Space Voyage

1 Objectives

In this program you will model the motion of a spacecraft traveling around the Earth. To do this you will iteratively (repeatedly) :

- calculate gravitational force on the spacecraft by the Earth,
- apply the momentum principle to update the momentum of the spacecraft,
- apply the position update formula to update the location of the spacecraft,
- explore the effect of the spacecraft's initial velocity on its trajectory, and
- study the effect of your choice of Δt (`deltat`) on the accuracy of your predictions.

2 Data

Useful data for writing your program is contained in the following table.

Mass of Earth: 6×10^{24} kg	Radius of Earth: 6.4×10^{6} m
Mass of spacecraft: 15×10^{3} kg	Radius of spacecraft: very small
Mass of Moon: 7×10^{22} kg	Radius of Moon: 1.75×10^{6} m
Distance from Earth to Moon: 4×10^{8} m	G = 6.7×10^{-11} N m^2/kg^2

3 Program Shell

A program shell (a skeleton program) has been provided. Read through the program shell and make sure you understand the function of each VPython instruction. **Answer the following questions on a piece of paper before proceeding.**

- Given the initial location and momentum of the spacecraft, what should the path of the spacecraft be?
- How many years will the loop run?

3.1 Iterative Calculations

In order to get the spacecraft to move, you must calculate the gravitational force, update momentum, and update position. Because this must be done after each time step, you need to put these calculations inside the loop.

Use the symbolic names defined in the program shell. You may need to add other symbolic names in your calculations.

In your calculations, for readability and ease of making changes, use the position of the Earth in the form Earth.pos rather than assuming this position never changes. This makes it easier to modify the program

should you wish to let the Earth move.

Do this, and run your program. Is the behavior what you expected?

Questions:
How you can determine how accurate your program is? Try making `deltat` 10 times larger. Make it 10 times smaller. What happens? Why?

3.2 Detecting a Collision

If your spacecraft collides with the Earth, the program should stop. Add code similar to this inside your loop (using the name you defined for the distance between the spacecraft and the center of the Earth):

```
if rmag < Earth.radius:
    break
```

This code tells VPython to get out of the loop if the spacecraft touches the Earth.

3.3 Answer these question on a piece of paper

- Give the spacecraft a speed of 0.6 km/s (0.6e3 m/s), headed in the +y direction. What happens?
- Increase the initial momentum slightly until the spacecraft just misses the Earth. Approximately (to 3 significant figures), what is the minimum initial speed required to miss the Earth? What does the orbit shape look like?
- Approximately, what initial speed is required to make a circular orbit around the Earth? You may wish to zoom out to examine the orbit more closely.
- Approximately, what minimum initial speed is required so that the spacecraft "escapes" and never comes back? You may have to zoom out to see whether the spacecraft shows signs of coming back. You may also have to extend the time in the while statement.

4 Important Kinematic and Dynamic Quantities

In this section you will add code to make arrows that represent three quantities: the spacecraft's momentum, the net force acting on the spacecraft, and the change in the spacecraft's momentum.

You will also print two quantities: the spacecraft's final position and final velocity.

4.1 Visualizing the Momentum Vector

You will create an `arrow` before the `while` loop and update its position and axis inside the loop, as the spacecraft's momentum changes. You need to know the approximate magnitude of the momentum in order to be able to scale the arrow to fit into the scene, so just before the loop print the momentum vector:

```
print('p=', pcraft)
```

As an example, you know that the distance between the center of the Earth and the Spacecraft is about 10 Earth radii, or 10*6e6 m = 6e7 m. With your fingers, estimate about how long you'd like the momentum arrow to be in the scene.

How long is that distance in the units of your virtual world? What scalar factor would you need to multiply the momentum by to change its magnitude to the number you want? That is the value of your scale factor.

In the `constants` section of your program, add this scale factor

```
pscale = ??
```

Also insert the following statement in the `objects` section of your program (NOT inside the loop):

```
parr = arrow(color=color.green)
```

This statement creates an arrow object with default position and axis attributes. You will set these attributes inside the loop.

- Comment out any print statements which are inside your loop because they slow down plotting.
- Inside your loop, update the `pos` attribute of the `parr` arrow object to be at the center of the spacecraft, and update its axis attribute to represent the current vector value of the momentum of the spacecraft (multiplied by `pscale`).
- Run the program with an initial velocity that produces an elliptical orbit.
- You may have to adjust the scale factor once you have seen the full orbit.

4.2 Visualizing the Net Force Vector

Now, you will create an `arrow` before the `while` loop to represent the net force that acts on the spacecraft. You need to know the approximate magnitude of the force in order to be able to scale the arrow to fit into the scene, so just after the loop print the force vector:

```
print('Fnet=', Fnet)
```

As you did in section 4.1, you know that the distance between the center of the Earth and the Spacecraft is about 10 Earth radii, or 10*6e6 m = 6e7 m. With your fingers, estimate about how long you'd like the force arrow to be in the scene.

In the `constants` section of your program, add this scale factor

```
fscale = ??
```

Also insert the following statement in the `objects` section of your program (NOT inside the loop):

```
farr = arrow(color=color.cyan)
```

This statement creates an arrow object with default position and axis attributes. You will set these attributes inside the loop.

- Comment out any print statements which are inside your loop because they slow down plotting.

- Inside your loop, update the `pos` attribute of the `farr` arrow object to be at the center of the spacecraft, and update its axis attribute to represent the current vector value of the net force acting on the spacecraft (multiplied by `fscale`).
- Run the program with an initial velocity that produces an elliptical orbit.
- You may have to adjust the scale factor once you have seen the full orbit.

4.3 Visualizing the Change in Momentum

Creating an `arrow` that represents the change in momentum, $\Delta\vec{p}$, involves a little more work. You will need to compute `deltap` in your loop, each time the loop executes.

- Before you update the momentum in the loop, add a line that stores the current value of the momentum (e.g., `pcraft_i = pcraft + vector(0,0,0)`).
- Adding the zero vector is necessary because of the way VPython uses "assignment".
- After the momentum update, take the vector difference between the updated momentum and the old momentum (e.g., `deltap = pcraft - pcraft_i`).

In the `constants` section of your program, add a scale factor. Will the scale factor be smaller or larger than `pscale`?

```
dpscale = ??
```

Also insert the following statement in the `objects` section of your program (NOT inside the loop):

```
dparr = arrow(color=color.red)
```

This statement creates an arrow object with default position and axis attributes. You will set these attributes inside the loop.

- Comment out any print statements which are inside your loop because they slow down plotting.
- Inside your loop, update the `pos` attribute of the `dparr` arrow object to be at the center of the spacecraft, and update its axis attribute to represent the current vector value of `deltap` (multiplied by `dpscale`).
- Run the program with an initial velocity that produces an elliptical orbit.
- You may have to adjust the scale factor once you have seen the full orbit.

Questions:
Can you see all the arrows? Adjust your scale factors so you can see all your arrows at the same time.

4.4 Print the position and velocity of the spacecraft

After your program is done running you want to print the final position and velocity of the spacecraft.

- Add the code, after your physics loop, that prints the final position and velocity of your spacecraft.

4.5 Answer these questions on a piece of paper

- For this elliptical orbit, what is the direction of the spacecraft's momentum vector? Tangential? Radial? Something else?
- What happens to the momentum as the spacecraft moves away from the Earth?
- As it moves toward the Earth?
- What happens when you increase deltat? Why?
- What do you need to change in order to calculate the force of the Earth on the spacecraft?
- If you double the initial velocity, how much will the force change?
- If you decrease the spacecraft mass by a factor of 4, what is the value of the new force?
- Explain the direction of the momentum arrow.
- Why? Explain these changes in momentum in terms of the Momentum Principle.
  Be careful: The Momentum Principle does NOT imply "big force $\rightarrow$ big momentum".

5 The Moon

In the real world, the net force on a spacecraft is rarely due to only one other object. By adding the Moon to your program, you will get to observe the kind of complex motion that the interaction of three or more objects can produce.

Create a sphere to represent the Moon:

Object name	Kind of object	Initial location	Radius	Color
Moon	sphere	4e8 m to the right of Earth	1.75e6 m	white

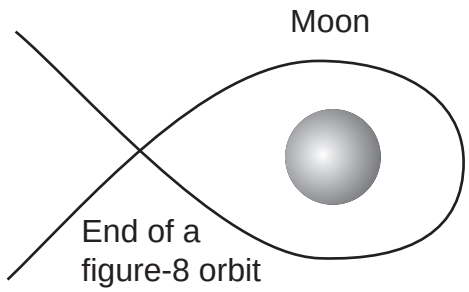
Inside your `while` loop:

- Add a calculation of the gravitational force that the Moon exerts on the spacecraft.
- Calculate the net force on the spacecraft, and use the net force to update the momentum.
- **Add a check for crashing on the Moon, like your check for crashing on the Earth.**
- Initially, use a step size of 10 seconds.
- Lengthen the loop time to 60 days to follow the more complicated orbits that can occur.

5.1 Changing initial conditions

Now you will use your working program to observe different motion. The gravitational interaction of 3 objects can lead to very complicated trajectories. (Orbits are a specific type of trajectory that repeats itself also called a "closed" trajectory.)

- Find an initial speed that leads to crashing on the Moon.
- Find an initial speed that yields a "figure-8" trajectory that loops around the Moon before returning to Earth. Make a note of it on a piece of paper.
- How sensitive is this to the initial velocity? How much can you change the initial speed and still get a figure-8 trajectory?
- Play around with the initial speed and see what other kinds of trajectories you can find.



When the spacecraft interacted solely with the Earth (a "two-body" interaction), there were only a few kinds of orbits possible, and they were quite simple curves (i.e., a circle, an ellipse, a parabola, a hyperbola, or a straight line).

Small changes in the initial velocity typically led to small changes in the orbit (e.g., an ellipse got longer with a larger initial speed.)

In the "three-body" interaction of Earth, Moon, and spacecraft, there is a much larger variety of orbits. An extremely slight change in the initial velocity can make a major change in the motion: **the orbit is highly sensitive to the initial velocity**.

The rich variety of types of motion, and the high sensitivity to initial velocity, are typical of complex systems. This is the subject of the relatively new science of "chaos."

5.2 Questions about trajectories

Answer these questions on your piece of paper about a particularly interesting trajectory.

- Give the spacecraft a speed of 3.27 km/s (3.27e3 m/s), headed in the $+$y direction. What happens?
- Why does coming nearly to a stop lead to retracing the path? Explain in terms of the Momentum Principle and the gravitational force law.

6 Accuracy

If you use a very large Δt, the calculation is inaccurate, because during that large time interval the force changes a lot, making the momentum update inaccurate, and the momentum changes a lot, making the

position update inaccurate.

On the other hand, if you use a very small Δt for high accuracy, the program runs very slowly. In computational science, there is a trade-off between accuracy and speed.

How can you tell whether an orbit is accurate?
There's a simple test: Make Δt smaller and see whether the motion changes. That is, see whether the orbit changes shape. (*Obviously, the program will run more slowly*.)

- You've been using a step size of 10 seconds. Try a step size one-fifth as large (2 seconds). With an initial speed of 3.27 km/s, is the orbit the same? If the orbit is the same using this smaller step size, that implies that a step size less than or equal to 10 seconds is adequately short to give accurate results.
- To see the effects of Δt being too large, make Δt 100 times as large as you had been using (1000 seconds).

Questions:
Does the 10 second step size give an accurate orbit? How do you know? Why does the large step size (1000 seconds) give an inaccurate orbit?

7 Getting a better view

Sometimes, it can be difficult to see the arrows, because your camera view is of the entire scene. We can ride along with the spacecraft by adding to lines of code to the program. This might be helpful for answering the questions in Section 8.

- In the loop that performs your calculations, add these two lines of code:
```
scene.center=craft.pos
scene.range=craft.radius*60
```
- The first line recenters the camera on the spacecraft each time the loop executes.
- The second determines how large of a zoom the camera has in your window. Increase the number to zoom out.
- **Run your program both with and without the influence of the moon.**

8 Exploration

Shoot the spacecraft straight toward the Moon from near the surface of the Earth, starting at location `< 1.01*Earth.radius, 0, 0 >`, with an initial speed of 11 km/s. What happens? Why?

What happens if you increase or decrease the initial speed? It's interesting to watch the momentum vector and think about the effects of the forces on the momentum during this motion.

You can make a more realistic model if you let the Moon and Earth orbit each other while the spacecraft is moving. Calculate all the forces among the three bodies, then update all of the momenta before updating

any of the positions. That way you calculate the forces in a consistent way at a particular instant.

You will have to experiment a bit with the initial velocity to get the nearly circular orbit of the Moon around the Earth (the Earth follows a nearly circular orbit, too, but it is only a small wobble because the Earth's mass is much larger than the Moon's mass.)

You might even try to model the Sun-Earth-Moon system, with or without the spacecraft. A practical difficulty with visualizing the Sun-Earth-Moon system is that the Earth-Moon system is very small compared to the great distance to the Sun, so its hard to see the details of the Moon's motion. However, you can continually update the center of the scene by resetting `scene.center` inside the loop to follow the Earth-Moon system, and zoom in on this region. Alternatively, you can set `scene.autocenter = True`, which makes the centering automatic.

Last edited: Tuesday 12th June, 2012 at 12:34pm