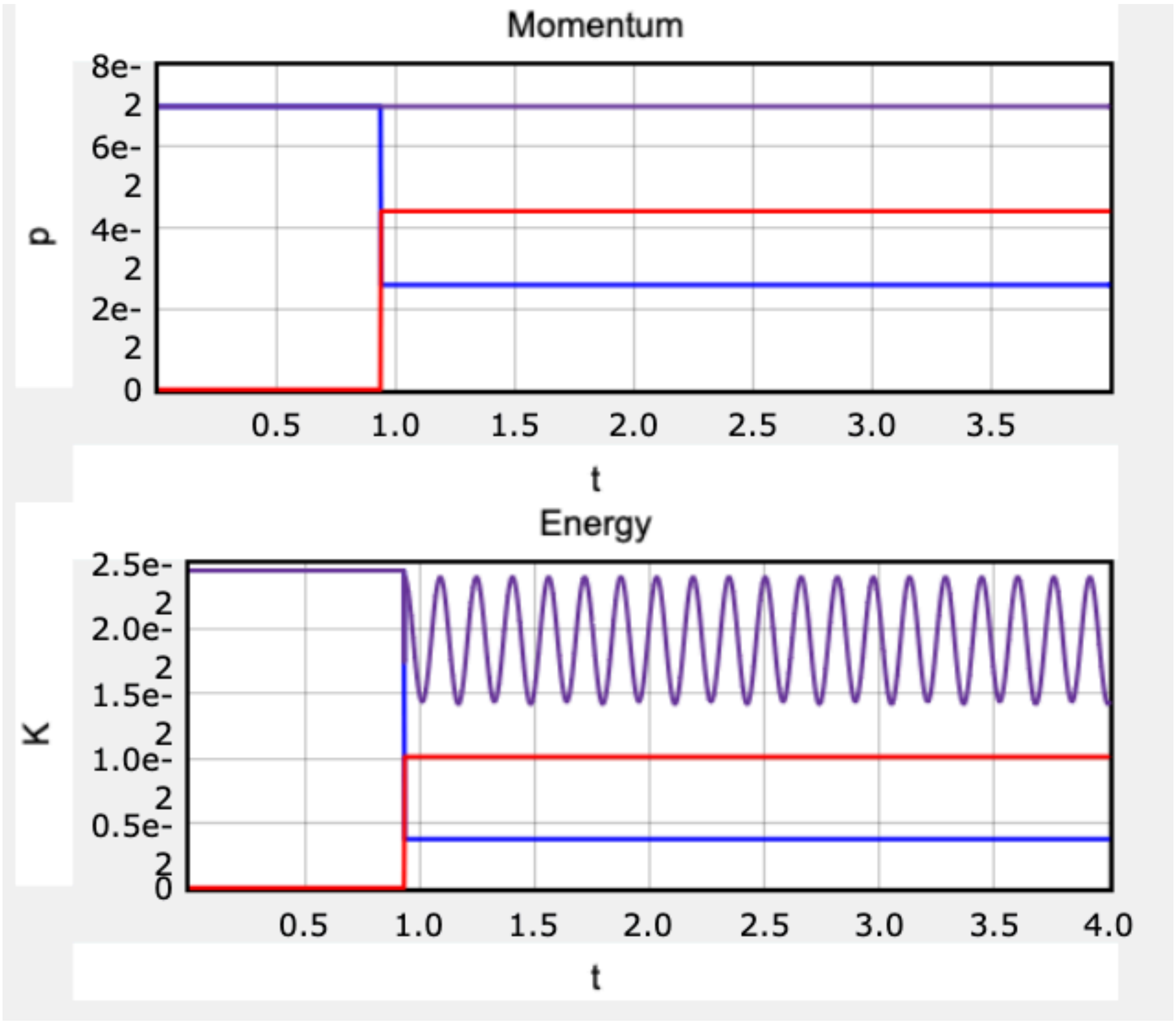
- Line 49 is for the detection of the collision between the launched ball and the leftmost target (interior) ball. Technically, you should check for a collision with the other ball but I just wanted to keep it as simple as possible.
- Lines 53-54: This calculates the force for the internal spring.
- The rest should be clear unless you skipped all the previous chapters in this book.

Let's do something with this now. Here is a glancing collision.



I wish you could see this as an animation—well, I guess you can if you enter the code yourself and run it. You need to do that. The spring ball bounces off at an angle as you can see if the image above. But what you can't see is that after the collision, the interior balls are still oscillating. That's where the lost kinetic energy goes—it goes into some type of internal energy of the objects. In this case, that internal energy is a combination of relative motion and spring potential energy.

Here's a check on the momentum (of the center of mass for the target) and the kinetic energy.


Momentum
p
8e-2
6e-2
4e-2
2e-2
0
0.5 1.0 1.5 2.0 2.5 3.0 3.5
t
Energy
K
2.5e-2
2.0e-2
1.5e-2
1.0e-2
0.5e-2
0
0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0
t