Flyby: work done by a non-constant force

OBJECTIVES

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

- calculate the gravitational force on the spacecraft by the Earth
- apply the momentum principle to update the momentum of the spacecraft
- update the position of the spacecraft.

You will use your working program to explore the work done on the spacecraft by the Earth's gravitational pull.

TIME

You should plan to finish this activity in 50 minutes or less.

GROUP ROLES

Before you begin, you should agree on the responsibilities of the Manager, the Recorder, and the Skeptic for this activity.

Data used in this program:

Mass of Earth: 6e24 kg	Radius of Earth: 6.4e6 m
Mass of spacecraft: 15e3 kg	Radius of spacecraft: very small (exaggerated in program)
	G = 6.7e-11 N m^2/kg^2

You may find it useful to refer to the Earth/Sun program in which you calculated gravitational forces and predicted motion iteratively.

PROGRAM SHELL

A program shell (a skeleton program) has been provided on the class Files webpage. Remember to give it the extension ".py". (If you have problems copying and pasting the shell, perhaps ending up with one very long line with no returns, try saving the file from your browser and then opening it up with IDLE.)

Read through the program shell together. Skeptic: make sure everyone in the group understands the function of each VPython instruction.

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. There are comments in the shell file to help guide you.**
- **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? If it runs too quickly to be seen, insert a rate(xxx) statement inside the loop.**

2. Visualize the force and $\Delta\vec{r}$ vectors with arrows

In a previous program you used arrows to visualize gravitational force vectors. In this program you will also use an arrow to visualize the change in position of the spacecraft.

You will create two **`arrow`**s before the **`while`** loop and update each position and axis inside the loop, as the spacecraft moves. You need to know the approximate magnitude of the force and change in position in order to be able to scale the arrow to fit into the scene, so near the end of the loop print the force vector:

```
print "Fnet =", Fnet
```

Roughly estimate the magnitude of the largest force. Now determine how much the spacecraft will move during the time interval of the loop, which is given by the velocity vector (momentum divided by mass) multiplied by Δt. Replace your previous print statement with two new statements:

```
deltaR =(craft.p/craft.m)*deltat

print "distance moved =", mag(deltaR)
```

This allows you to quickly detect the largest magnitude of the $\Delta\vec{r}$ vector. Use this and the force magnitude estimate to find the scaling factors for the arrows that will represent these vectors. As an example, you know that the distance between the center of the Earth and the initial position of 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. How long is that distance in the units of your virtual world? What scalar factor would you need to multiply the force 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

```
fscale = ?? ## the value you decided on
```

Do a similar estimation for the $\Delta\vec{r}$ vector.

```
rscale = ?? ## the value you decided on
```

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

```
Force_Arrow = arrow(color=color.yellow)

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

These statements create arrow objects with default position and axis attributes. You will set these attributes inside the loop.

- **Comment out any print statements that are inside your loop because they slow down plotting.**
- **Inside your loop, update the pos attribute of the Force_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 on the spacecraft (multiplied by fscale).**
- **Inside your loop, calculate the change in position of the spacecraft during this time through the loop. Recall that velocity multiply by time interval gives distance traveled.**

```
deltaR = craft.p/craft.m*deltat
```

- **Now, update the pos and axis of the deltaR_Arrow object to represent the current change in position of the spacecraft (multiplied by rscale).**
- **Run the program and adjust the scaling factors, as necessary, to produce the best display.**

3. Determine the work being done by the Earth's gravitational force

- **Initialize the amount of work done in the `#constants` section of your program.**

```
Work = 0
```

- **Near the end of your loop, add a statement that increments the total amount of work done by adding the additional work done during this time through the loop. Notice that VPython knows how to calculate the dot product of two vectors.**

```
Work = Work + dot(Fnet,deltaR)
```

- **Print this value so you can see how the total amount of work done on the spacecraft changes as it passes by the Earth.**

```
print "Total Work Done = ", Work
```

- **Run the program and adjust the windows so you can watch the changing value of the total work done of the craft as it is printed on the screen.**

When is the work increasing, and when is it decreasing? What is happening in the simulation during these times? Relate this to the dot product calculation.

4. Examine the work being done

- **After the loop (don't indent), print the final speed of the craft.**

```
print "Final speed = ", mag(craft.p/craft.m)
```

How does the final speed of the craft compare to its initial speed? Is this what you expected, given the total amount of work done on the craft?

- **Increase the number of steps to 1000.**

Again compare the final speed of the craft compare to its initial speed. Is this what you expected, given the total amount of work done on the craft?

5. Think about what you did

Phys 0475 is a calculus-based course. The calculations done by your program are sometimes called "numerical integration." Explain why this term is appropriate and why calculus was needed in this situation. You may want to read page 184 in the textbook.