5. DOING STUFF A BUNCH OF TIMES: PYTHON FUNCTIONS

Maybe we should do a quick review. Here's what we have done so far:

- Introduced the idea of a numerical calculation with simple 1D kinematics.
- Looked at more difficult problems involving non-constant forces—the simple harmonic motion for a mass and spring. Then we did another non-constant force: air resistance.
- 3D objects and vectors and stuff.
- Modeling crazy complicated things that everyone likes to pretend are easy—like a pendulum.

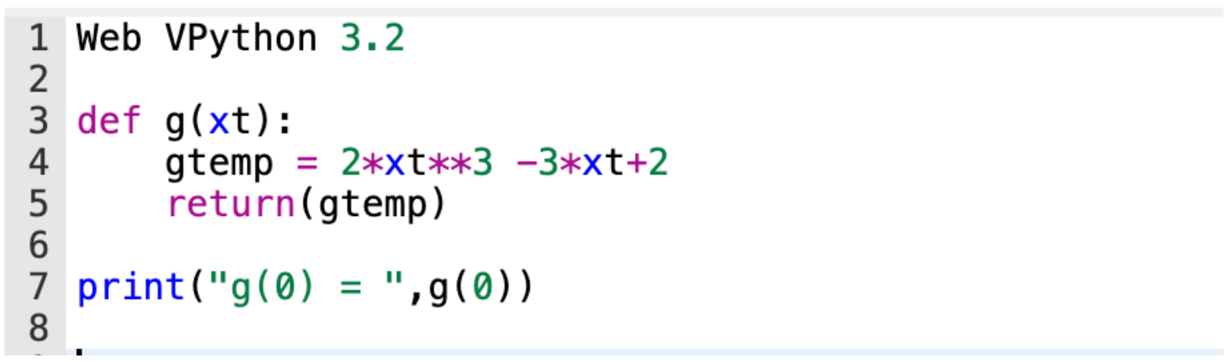
That's what we've done so far. Now for some more python-specific stuff.

WHAT IS A FUNCTION?

I like to think about python functions like they are mathematical functions. This is nice because you can make a python function that looks just like a normal plain math function. Suppose we have the following:

$$g(x) = 2x^3 - 3x + 2$$

This says that we can give a value of x to the function g and it returns some value. If you give it the value x = 0, you get back the value of 2. Right? Let's build a python function that does the same thing.

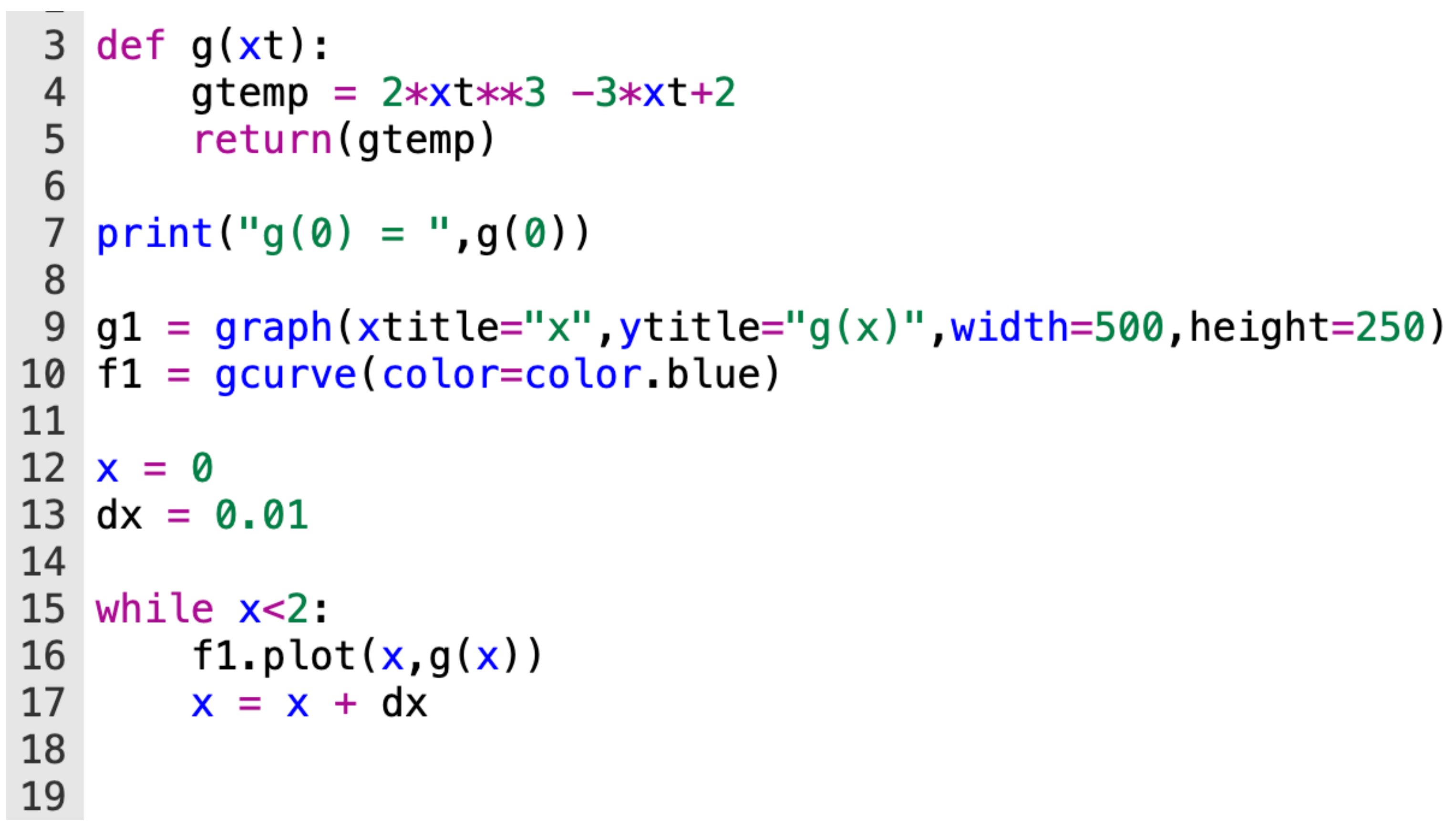


```
Web VPython 3.2

def g(xt):
    gtemp = 2*xt**3 -3*xt+2
    return(gtemp)

print("g(0) = ",g(0))

```

Some comments:

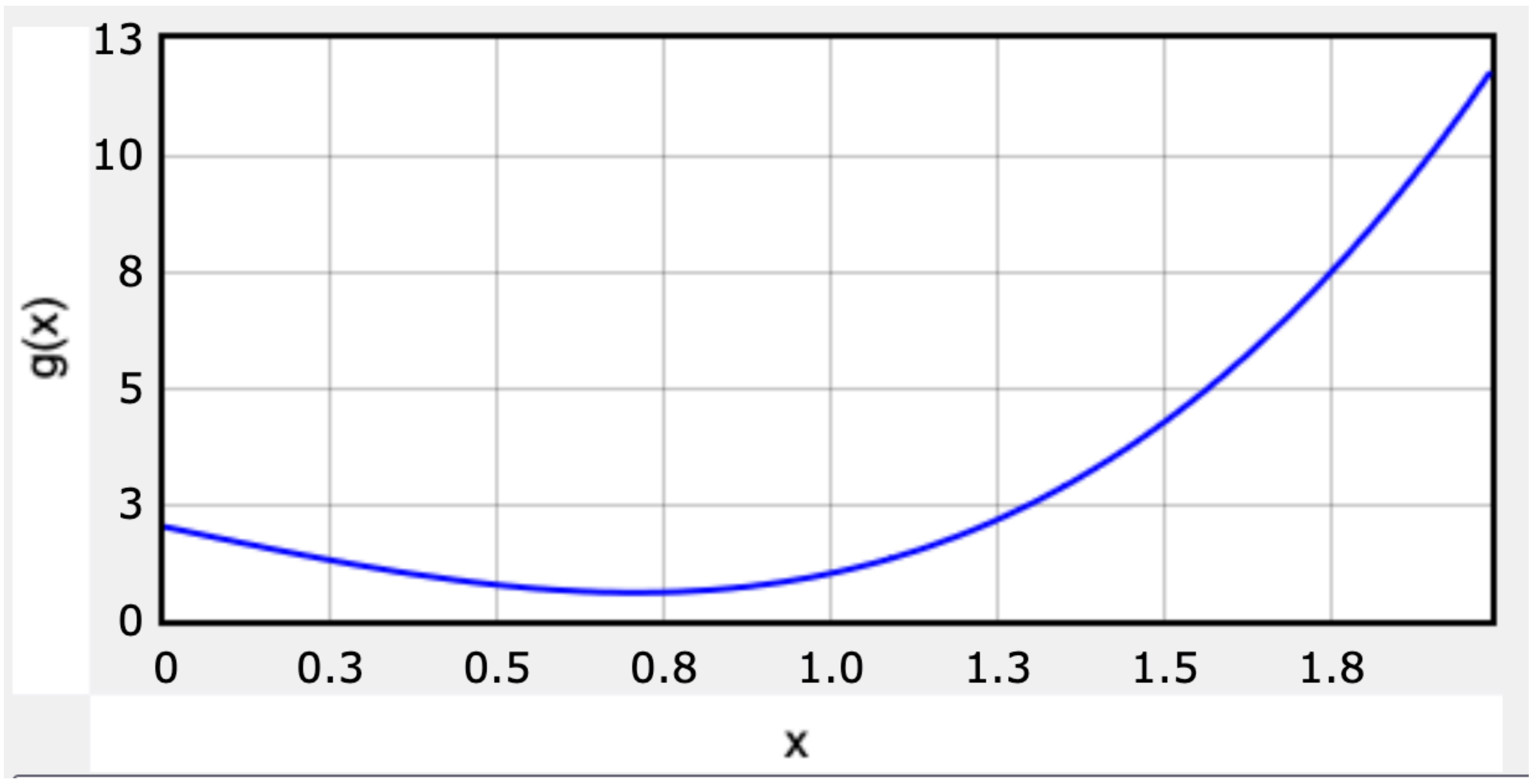
- You can create a function by starting with `def` and then giving it some appropriate name. You can't start your function name with a number and it can't be a reserved word (like `while` or `print`).
- In definition of the function, you need to tell it what things it is expected to receive. Here, I'm passing something called `xt`. Yes, we want g as a function of x but I like to use temporary names (like `xt`) to distinguish between global variables and variables that are just used in the function.
- The function starts after the colon and everything that is tab-indented is part of the function.
- You can do stuff (like calculate a temp value for g) and then the `return()` is the output.

Let's use this function to make a graph of g(x) from x = 0 to x = 2.



```
def g(xt):
    gtemp = 2*xt**3 -3*xt+2
    return(gtemp)

print("g(0) = ",g(0))

g1 = graph(xtitle="x",ytitle="g(x)",width=500,height=250)
f1 = gcurve(color=color.blue)

x = 0
dx = 0.01

while x<2:
    f1.plot(x,g(x))
    x = x + dx


```

Notice that I have the variable x that goes from 0 to 2 (just like did with other stuff). The cool part is that I can treat g(x) just like a variable. I can put it in the plot function and it will go ahead and pass the current value of x to the function which returns the calculation. Here's the graph.



How about a useful example?

THE QUADRATIC EQUATION

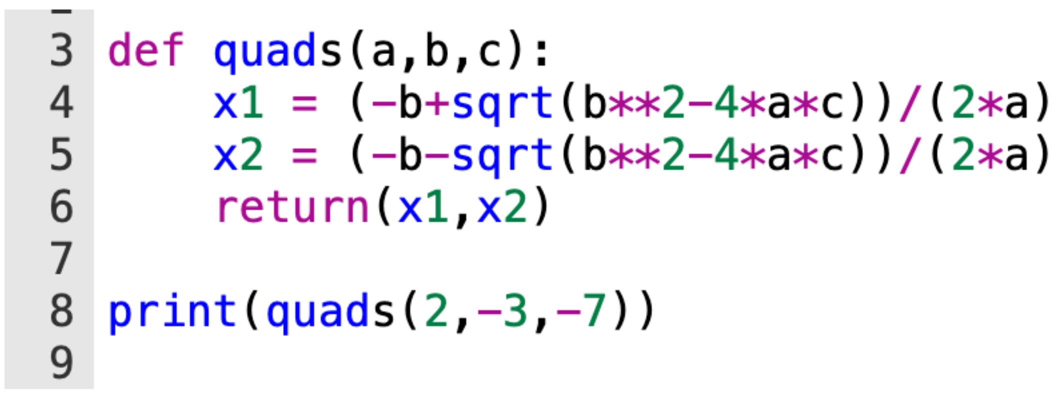
Suppose I have a quadratic function that is set to zero. The generic form looks like this:

$$0 = ax^2 + bx + c$$

This has the solutions:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

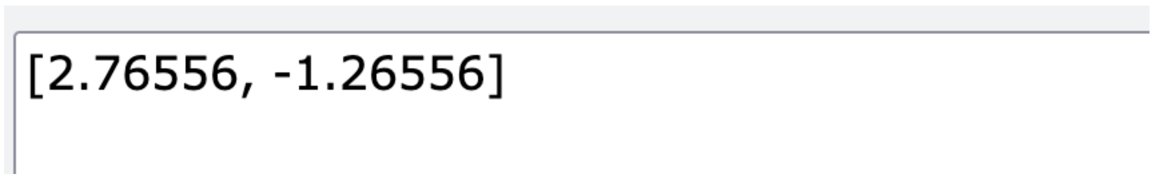
Let's make a generic python function that gives the two solutions for this polynomial. Here it is.



```
def quads(a,b,c):
    x1 = (-b+sqrt(b**2-4*a*c))/(2*a)
    x2 = (-b-sqrt(b**2-4*a*c))/(2*a)
    return(x1,x2)

print(quads(2,-3,-7))
```

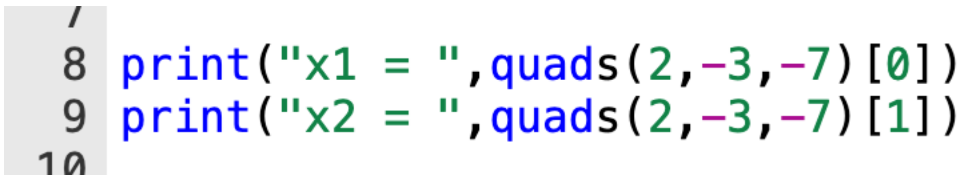
This is not a perfect function (because it's possible to give it some parameters that do not have real solutions. We can deal with that later. The important thing is that this function, just like the actual quadratic formula, gives us TWO values. Here, I calculate these as x1 and x2.

But how does a function output two things? It does it as a list. When you run the above code, you get the following:



```
[2.76556, -1.26556]
```

This is a list of the two solutions. If you want to print them separately, you could use this code.



```
print("x1 = ",quads(2,-3,-7)[0])
print("x2 = ",quads(2,-3,-7)[1])
```

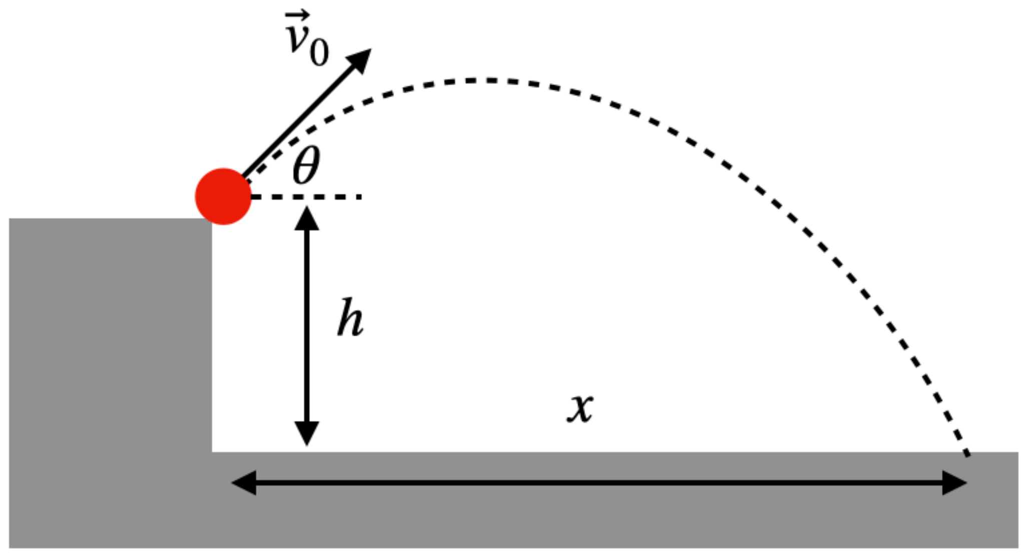
The `quads(2,-3,-7)[0]` refers to the first item in the list and `quads(2,-3,-7)[1]` is the second item in the list. Don't worry—we will get into lists later.

MAXIMUM RANGE FOR A PROJECTILE

I feel like it's fairly common knowledge that the maximum range for a projectile is when it's launched at a 45 degree angle above the horizontal. I'm not even going to derive this. HOWEVER—you have to be careful. This 45 degree angle is only true for a ball that is launched from the ground level and lands back on the ground. It also assumes no air resistance.

OK, let's tackle this problem in a couple of different ways. First, I'm going to get an expression for the horizontal distance as a function of launch velocity and angle and starting height. Then we can plot this for different values of launch angle.

Here's some physics. Suppose I have the following.



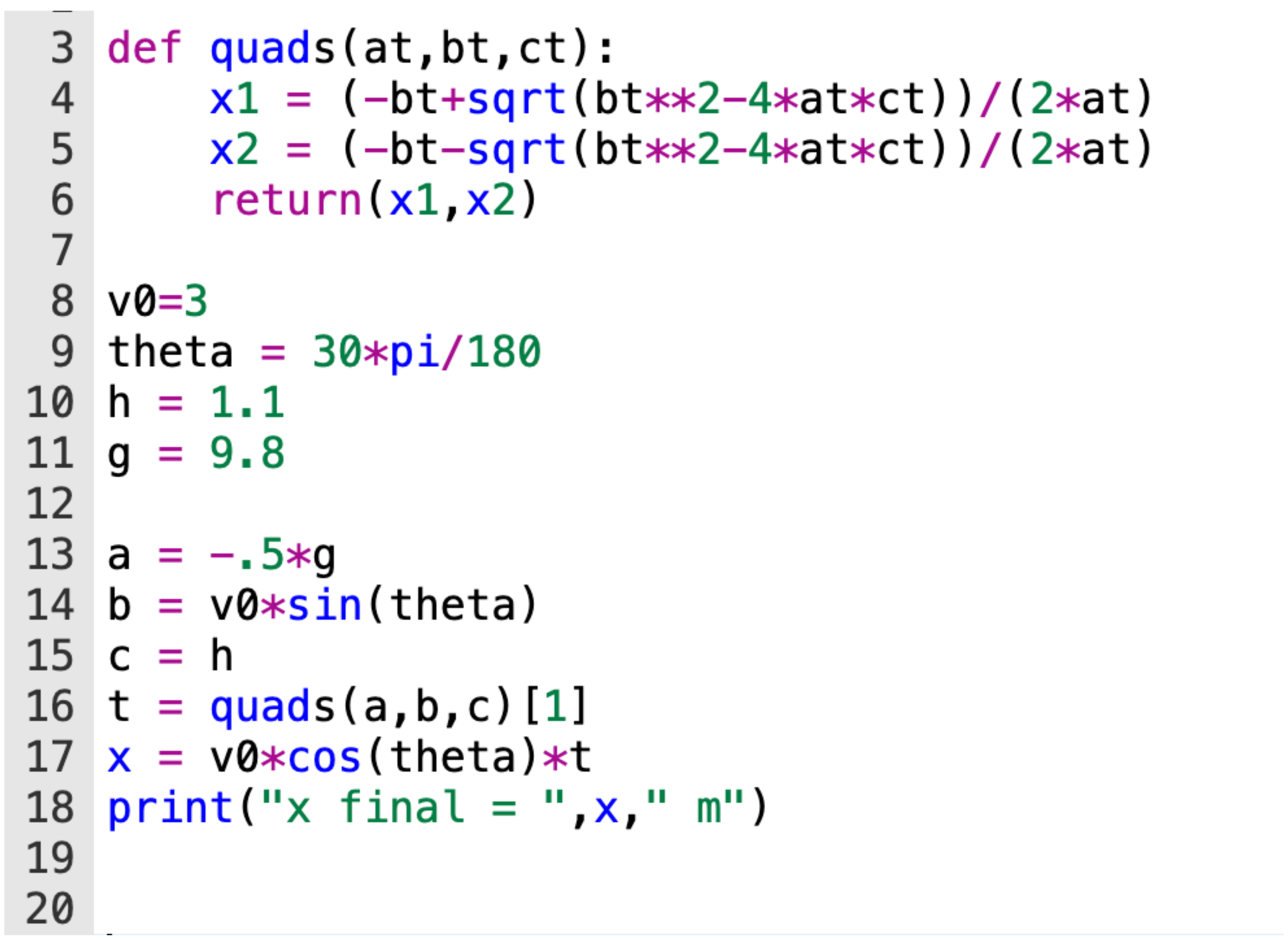
We can break this trajectory into a motion in the x-direction and a motion in the y-direction. The two motions are independent except for the time they take. In the x-direction, the acceleration is zero and it starts at x = 0 and ends at some value x. With that, we have the following kinematic equation.

$$x = v_0(\cos\theta)t$$

In the y-direction, the ball starts at y = h and ends at y = 0.

$$0 = h + v_0(\sin\theta)t - \frac{1}{2}gt^2$$

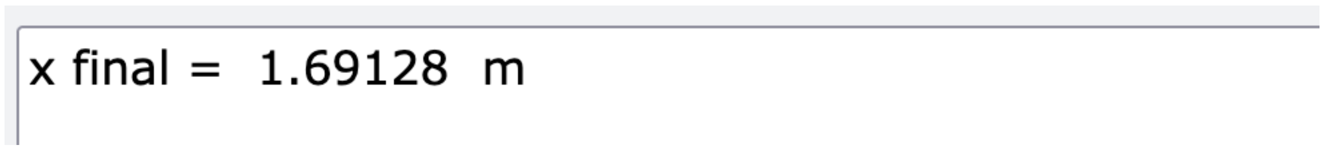
We can use the quadratic equation to solve for the time from the y-motion. But you know what? I'm not even going to do that symbolically—I already have a quadratic formula function from above? This quadratic formula will give me two values for time. One of those times will be negative—and we want the positive version. With that time, I just plug the time into the x-equation to get the horizontal distance. Let's try it out.



```python
def quads(at,bt,ct):
    x1 = (-bt+sqrt(bt**2-4*at*ct))/(2*at)
    x2 = (-bt-sqrt(bt**2-4*at*ct))/(2*at)
    return(x1,x2)

v0=3
theta = 30*pi/180
h = 1.1
g = 9.8

a = -.5*g
b = v0*sin(theta)
c = h
t = quads(a,b,c)[1]
x = v0*cos(theta)*t
print("x final = ",x," m")
```

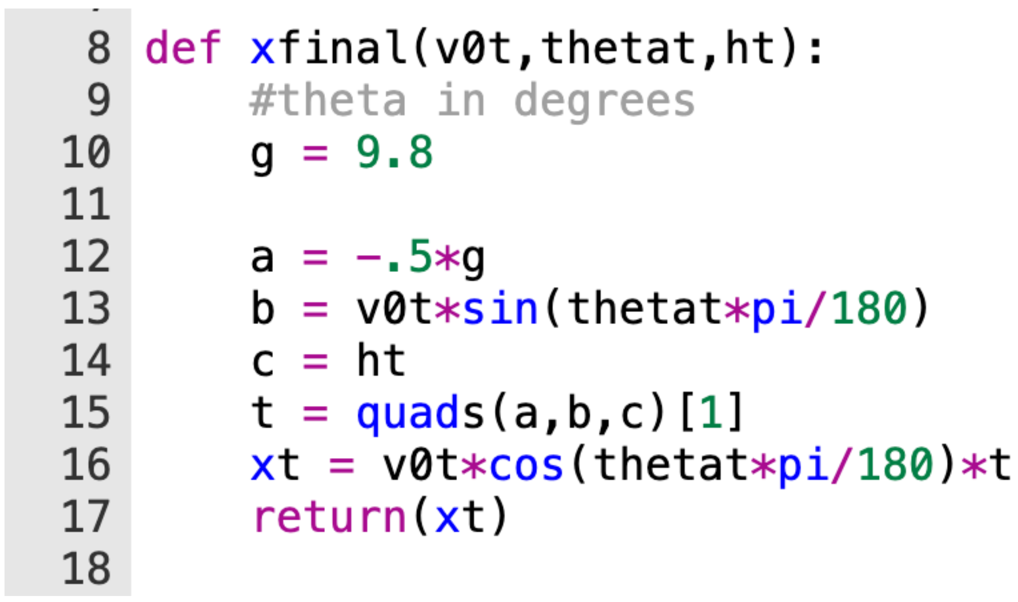
Here I'm using that same quadratic formula function. I changed the input variables so that they would be different than the global a, b, and c. But you can see how awesome this can be. I don't need to use symbolic expressions for a, b, and c and simplify the equation. Remember, I'm just dealing with numbers so I might as well let python do all the work.

When this runs, I get two values for the time—it's the second one that is positive. So I'm using that time to calculate the final position and print it. Nice.



```
x final =  1.69128  m
```

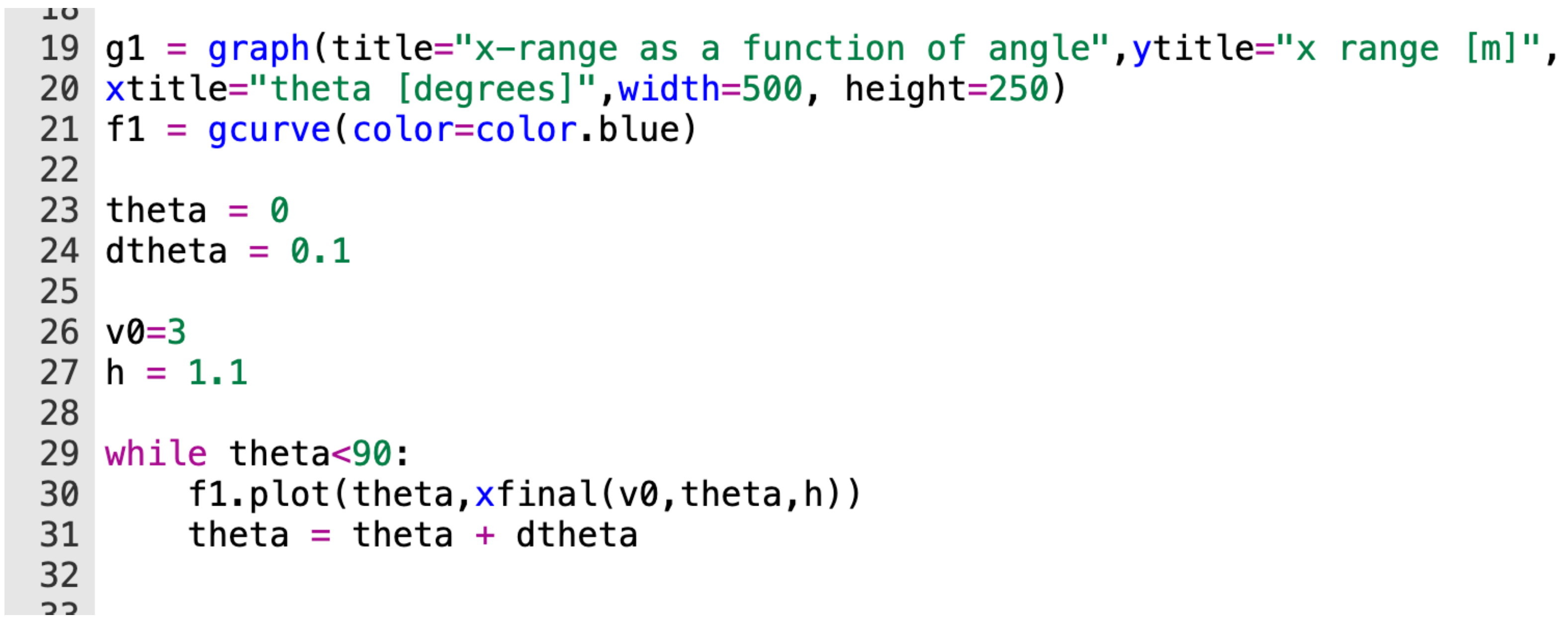
But this just gives us the final x-position for one launch angle. We are going to put all of this stuff into another function.



```python
def xfinal(v0t,thetat,ht):
    #theta in degrees
    g = 9.8

    a = -.5*g
    b = v0t*sin(thetat*pi/180)
    c = ht
    t = quads(a,b,c)[1]
    xt = v0t*cos(thetat*pi/180)*t
    return(xt)

```

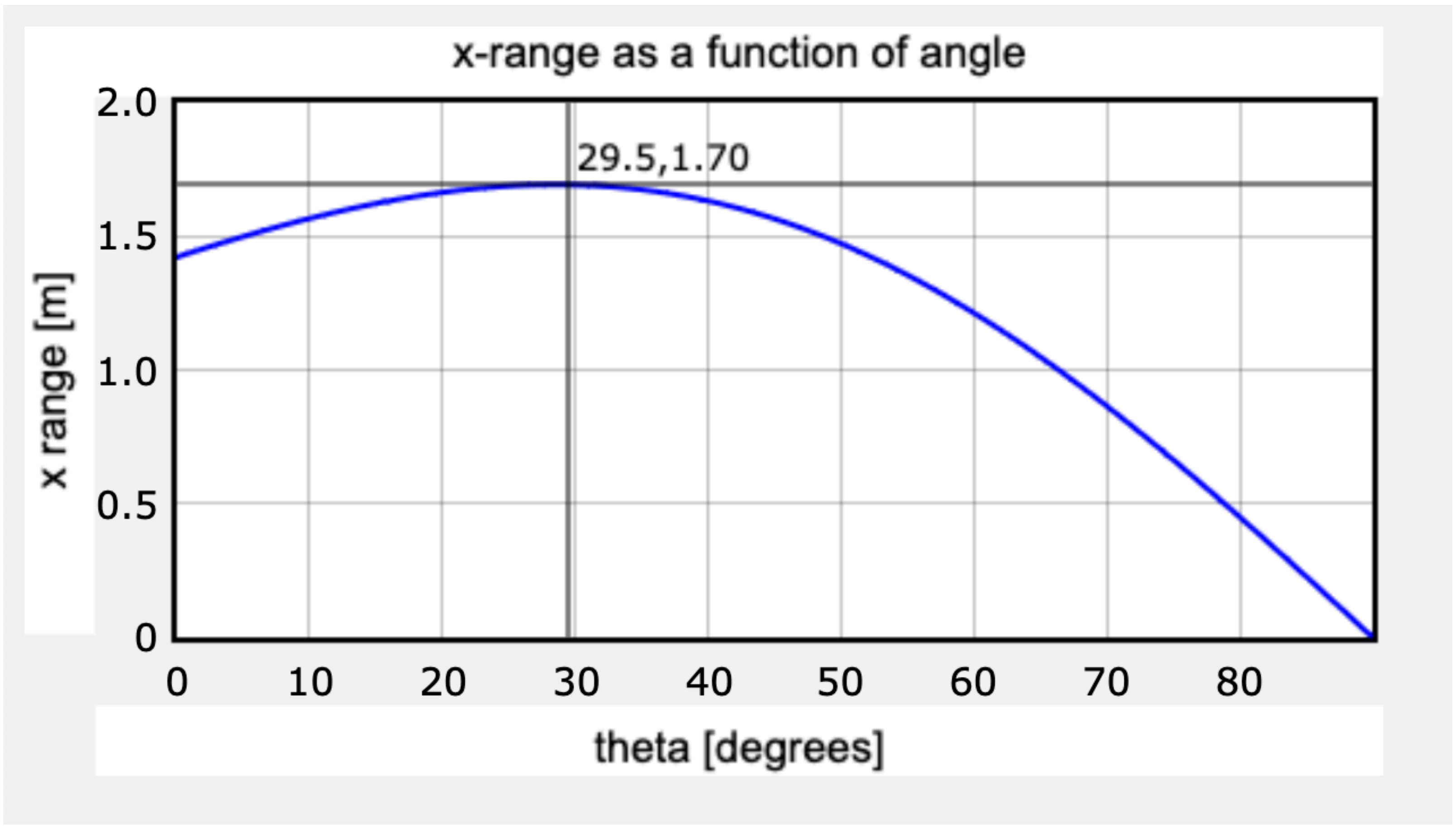
Yes, the default version of sine and cosine take the angle in radians—but I like to think in terms of degrees. There is a built in function to convert from degrees to radians, but I just do it myself.

Now we are ready to roll. I'm going to pick a value of v0 and h and then keep changing the launch angle. I will then plot the final position as a function of launch angle. Here's the code (after the two functions above).



```python
g1 = graph(title="x-range as a function of angle",ytitle="x range [m]",
xtitle="theta [degrees]",width=500, height=250)
f1 = gcurve(color=color.blue)

theta = 0
dtheta = 0.1

v0=3
h = 1.1

while theta<90:
    f1.plot(theta,xfinal(v0,theta,h))
    theta = theta + dtheta

```

Notice that I just call the xfinal function right in the plot function—that's just how I like it. Here's the output.



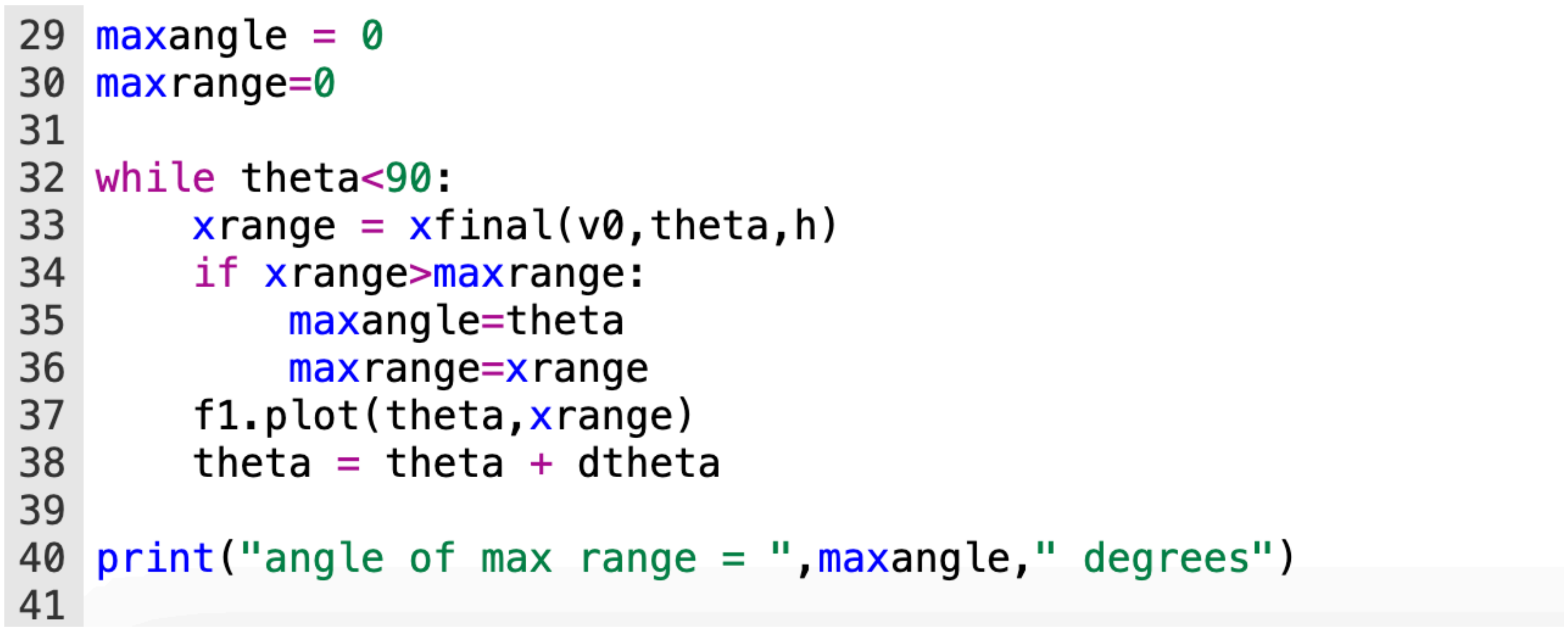
From this, you can see that an angle of 29.5 degrees gives the maximum range. Notice that is NOT the common 45 degrees.

HOMEWORK

- Use the code above and let h = 0. See what you get as the maximum range.
- If you increase the value of h, what happens to the angle of maximum range?
- Modify the code so that it prints the maximum angle instead of reading it from the graph.
- Is it possible that the maximum angle would be greater than 45 degrees?

Key

Here's a little snippet of code to get the angle of maximum range.



```python
maxangle = 0
maxrange=0

while theta<90:
    xrange = xfinal(v0,theta,h)
    if xrange>maxrange:
        maxangle=theta
        maxrange=xrange
    f1.plot(theta,xrange)
    theta = theta + dtheta

print("angle of max range = ",maxangle," degrees")
```

Notice that I have two variables—`maxangle` and `maxrange` that both start off as zero. When I calculate the range, I check if it's greater than the maxes. If so, then I update those values. It works.

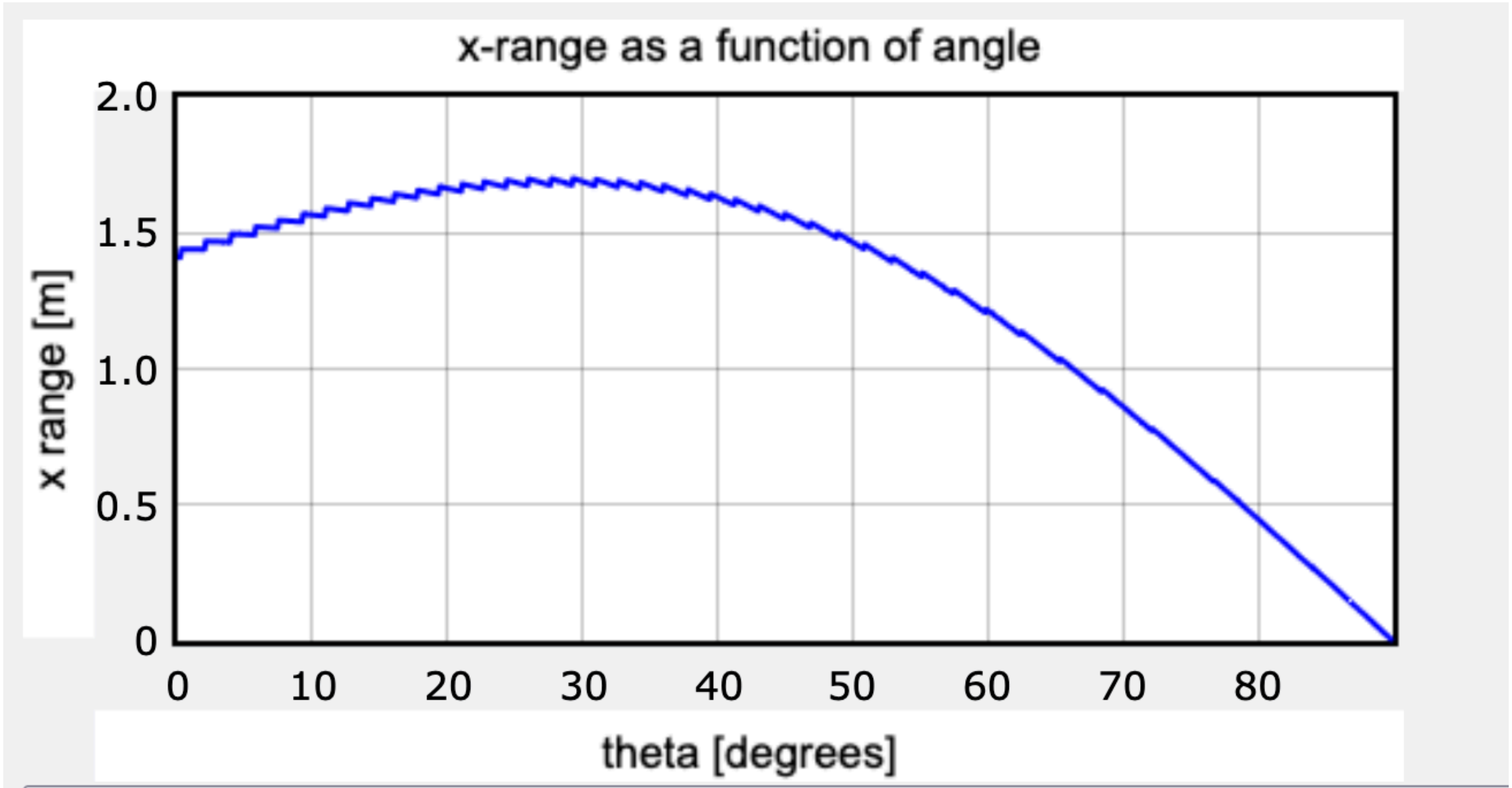
THAT METHOD IS OUR ONLY HOPE TO CALCULATE THE ANGLE FOR MAXIMUM RANGE. NO, THERE IS ANOTHER.

That was what Yoda said about Luke Skywalker. But here I'm talking about python methods. So, let's do this range thing again. Instead of just calculating the theoretical final position of the launched ball, we can instead determine where the ball ends up by doing a numerical calculation.

I can use the same code as before but just change the `xfinal` function. Check it out.

```python

def xfinal(v0t,thetat,ht):
    g = vector(0,-9.8,0)
    tt = 0
    dtt = 0.01
    rt = vector(0,ht,0)
    vt = v0t*vector(cos(thetat*pi/180),sin(thetat*pi/180),0)
    while rt.y>=0:
        vt = vt + g*dtt
        rt = rt + vt*dtt
        tt = tt + dtt
    return(rt.x)

```

You can see that this is just your basic projectile motion calculation—but in a function. Yes, I'm using velocity instead of momentum but that's fine. Here's the output.



```
angle of max range =  29.3  degrees
```

This looks like it works—but why is the curve jagged like that? This is an artifact of the numerical method in the function. Since I have a time step of 0.01 seconds, it's possible that two launched balls could have slightly shifted ranges depending when that ball gets below y =0. It's fine.

MAXIMUM RANGE WITH AIR RESISTANCE

Now for something super fun. Suppose we have a ball that we want to throw to achieve a maximum range. Just for simplicity, imagine that the ball has a mass of 0.02 kilograms and a radius of 0.05 meters. The ball is thrown with an initial velocity of 10 meters per second. If the ball starts and ends at the same level, what is the optimal launch angle? Hint: it's not 45 degrees. Oh, since this is just a normal ball we can assume that the drag coefficient is 0.47 and let's use a density of air at 1.2 kilograms per cubic meter.

Remember that we can model the air resistance as:

$$\vec{F}_{\text{air}} = -\frac{1}{2}\rho A C |\vec{v}|^2 \hat{v}$$

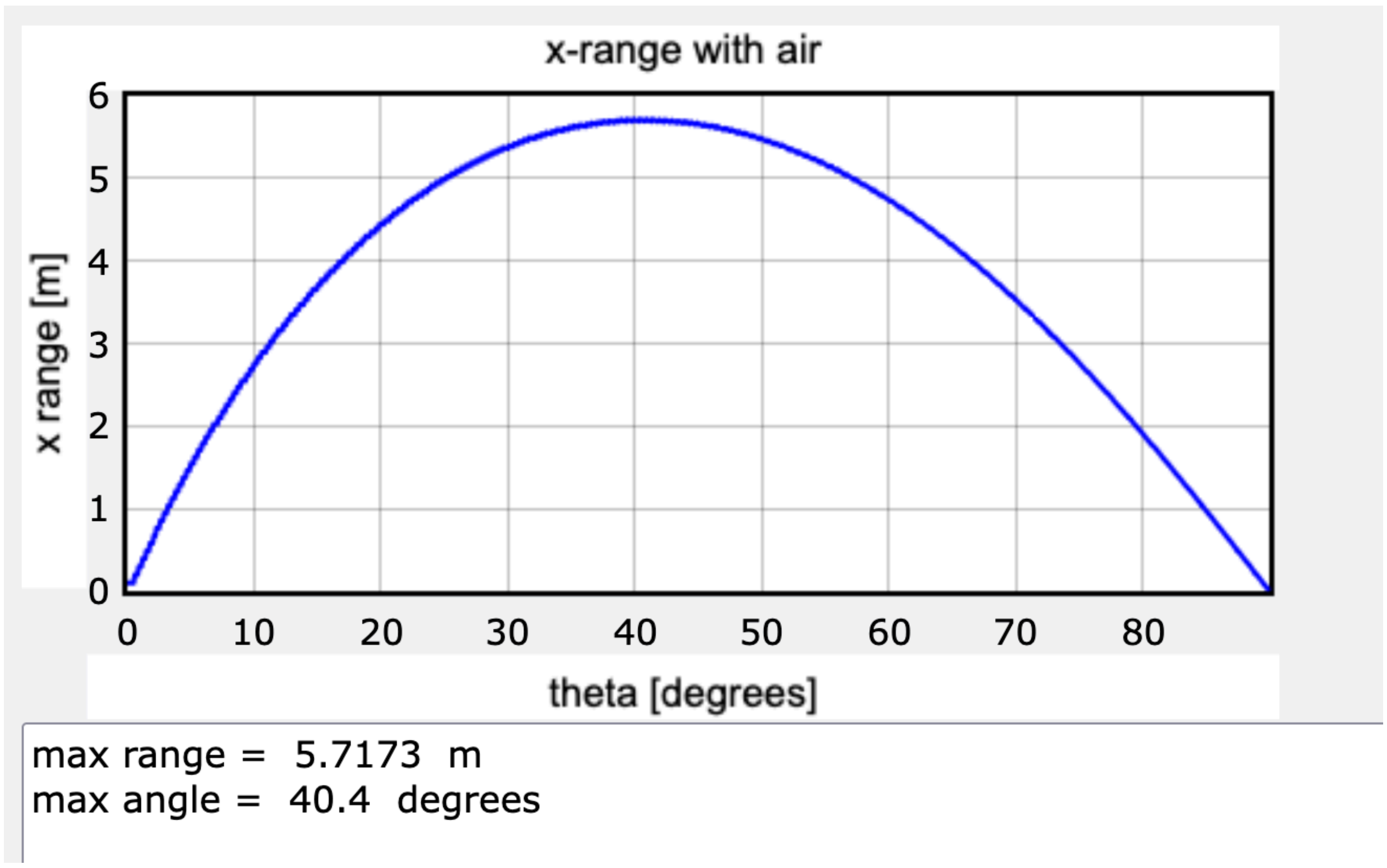
Now I can just add this into the previous range function. OK, I should point out that I'm not going to be using momentum. That's because this version of the air force depends on velocity instead. It's not a big deal. Trust me.

Here is the new xfinal function.

```python
def xfinal(v0t,thetat,ht):
    g = vector(0,-9.8,0)
    m = 0.02
    R = 0.05
    C = 0.47
    A = pi*R**2
    rho = 1.2
    tt = 0
    dtt = 0.01
    rt = vector(0,ht,0)
    vt = v0t*vector(cos(thetat*pi/180),sin(thetat*pi/180),0)
    while rt.y>=0:
        F = m*g -.5*rho*A*C*mag(vt)**2*norm(vt)
        vt = vt + F*dtt/m
        rt = rt + vt*dtt
        tt = tt + dtt
    return(rt.x)

```

I don't really have any comments about this code. It's pretty much just like the last time we did a calculation with air resistance except that it's in a function.

Here's the output



```
max range =  5.7173  m
max angle =  40.4  degrees
```

You can see the maximum angle is indeed LESS than 45 degrees.

HOMEWORK

- There are just too many questions here, but I will list a couple.
- How does the optimal launch angle depend on the launch speed?
- What happens if you put this up on a hill—like in the case without air resistance?
- How does the launch angle change with mass?
- Will the optimal launch angle ever be GREATER than 45 degrees when on flat ground? Hint: the answer might surprise you.

Key

There is no key for these questions.