Calculating and displaying gravitational force with VPython

OBJECTIVES

In a future program you will model the motion of a spacecraft traveling around a planet and a moon. The objectives of the current exercise are:

- to learn how to instruct VPython to calculate the vector gravitational force on an object due to another object
- to write the instructions in a symbolic form that can later be used in an iterative calculation to predict the motion of the spacecraft even though the gravitational force is changing in magnitude and direction
- to create and scale arrows to represent the gravitational force on an object

TIME

You should plan to finish this activity in 40 minutes or less.

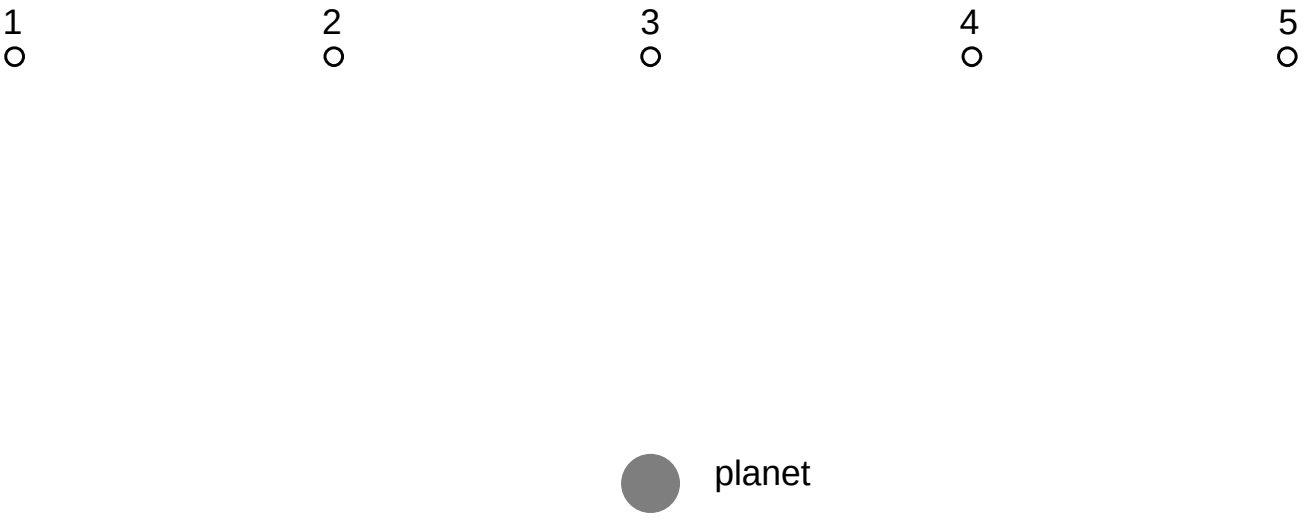
GROUP ROLES

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

PLANNING

On a whiteboard draw a diagram like the one below. Each numbered location represents the position of a different spacecraft. (A single spacecraft near a planet would not move in a straight line.)

- At each numbered loction, draw an arrow representing the gravitational force on a spacecraft at that location, due to the planet. Make sure the direction of your arrows is correct, and that the length of the arrow is proportional to the magnitude of the quantity it represents.



Look at your diagram. Does it make sense? Compare your work to that of another group.

REVIEW: COMPUTER PROGRAM ORGANIZATION

A computer program consists of a sequence of instructions.
The computer carries out the instructions one by one, in the order in which they appear, and stops when it reaches the end.
Each instruction must be entered exactly correctly (as if it were an instruction to your calculator).
If the computer encounters an error in an instruction (such as a typing error), it will stop running and print a red error message.

A typical program has four sections:

1. Setup statements
2. Definitions of constants (if needed)
3. Creation of objects and specification of initial conditions
4. Calculations

I. Setup statements

Using IDLE for VPython, create a new file and save it to your own K drive. Make sure to add ".py" to the file name. Enter the following two statements in the IDLE editor window:

```
from __future__ import division
from visual import *
```

Remember that every VPython program begins with these setup statements.

2. Constants

Since you will be calculating a gravitational force, you will need the constant G (without units)

```
G = 6.7e-11
```

You can also put the masses of the planet and the spacecraft in this section. Define constants to represent:

- the mass of the spacecraft (15e3 kg) (you could call this "**`mcraft`**", for example)
- the mass of the planet (6e24 kg) (you might call it "**`mplanet`**")

3. Objects

Create a sphere object located at the origin to represent the planet, and call it "planet". Its radius should be 6.4e6 m, and its color should be something other than white.

Create a second sphere object named "craft" to represent a spacecraft at location < -13e7, 6.5e7, 0 > m. You will need to exaggerate the radius of the craft to make it visible; try 3e6 m. Make its color different from the color of the planet.

Run the program by pressing F5. Arrange your windows so the Python Shell window is always visible.
Kill the program by closing the graphic display window.

4. Calculations

In a program that models the motion of objects, calculations that are to be repeated are placed inside a loop. In the current exercise we will do a calculation only five times, so it isn't really necessary to use a loop -- once you

get the first calculation right, you can copy and paste to do the others. This is inelegant, but acceptable for this exercise.

Gravitational force law

A sphere of mass m_1 attracts a sphere of mass m_2 with a (vector) force given by $\vec{F}_{grav\ on\ 2\ by\ 1} = -G\frac{m_1 m_2}{|\vec{r}|^2}\hat{r}$, where

G = 6.7e-11 N·m^2/kg^2
$\vec{r}$ is a relative position vector pointing from object 1 toward object 2 ("final minus initial")
$\hat{r}$ is a unit vector pointing from object 1 toward object 2

The steps in calculating gravitational force in VPython are the same as the steps you use on paper:

1. calculate the relative position vector $\vec{r}$ that points from the planet toward the spacecraft
2. calculate its magnitude $|\vec{r}|$
3. use $|\vec{r}|$ to calculate the magnitude of the gravitational force $|\vec{F}_{\text{grav}}| = G\frac{m_1 m_2}{|\vec{r}|^2}$
4. calculate the unit vector $\hat{r}$, which points from object 1 (Planet) toward object 2 (spacecraft)
5. calculate the vector gravitational force acting on the spacecraft, the product of its magnitude and direction: $\vec{F}_{grav} = |\vec{F}_{grav}|\hat{F}_{grav} = -|\vec{F}_{grav}|\hat{r}$

All of the instructions you type should use only symbolic quantities: No numbers (except for exponents, etc.)

Now translate the algebraic expressions into VPython expressions.

1. The relative position vector
You know the vector positions of the two objects, which are **`craft.pos`** and **`planet.pos`**.

- **Add a statement to your program to calculate a vector `r` that points from the planet to the spacecraft, representing the vector $\vec{r}$. Think about what you know about calculating relative position vectors between two objects.** *Don't use any numbers, just symbols. The point is for VPython to do the numerical calculations for a variety of positions of the planet and the spacecraft.*

```
r = ?
```

2. The magnitude of the relative position vector
In order to instruct the computer to calculate the magnitude $|\vec{r}|$ of the relative position vector , you need to know the following:

- **The components of a vector in VPython are given by its x, y, and z attributes. For example, `craft.pos.x` is the x component of the position of the spacecraft, and `r.y` is the y component of the vector `r` that you created.**
- **To calculate a square of a quantity, use two asterisks. For example, `3**2` gives 9.**
- **To calculate the square root of a quantity, use the function sqrt. For example, `sqrt(9)` is 3.**

Knowing these features of VPython,

- **Add a statement to your program to calculate the magnitude $|\vec{r}|$ of the relative position vector:**

```
rmag = ?
```

3. Tthe magnitude of the gravitational force

Using the quantity $|\vec{r}|$ that you just calculated (**`rmag`**), and the masses **`mcraft`** and **`mplanet`**(or whatever you chose to call them),

- **Add a statement to your program to calculate the magnitude $|\vec{F}_{\text{grav}}|$ of the gravitational force. You should define `G` near the start of your program. Use this symbol `G` rather than a number in calculating $|\vec{F}_{\text{grav}}|$:**

```
Fmag = ?
```

4. The unit vector

- **You know both the vector $\vec{r}$ (`r`) and its magnitude (`rmag`). Use these quantities to add a statement to your program to calculate the unit vector $\hat{r}$ (pronounced "r-hat"):**

```
rhat = ?
```

5. The gravitational force as a vector

Now you have everything you need to be able to calculate the gravitational force that the Planet exerts on the spacecraft: $\vec{F}_{grav} = |\vec{F}_{grav}|\hat{F}_{grav} = -|\vec{F}_{grav}|\hat{r}$

- **Add a statement to your program to calculate the net vector force acting on the spacecraft. We'll assume that the planet is the only object near enough to have a significant effect on the spacecraft.**

```
Fnet = ?
```

- **Add a print statement to your program to show the components of the net force:**

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

- **Run the program, and ask yourself whether the signs of the force components make sense (then make changes to your program if necessary).**

Visualizing the force vector with an arrow

Having calculated the gravitational force vector, we want to visualize it by displaying an arrow representing the vector.

- **Add a statement to your program to create an arrow on the screen that represents the gravitational force acting on the spacecraft (object 2). Choose `pos` and `axis` attributes so that the tail of the arrow is on the spacecraft and the arrow points toward the planet. *Do NOT use numbers*! Write the values for the `pos` and `axis` attributes symbolically, in terms of the quantities you have already calculated.**

```
arr2 = arrow(pos=?, axis=?, color=color.yellow)
```

If you have calculated the gravitational force correctly, you probably don't see an arrow! The force is so small you have to scale it up to be able to see it, by multiplying by a scalar factor. How do you pick a scale factor?

Estimating a good value for a scale factor

This is what mapmakers have to do: pick a scale factor and stick to it for representing the real world on a small map.

- **Run your program so you see your display showing the spacecraft and the planet.**
- **Put your hand near the monitor and, with your fingers, show about how long you would like the arrow representing the force to be.**
- **Estimate how long this is in the units of your virtual world. (Approximately what is the distance between the spacecraft and the planet? How long do you want your arrow to be, compared to this distance?)**
- **You printed out the value of the force, so you can estimate the magnitude of the force. What would you need to multiply this by to get an arrow of the length you want? This multiplier is the scale factor (a scalar constant). For example, if the magnitude of the force was 2 Newtons, and you wanted an arrow that was 100 meters long in your virtual world, what scalefactor would you use?**

Desired_length_of_arrow_ = approximate_magnitude_of_quantity * scalefactor

You may need to experiment a little to fine-tune your choice. Once you assign a value to the scalefactor, your program should not change its value (your "map" has to have a constant scale).

- **Add a statement in the Constants section,** *near the start of your program,* **to calculate an appropriate scale factor `Fscale`, and use it to scale the axis of your arrow statement, so that the arrow is visible. Single numbers, positive or negative, are called "scalars," and they can be used to scale vectors. Use this same scale factor with all arrows in this program.**

```
Fscale = ?
```

The force at other positions

In order to see how the force on a spacecraft would vary at different positions, add code to create 4 other spacecraft (spheres) at the locations listed below, and to calculate and display the force on each spacecraft.

< -6.5e7, 6.5e7, 0 > m
< 0, 6.5e7, 0 > m
< 6.5e7, 6.5e7, 0 > m
< 13e7, 6.5e7, 0 > m

- **It is okay to do this by copying and pasting the code you have already written.**

- **If the arrows get too big, reduce your scalar `Fscale` so that the arrows don't run into each other.** You must use the *same* `Fscale` with *all* of your arrows, so that all the force vectors are consistent.

Look at your display. Does it make sense? Does it look like the diagram you drew on the whiteboard? Compare your result to that of another group.
The Recorder should turn in the program to WebAssign, and make sure everyone has a copy of the program.

Optional additions

Label the arrows:

Look in the online reference manual to find out how to add a text label to your display. Label the spacecraft and the planet. Also label at least one of the arrows representing force.

You needn't set all the many possible attributes of a label, but you do need to place the label appropriately and specify what text to display.

To access the VPython reference manual, pull down the Help menu in IDLE, and select "Visual". You will get an html page; click on the link labeled "Reference manual" under the header "The Visual Module for 3D graphics".

Write a loop:

In order to emphasize the physics rather than programming techniques, we had you make multiple copies of the calculational statements, which is clunky, and not feasible if there are hundreds of positions involved. It is not difficult to have just one copy of the calculations in a while loop where you increment the x component of the position of the spacecraft each time through the loop. Try it!

Drag the spacecraft (advanced):

An advanced topic is to modify your program in such a way that you can drag the spacecraft around with the mouse, displaying the forces as you move, in which case you don't want to create new arrows all the time, just update the attributes of the existing arrow. To see how to handle mouse interactions, study the VPython example programs that use the mouse (File > Open just after starting up IDLE), and see the "Visual" documentation on the IDLE help menu. Choose "Reference manual", then "Mouse Interactions".