2019

03 Moving Objects Using Formulas

W.A. Morgan

L.Q. English

Moving Objects Using Formulas

In this activity, you will be doing four simple exercises. In three of them, a ball will move with either constant velocity or in in the presence of a constant force. In the last, you will move a planet (just a large ball!) around the Sun.

3.1 Motion of a Ball with Constant Velocity

Examine the program provided for you in Figure 3.1. Type it in. Please note that the indentations are important (as is capitalization, usually).

```
1  GlowScript 2.8 VPython
2  # Defining the Object and Constant Velocity
3  obj=sphere(pos=vector(-1,0,0),radius=0.1,color=color.red, make_trail=True)
4
5  # Setting initial conditions and step size, dt
6  t=0
7  dt=0.05
8  x0=-1
9  v0=1.0
10
11 # This is main part - the loop
12 while t<2:
13     rate(10)
14     x=x0+v0*t
15     obj.pos=vector(x,0,0)
16     t=t+dt
```

Figure 3.1: Motion of a ball with constant velocity

What is each line doing? You can probably figure it out using the commented-out lines. Now run the program. What is happening?

You probably found that this program is having a sphere move horizontally at a constant velocity. The command in line 14:

```
x = x0 +v*t
```

is telling the program to take the original value of x, x0, and to add a constant velocity times the time elapsed t. This works, but we will see in the next chapter that it is not the most elegant way of updating the value of x.

3.2 Motion of a ball with constant velocity with a plot of position vs. time

Looking at the motion of the ball is fine, but plotting the values of position, velocity, or acceleration versus time is a way to visualize what is happening.

Take the first program, copy it with a new name, and add commands to plot the ball's position with respect to time. To do this, you must set up a graph and declare its width and height:

Your program should look something like the one in Figure 3.2:

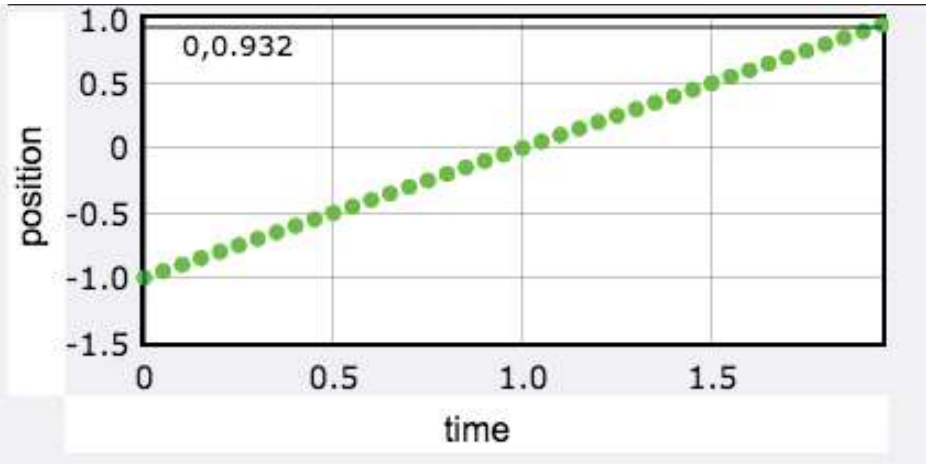
```
1  GlowScript 2.7 VPython
2  # Setting up the Position Graph
3  g1=graph(width=400, height=200,xtitle='time', ytitle='position')
4  xDots=gdots(color=color.green, graph=g1)
5
6  # Defining the Object
7  obj=sphere(pos=vector(-1,0,0),radius=0.1,color=color.red, make_trail=True)
8
9  # Setting initial conditions and step size, dt
10 t=0
11 dt=0.05
12 x0=-1
13 v0=1.0
14
15
16 # This is main part - the loop
17 while t<2:
18     rate(10)
19     x=x0+v0*t
20     obj.pos=vector(x,0,0)
21     xDots.plot(t,x)
22     t=t+dt
```

Figure 3.2: Program Illustrating Graphing Commands

The command **graph** in line 3 sets up a graph with width 400 pixels, height 200 pixels, with an x-axis labeled "time" and a y-axis labeled "position". The set-up graph is called "g1" for ease of use. Line 4 uses the command **gdots** to say how the data is to be plotted, and call it "xDots". Here, it says that green dots should be used in the graph called "g1" that was set up in the line before. Finally, line 21 says that "xDots" should be plotted with the values of "t" (for time) for the x-coordinates, and the values of "x" (for position) for the y-coordinates.

Your plot should like what is shown in Figure 3.3. Is this what you would expect? Why?

Figure 3.3: Plot of position versus time for ball with constant velocity.



3.3 Motion of a Ball Being Dropped from Rest Near the Surface of the Earth

Now, let's simulate the motion of a ball being dropped from a height of two meters.

Questions: What has to be done to modify the second program to mimic the ball being dropped? What is different about this ball's velocity compared to the ball's velocity in the second program?

What does the position-time plot look like now? How is it different from the previous plot? Why?

Plot the velocity versus time. How does this compare to the current position versus time plot and the previous position versus time plot?

3.4 Stop and Go

In the previous section you probably discovered that instead of using a constant speed `v0`, you needed to increment the speed with every pass within the `WHILE` loop.

Now let's use a similar idea to make the ball move to the right at a constant speed for 2 seconds, then to have it stop and sit there for 2 second, and then to have it move to the left at a constant speed for 2 seconds.

Discuss a possible strategy with your partners. You may find that there are multiple ways in which you could accomplish this objective. One way would be to have three different WHILE loops - one after the other. However, perhaps a more elegant approach uses the **if-then-else** syntax - a core syntax structure of any programming language. In VPython, it gets implemented in a very intuitive way:

```
If ("Condition to be tested"):
  "Execute these commands"
Else:
  "Execute those commands"
```

Here "Condition to be tested" should be something like `t<2`, `t>2`, or `t==2`, for example. Notice the double equal signs that are needed because the line should be interpreted as a test of a conditional statement (and not an assignment). Note that the indentations are essential - horizontal alignments signify and enforce structure in any Python program.

Can you incorporate if-statements to make the ball move to the right, stop, and then move to the left without any discontinuities? It may

take you a while to get the third part right.

3.5 Motion of a Planet

In Figure 3.5 examine the program simulating the motion of a planet (the Earth) around the Sun. This is a very simplified depiction; it does *not* take into account gravity or any of the laws of planetary motion. It merely assumes the Earth's orbit is a perfect circle (which in reality is not too far from the truth). In the program, note that there are cosine and sine terms, to define the x and y positions.

```
1  GlowScript 2.8 VPython
2  # Circular Planetary Motion
3  obj=sphere(pos=vector(10,0,0),radius=0.5,color=color.red)
4  sun=sphere(pos=vector(0,0,0),radius=1.0,color=color.yellow)
5
6
7  # Setting initial conditions and step size, dt
8  t=0
9  dt=1
10 y0=2
11 v0=0.0
   theta=0
   thetavenus=0
14 omega=2*pi/365
15
16
17 # This is main part - the loop
18 while True:
19     rate(100)
20     x=10*cos(theta)
21     y=10*sin(theta)
22     obj.pos=vector(x,y,0)
23     theta=omega*t
24     t=t+dt
```

Figure 3.4: Simplified Motion of a Planet Around the Sun

The term omega, written as ω, is called the angular velocity (or the frequency). You will see this term later in the course when we discuss rotational motion. Its value in the program will help you to see

how it is defined.

Exercise: Change the parameters of the provided program. How does the executed program change?

Exercise (optional): Define a third sphere and call it “Moon”. Use the same approach to make the Moon revolve around the Earth, while the Earth still simultaneously revolves around the Sun. Hint: Vector addition will be very useful here. You have to add something to the position vector of the Earth.