

Calculating and Displaying Gravitational Force

OBJECTIVES

In a future program you will model the motion of a spacecraft traveling around a planet and a moon, which will require repeated calculation of gravitational forces.

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; and
- 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 and the Recorder for this activity. (Your TA may assign those roles.)

PLANNING

On a whiteboard or a piece of paper, 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 location, 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.

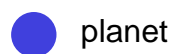
1
○

2
○

3
○

4
○

5
○



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

1 Setup statements

Using VIDLE for VPython, create a new file and save it to your own space. Make sure to add “.py” to the file name.

Enter the following two statements in the VIDLE editor window:

```
from __future__ import division, print_function
from visual import *
```

Both Python and VPython are undergoing continuous improvement. For example, before Python version 3.0, $1/2$ was truncated to zero, but beginning with Python 3.0, $1/2$ means 0.5. Also, starting with Python 3.0, print statements must be in the form `print('hello')` rather than the older form `print 'hello'`. The first statement in your program tells older versions of Python to treat $1/2$ as 0.5, and makes the new form of print statements work on older versions of Python. You don't need this statement if you are using Python 3.0 or later, but it doesn't hurt, because it is simply ignored by later versions of Python.

2 Constants

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

```
G = 6.67e-11
```

You should 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 Creating an object

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 $\langle -13e7, 6.5e7, 0 \rangle$ 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. (You can however use a loop if you prefer.)

5 Gravitational force law

A sphere of mass m_1 attracts a sphere of mass m_2 with a (vector) force

$$\vec{\mathbf{F}}_{\text{grav on 2 by 1}} = -\frac{Gm_1m_2}{|\vec{\mathbf{r}}|^2} \hat{\mathbf{r}},$$

where

- $G = 6.67 \times 10^{-11} \text{ N m}^2/\text{kg}^2$
- $\vec{\mathbf{r}}$ is a relative position vector pointing from object 1 (the source of the force) toward object 2 (the thing feeling the force)
- $\hat{\mathbf{r}}$ 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 $\vec{\mathbf{r}}$ that points from the planet toward the spacecraft.
2. Calculate its magnitude $|\vec{\mathbf{r}}|$.
3. Use $|\vec{\mathbf{r}}|$ to calculate the magnitude of the gravitational force $|\vec{\mathbf{F}}_{\text{grav}}| = (Gm_1m_2/|\vec{\mathbf{r}}|^2) \hat{\mathbf{r}}$.
4. Calculate the unit vector $\hat{\mathbf{r}}$, 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:
 $\vec{\mathbf{F}}_{\text{grav}} = |\vec{\mathbf{F}}_{\text{grav}}| \hat{\mathbf{r}} = -|\vec{\mathbf{F}}_{\text{grav}}| \hat{\mathbf{r}}$.

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 \mathbf{r} that points from the planet to the spacecraft, representing the vector $\vec{\mathbf{r}}$. 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 $|\vec{\mathbf{r}}|$ 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 \mathbf{r} that you created.**
- **To calculate the 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 $|\vec{\mathbf{r}}|$ of the relative position vector:**

```
rmag = ?
```

3. *The magnitude of the gravitational force*

Using the quantity $|\vec{\mathbf{r}}|$ 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 $|\vec{\mathbf{F}}_{\text{grav}}|$ of the gravitational force. You should define G near the start of your program. Use this symbol G rather than a number in calculating $|\vec{\mathbf{F}}_{\text{grav}}|$:**

```
Fmag = ?
```

4. *The unit vector*

- **You know both the vector $\vec{\mathbf{r}}$ (\mathbf{r}) and its magnitude (`rmag`). Use these quantities to add a statement to your program to calculate the unit vector $\hat{\mathbf{r}}$ (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: $\vec{\mathbf{F}}_{\text{grav}} = |\vec{\mathbf{F}}_{\text{grav}}|\hat{\mathbf{F}}_{\text{grav}} = -|\vec{\mathbf{F}}_{\text{grav}}|\hat{\mathbf{r}}$.

- **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 makes 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 a yellow 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.**

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?

6 Scalefactors

If you haven't already done so, watch **VPython Instructional Videos: 5. Scalefactors** at <http://www.youtube.com/VPythonVideos>.

Use what you have learned from this video to determine an appropriate scalefactor for this force arrow. Remember to define the scalefactor in the **# Constants** section of your code, and use it later in the program where the force arrow is defined. The same scalefactor should be used for all force arrows.

The force at other positions

In order to see how the force on the 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.

$\langle -6.5e7, 6.5e7, 0 \rangle$ m
 $\langle 0, 6.5e7, 0 \rangle$ m
 $\langle 6.5e7, 6.5e7, 0 \rangle$ m
 $\langle 13e7, 6.5e7, 0 \rangle$ m

- **Run your program so you see your display showing the spacecraft and the planet.**
- **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* scalefactor 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 earlier? Compare your result to that of another group.

The Recorder should turn in the program to eLC, and make sure everyone has a copy of the program.

7 Extras

You might try one or more of the following additions to your program:

Label the arrows: Look in the online VPython manual to find out how to add a “label” to your display. Label the spacecraft and the planet. Also label at least one of the arrows representing force. To access the VPython manual, pull down the Help menu and select “Visual”.

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!

Place the spacecraft with the mouse: The following statement waits for you to click the mouse somewhere on the screen and gives you the location of the mouse (“scene” is the name of the default graphics window):

```
location = scene.mouse.getclick()
```

With this statement in a loop, you could use this mouse location to specify a spacecraft’s position and then calculate and display the gravitational force on the spacecraft at that location, and then repeat.

You can read more about handling mouse events, including continuous dragging, in the VPython documentation (“Visual” on the Help menu). Also there are a number of example programs that use the mouse which you can find by choosing File > Open immediately upon starting up VIDLE (on a Mac, go to /Applications/VPython-PyX.Y/examples).