Understanding the Derivative

April 25, 2018

1 Understanding the Derivative

You just saw these three statements.

1. **Velocity** is the instantaneous rate of change of **position**
2. **Velocity** is the slope of the tangent line of **position**
3. **Velocity** is the derivative of **position**

But there's another, more formal (and mathematical) definition of the derivative that you're going to explore in this notebook as you build an intuitive understanding for what a derivative is.

1.1 BEFORE YOU CONTINUE

This notebook is a long one and it really requires focus and attention to be useful. Before you continue, make sure that:

1. You have **at least 30 minutes** of time to spend here.
2. You have the mental energy to read through math and some (occasionally) complex code.

1.2 Formal definition of the derivative

The **derivative of** $f(t)$ **with respect to t** is the function $\dot{f}(t)$ ("f dot of t") and is defined as

$$\dot{f}(t) = \lim_{\Delta t \to 0} \frac{f(t + \Delta t) - f(t)}{\Delta t}$$

You should read this equation as follows:
"F dot of t is equal to the limit as delta t goes to zero of F of t plus delta t minus F of t all over delta t"

1.3 Outline

In this notebook we are going to unpack this definition by walking through a series of activities that will end with us defining a python function called `approximate_derivative`. This function will look very similar to the math shown above.

A rough outline of how we'll get there:

1. **Discrete vs Continuous Motion** - A quick reminder of the difference between **discrete** and **continuous** motion and some practice defining continuous functions in code.

2. **Plotting continuous functions** - This is where you'll see `plot_continuous_function` which is a function that takes **another function** as an input.
3. **Finding derivatives "by hand"** - Here you'll find the **velocity** of an object *at a particular time* by zooming in on its **position vs time** graph and finding the slope.
4. **Finding derivatives algorithmically** - Here you'll use a function to reproduce the steps you just did "by hand".
5. **OPTIONAL: Finding the full derivative** - In steps 3 and 4 you actually found the derivative of a function *at a particular time*, here you'll see how you can get the derivative of a function for **all** times at once. Be warned - the code gets a little weird here.

1.4 1 - Discrete vs Continuous Motion

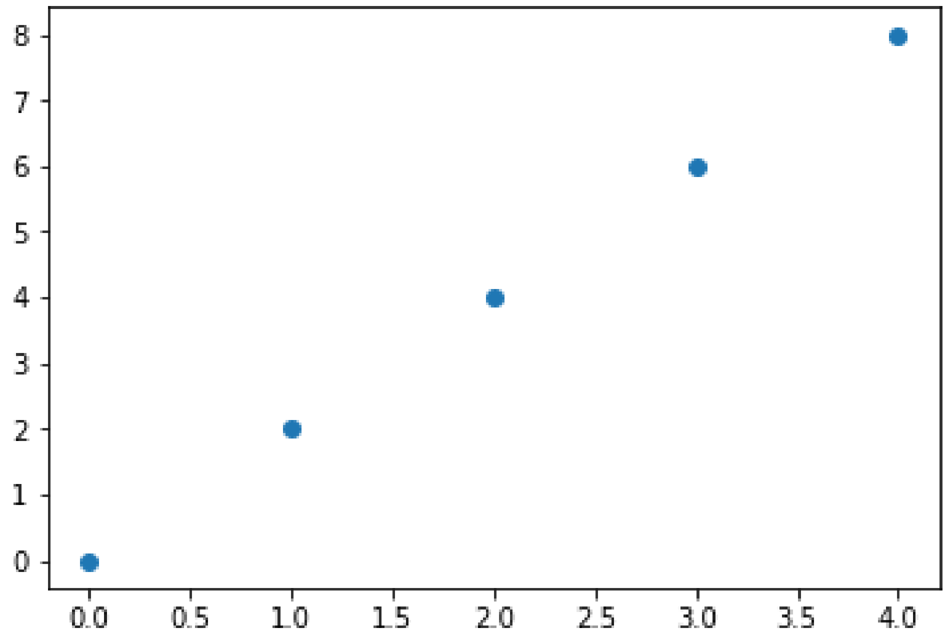
The data we deal with in a self driving car comes to us discretely. That is, it only comes to us at certain timestamps. For example, we might get a position measurement at timestamp `t=352.396` and the next position measurement at timestamp `t=352.411`. But what happened in between those two measurements? Did the vehicle **not have a position** at, for example, `t=352.400`?

Of course not!

> Even though the position data we measure comes to us **discretely**, we know that the actual motion of the vehicle is **continuous**.

Let's say I start moving forwards from `x=0` at `t=0` with a speed of $2m/s$. At t=1, x will be 2 and at t=4, x will be 8. I can plot my position at 1 second intervals as follows:

In [1]:
```
from matplotlib import pyplot as plt
%matplotlib inline

t = [0,1,2,3,4]
x = [0,2,4,6,8]

plt.scatter(t,x)
plt.show()
```



This graph above is a **discrete** picture of motion. And this graph came from two Python **lists**...

But what about the underlying **continuous** motion? We can represent this motion with a function f like this:

$$f(t) = 2t$$

How can we represent that in code?

A list won't do! We need to define (surprise, surprise) a function!

```
In [2]: def position(time):
            return 2*time

        print("at t =", 0, "position is", position(0))
        print("at t =", 1, "position is", position(1))
        print("at t =", 2, "position is", position(2))
        print("at t =", 3, "position is", position(3))
        print("at t =", 4, "position is", position(4))
```

```
at t = 0 position is 0
at t = 1 position is 2
at t = 2 position is 4
at t = 3 position is 6
at t = 4 position is 8
```

That looks right (and it matches our data from above). Plus it can be used to get the position of the vehicle in between "sensor measurements!"

```
In [3]: print("at t =", 2.2351, "position is", position(2.2351))
```

```
at t = 2.2351 position is 4.4702
```

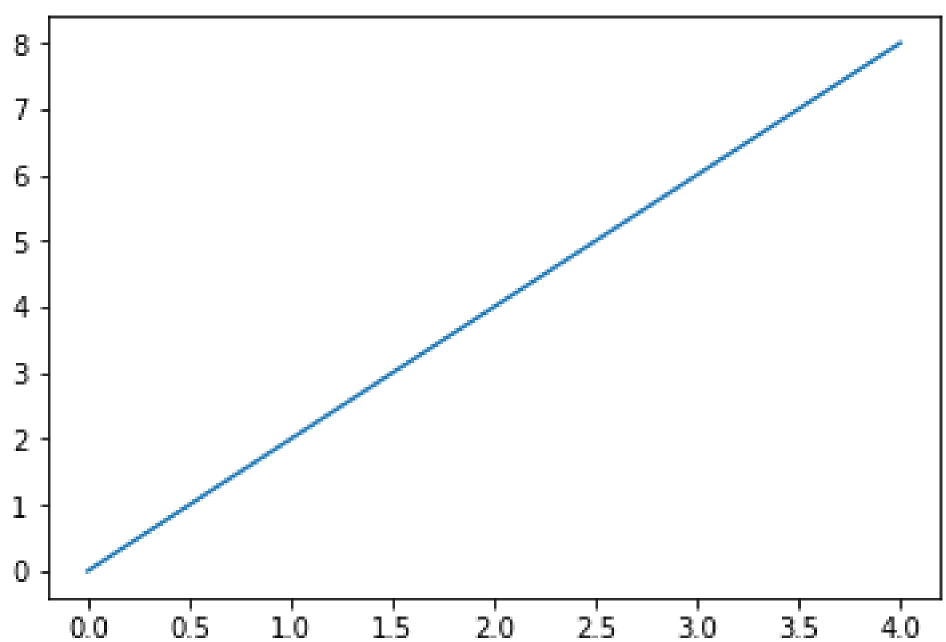
This `position(time)` function is a continuous function of time. When you see $f(t)$ in the formal definition of the derivative you should think of something like this.

1.5 2 - Plotting Continuous Functions

Now that we have a continuous function, how do we plot it??

We're going to use `numpy` and a function called `linspace` to help us out. First let me demonstrate plotting our position function for times between 0 and 4.

```
In [4]: # Demonstration of continuous plotting

        import numpy as np

        t = np.linspace(0, 4)
        x = position(t)

        plt.plot(t, x)
        plt.show()
```

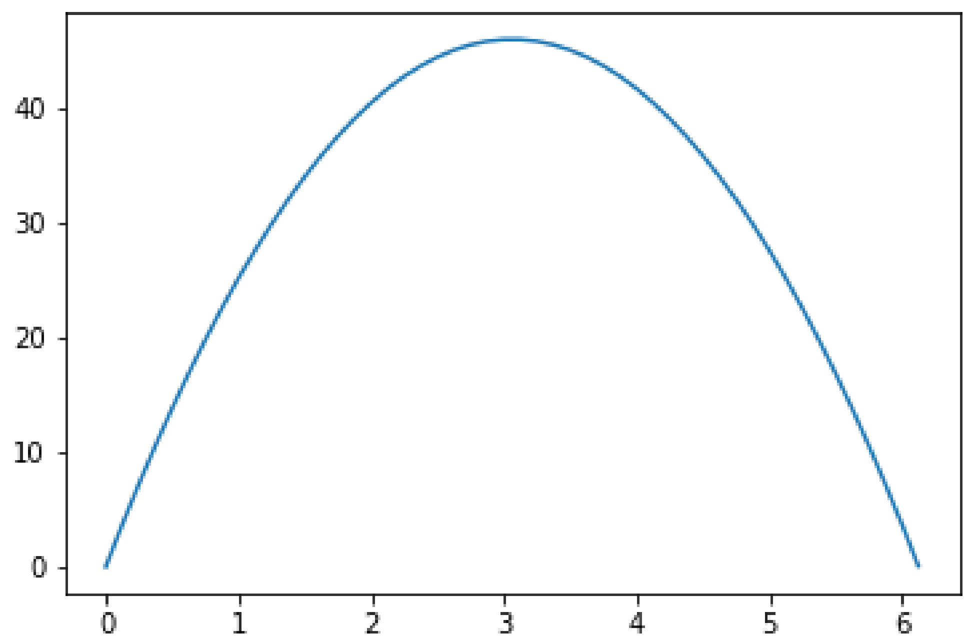


EXERCISE - create and plot a continuous function of time **Write a function,** `position_b(time)` **that represents the following motion:**

$$f(t) = -4.9t^2 + 30t$$

then plot the function from t = 0 to t = 6.12

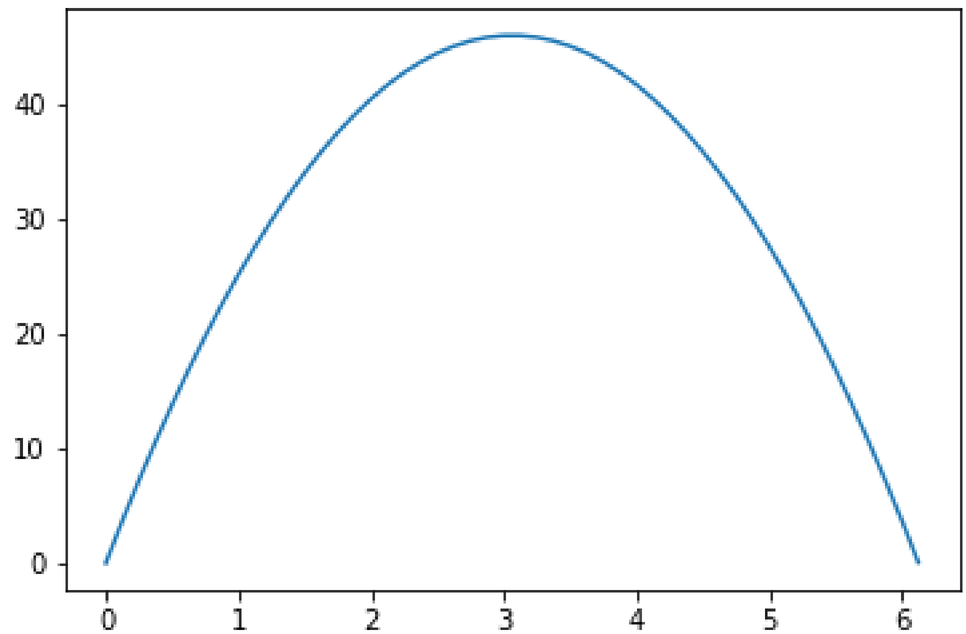
In [10]:
```
# EXERCISE
def position_b(time):
    # todo
    return -4.9 * time**2 + 30 * time

# don't forget to plot this function from t=0 to t=6.12
# Solution is below.
input_x = np.linspace(0, 6.12)
input_y = position_b(input_x)
plt.plot(input_x, input_y)
plt.show()
```



In [11]:
```
#

#

#

# Spoiler alert! Solution below!
```

```
#

#

#
```

```
In [12]: def position_b(time):
             return -4.9 * time ** 2 + 30 * time

         t = np.linspace(0, 6.12)
         z = position_b(t)

         plt.plot(t, z)
         plt.show()
```



Fun fact (maybe)

There's a reason I used the variable `z` in my plotting code. `z` is typically used to represent distance above the ground and the function you just plotted actually represents the height of a ball thrown upwards with an initial velocity of $30m/s$. As you can see the ball reaches its maximum height about 3 seconds after being thrown.

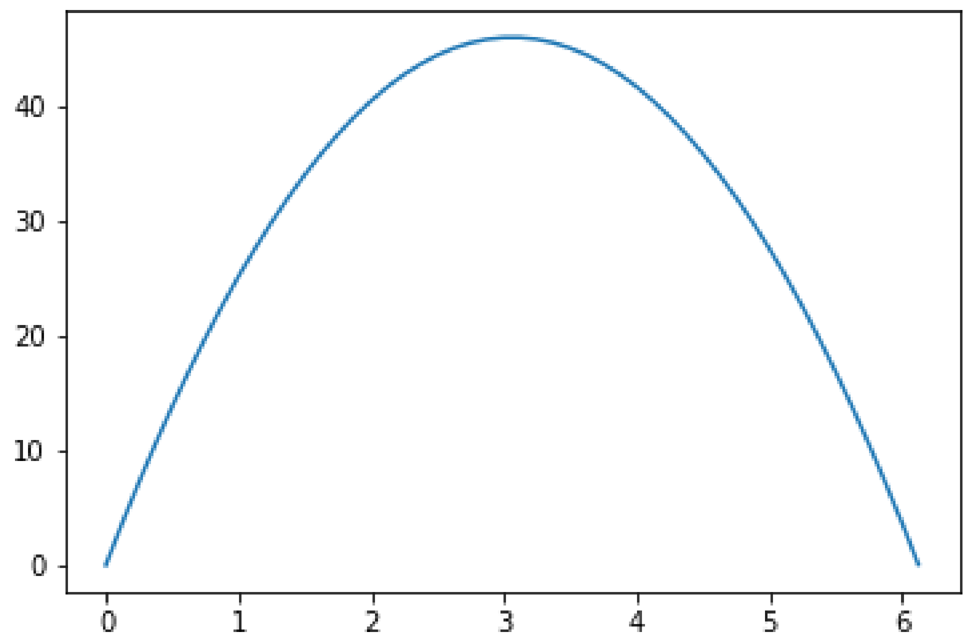
1.5.1 2.1 - Generalize our plotting code

I don't want to have to keep copy and pasting plotting code so I'm just going to write a function...

```
In [13]: def plot_continuous_function(function, t_min, t_max):
             t = np.linspace(t_min, t_max)
```

```
        x = function(t)
        plt.plot(t,x)
```

```
In [14]: plot_continuous_function(position_b, 0, 6.12)
         plt.show()
```



Take a look at `plot_continuous_function`.

Notice anything weird about it?

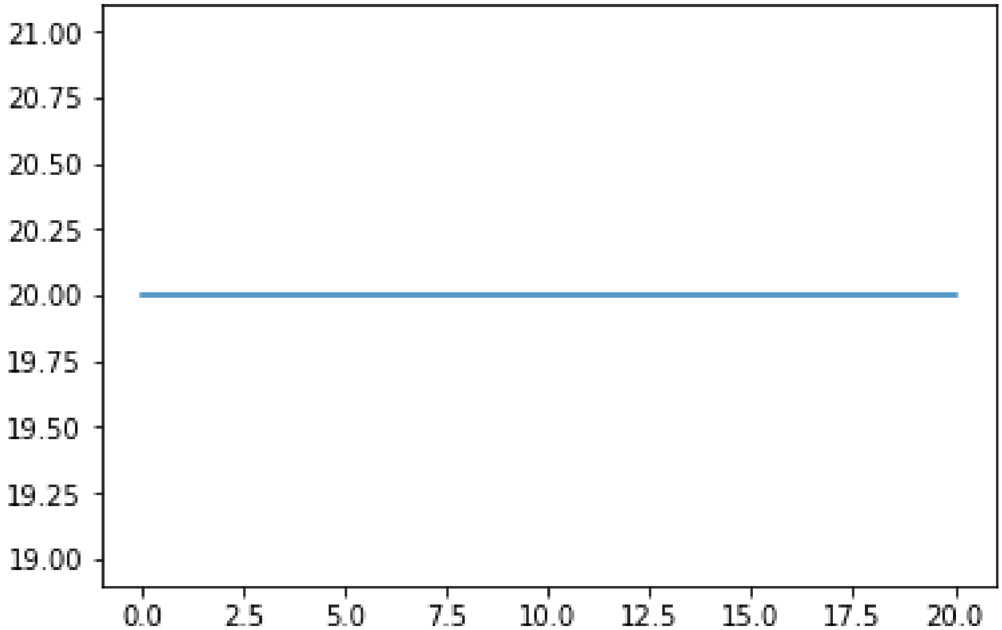
This function actually *takes another function as input*. This is a perfectly valid thing to do in Python, but I know the first time I saw code like this I found it pretty hard to wrap my head around what was going on.

Just wait until a bit later in this notebook when you'll see a function that actually `returns` another function!

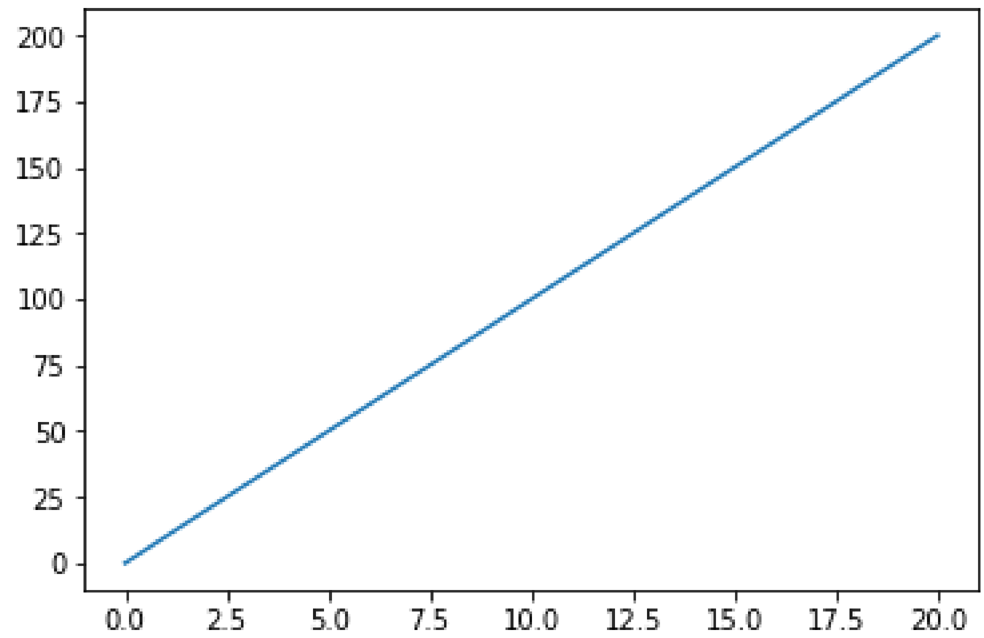
For now, let me show you other ways you can use `plot_continuous_function`.

```
In [15]: def constant_position_motion(time):
             position = 20
             return position + 0*time

         def constant_velocity_motion(time):
             velocity = 10
             return velocity * time

         def constant_acceleration_motion(time):
             acceleration = 9.8
             return acceleration / 2 * time ** 2
```

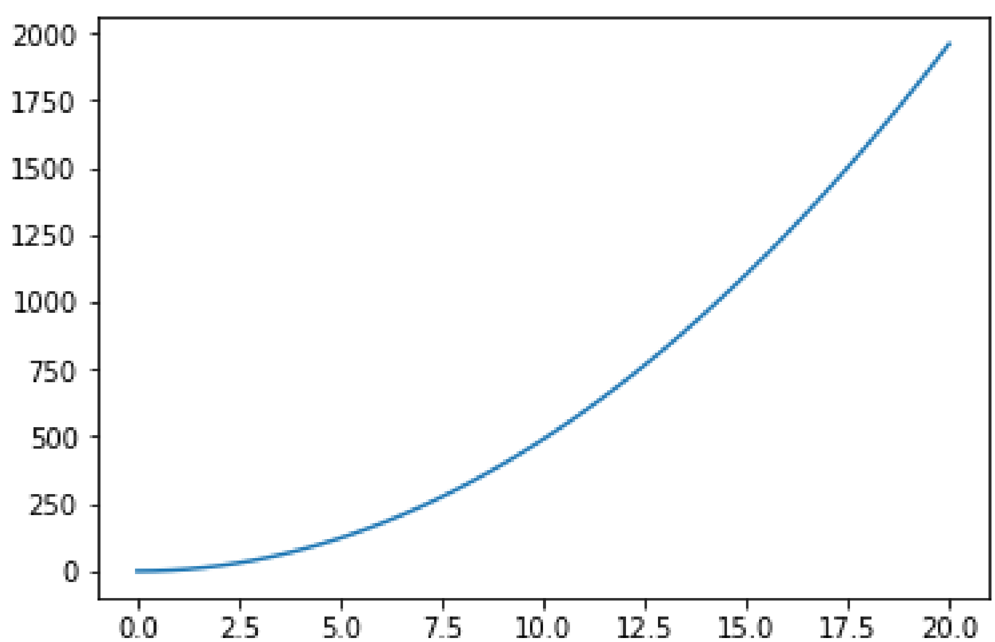
```
plot_continuous_function(constant_position_motion, 0, 20)
plt.show()
```




```
In [16]: # position vs time
         # with constant VELOCITY motion

         plot_continuous_function(constant_velocity_motion, 0, 20)
         plt.show()
```



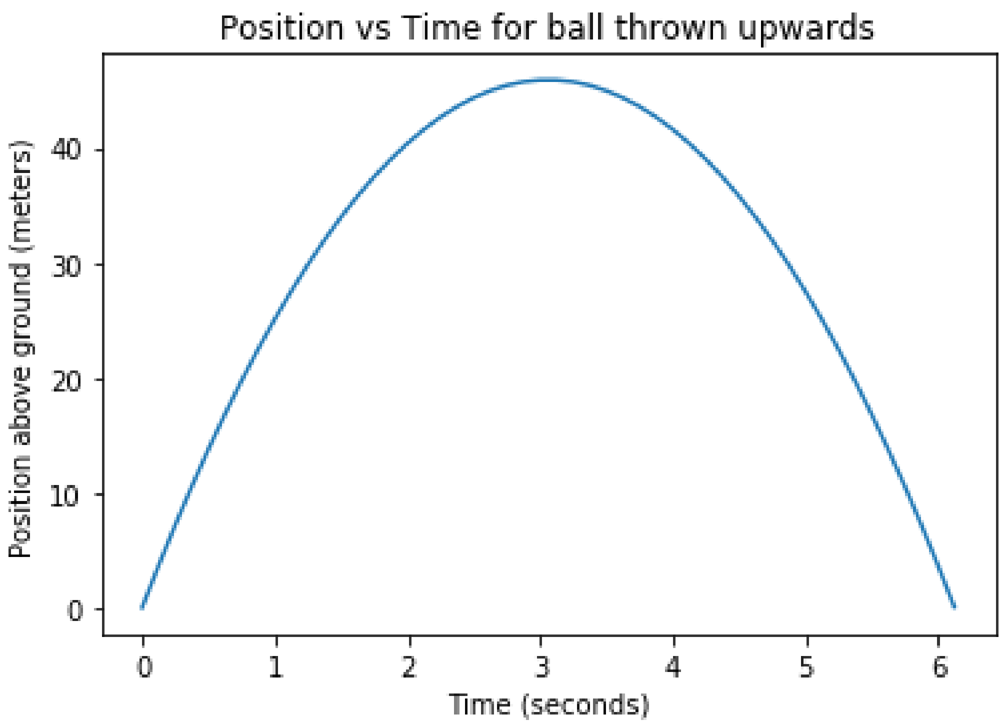

```
In [17]: # position vs time
         # with constant ACCELERATION motion

         plot_continuous_function(constant_acceleration_motion, 0, 20)
         plt.show()
```



1.6 3 - Find derivative "by hand" *at a specific point*

Let's go back to the ball-thrown-in-air example from before and see if we can find the **velocity** of the ball at various times. Remember, the graph looked like this:

```
In [18]: plt.title("Position vs Time for ball thrown upwards")
         plt.ylabel("Position above ground (meters)")
         plt.xlabel("Time (seconds)")
         plot_continuous_function(position_b,0,6.12)
         plt.show()
```



Now I would like to know the **velocity** of the ball at t=2 seconds.

GOAL - Find the velocity of the ball at t=2 seconds

And remember, **velocity is the derivative of position**, which means **velocity is the slope of the tangent line of position**

Well we have the position vs time graph... now we just need to find the slope of the tangent line to that graph AT t=2.

One way to do that is to just zoom in on the graph until it starts to look straight. I can do that by changing the `t_min` and `t_max` that I pass into `plot_continuous_function`.