Equation Solver

Unless the equation you are solving is of a certain type, it's quite likely that there is no easy method for finding solutions. Sometimes it isn't even obvious that there are any solutions. If all you can do is plug numbers in and see if they work, computers can be remarkably efficient at finding solutions for you…

Bisection halves the size of the range within which a solution must lie:

- If you have an upper bound (a number which is too large) and a lower bound (a number which is too small), use the number halfway between for the next guess.

Linear interpolation uses the size of any previous guesses to improve accuracy:

- If you have an upper bound which is way too big, and a lower bound which is only a bit too small, linear interpolation makes use of this fact, and gives a proportionate value for the next guess, much closer to the lower bound in this case.

The Newton-Raphson method uses the value and *gradient* to improve guesses:

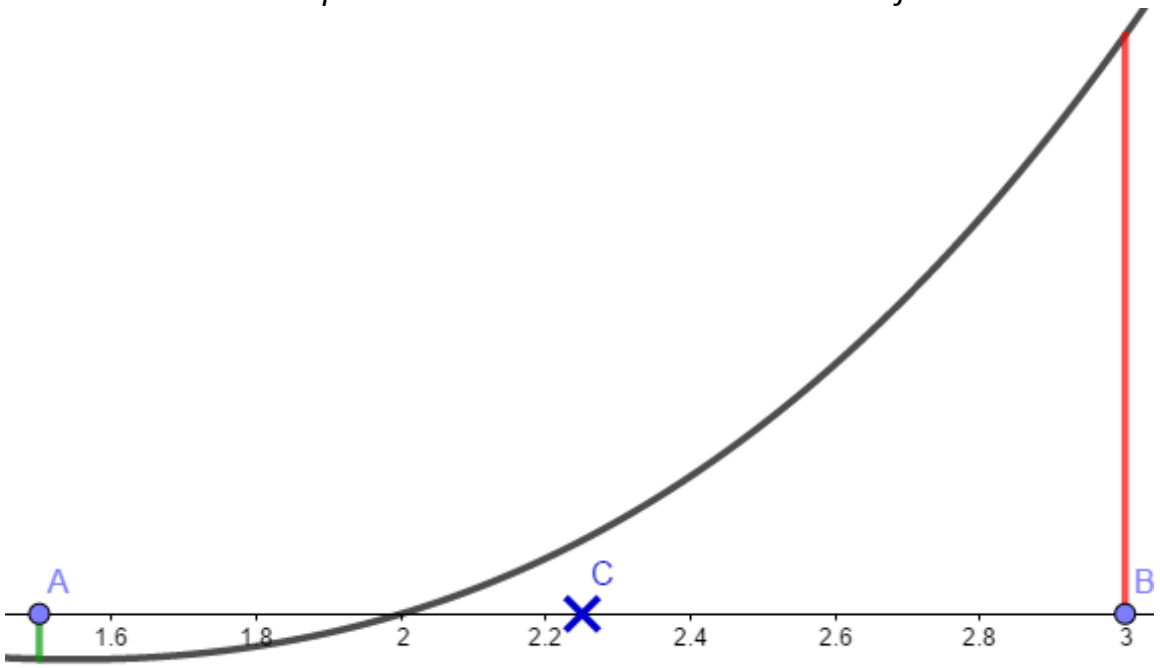
- If you have an initial guess for a root of a function, and you know both the height at that point and the gradient, you can extrapolate down a tangent for the next guess.

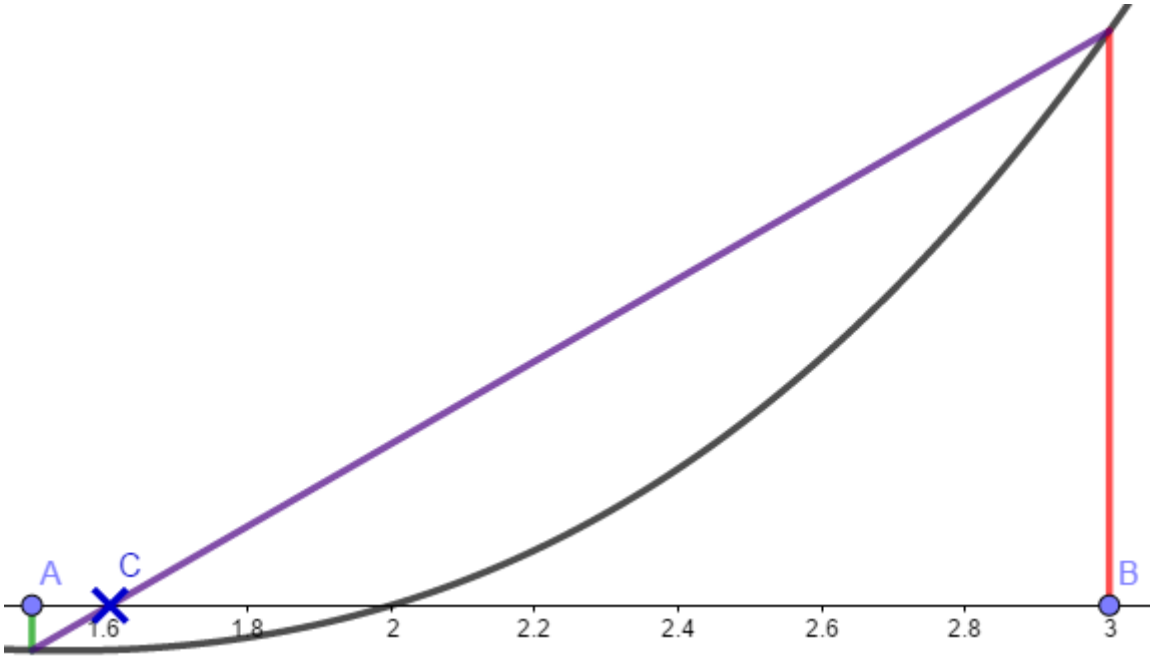
An analogy:

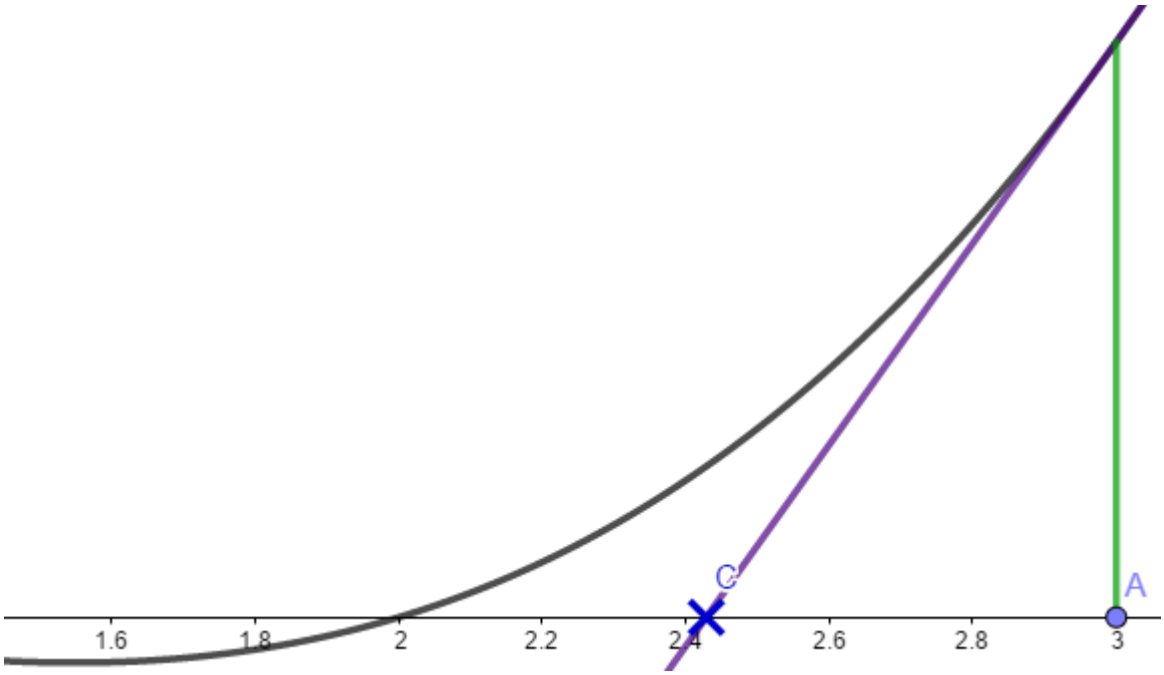
Imagine the police are tracking a suspicious shipment and want to work out exactly when it crossed the US/Mexico border. Depending on the information they have, they can have a guess at when the crossing occurred.

Method	**Known info**	**Analogy**
Bisection	Two guesses, one too large and one too small.	The shipment was in the US at 1pm, and it was in Mexico at 7pm. *Best guess: crossed the border at* _____
Linear interpolation	Two guesses, one too large and one too small, with known values.	The shipment was in San Antonio, 200 miles north of the border, at 11am, and by 5pm it was in Monterrey, 100 miles south of the border. *Best guess: crossed the border at* _____
Newton-Raphson	One guess, with known value and gradient.	The shipment was clocked doing 65mph, 25 miles north of the border, at 2:50pm. *Best guess: crossed the border at* _____

Can you match each visual representation with the correct method from the three methods?







Equation Solver *with Python*

To start with, we can define a mathematical function in Python:

```
1  def f(x):
2      return x ** 3 - x + 3
3
4  for x in range(-5, 5):
5      print(f"f({x}) = {f(x)}")
```

This function takes a single input, **x**, and produces an output equal to $x^3 - x + 3$.

We can get a feel for the function by printing a range of output values.

***f-strings** can be very confusing sometimes – there are a lot of brackets and f's around to keep track of! This print statement will give results like **f(-2) = -3**.*

Note that, when finding solutions to equations mathematically, it is usual to rewrite the equation so it is in the form $f(x) = 0$. Then finding solutions to the original equation is equivalent to finding roots to the function f. In the example above, finding solutions to $x^3 = x - 3$ is the same as finding an x for which $f(x) = 0$ where $f(x) = x^3 - x + 3$.

The key thing to look for when solving equations is a range within which a root must lie. If we have a *continuous* function (that is, no pesky asymptotes or other discontinuities), if it gives negative results at one point and positive results at another, it must pass through zero at least once in between. Use a **while** loop to find a pair of integers either side of a root:

```
1  def f(x):
2      return x ** 3 - x + 3
3
4  x0 = -100
5  x = x0
6  while f(x) * f(x0) > 0:
7      x += 1
8  print(f"f({x-1}) = {f(x-1)}")
9  print(f"f({x}) = {f(x)}")
```

This code initiates a variable **x0**, the first guess for a root. The **while** loop uses a mathematical trick to spot when the sign changes. As long as the signs of $f(x)$ and $f(x_0)$ are the same, their product will be positive, so keep going up.
Once a sign change is detected, print out the latest two results.

Now we have our upper and lower limits, we can use a bisection function to home in on the result to whatever level of precision we require:

```
a = x - 1
b = x
while abs(f(a)) > 0.0000001:
    new = (a + b) / 2
    if f(a) * f(new) < 0:
        b = new
    else:
        a = new
print(f"f({a}) = {f(a)}")
print(f"f({b}) = {f(b)}")
print(f"A root lies between {a} and {b}")
```

First, we set our two bounds **a** and **b** (based on the values found in the first part of the program).
Next, we determine the level of precision we need, using the **abs** function (which gives the size, but ignores the sign).
Until our lower bound is close enough, we repeatedly improve our guess using the same trick to check for a sign change.

Try making your own version of this, and see how well it adapts to different functions. The version shown above is by no means perfect. For a start, it assumes there will be a root for some $x > -100$. How could you adapt your function to hunt for roots when you don't even know if they exist? How could you find multiple roots? Can you build in a counter to keep track of the number of iterations required to find roots to the required precision?

Linear interpolation

If we know that $f(a) < 0$ and $f(b) > 0$, but we *also* know the values of these results, our best guess for the root is where the line between $\big(a, f(a)\big)$ and $\big(b, f(b)\big)$ cuts the x-axis:

$$m = \frac{f(b) - f(a)}{b - a} \quad \Longrightarrow \quad y - f(a) = \left(\frac{f(b) - f(a)}{b - a}\right)(x - a)$$

This line crosses the x-axis when $y = 0$:

$$-f(a) = \left(\frac{f(b) - f(a)}{b - a}\right)(x - a)$$

Rearranging to make x the subject:

$$x = a - \frac{b - a}{f(b) - f(a)} f(a)$$

Can you write a program that finds a root for the equation $x^3 = x - 3$ using linear interpolation? We can use the fact that there is a root between -1 and -2. Build in a counter to keep track of how many times the process needs to run to generate an answer accurate to 8 decimal places. Compare it to the result for bisection.

Newton-Raphson

For this method, we have an initial value, x_0, and we have both the function $f(x)$ and the gradient function $f'(x)$ so that for any value we can find both the height of the curve and the gradient of the curve.

If we know a point on a curve and the gradient at that point, we can find the equation of the tangent line through that point:

$$y - f(x_0) = f'(x_0)(x - x_0)$$

Since we are interested in where this cuts the x-axis, set $y = 0$ and rearrange for x:

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Can you make a function that uses the Newton-Raphson process to find solutions to the equation $x^3 = x - 3$? Start from $x_0 = -1$ and see how many steps it takes to reach a solution to the same level of precision as before.

What's next?

The best equation solvers often use a combination of the above methods. Newton-Raphson, for instance, has problems close to stationary points, so often bisection is used to get close to a root, then Newton-Raphson takes over (or, if we don't know the gradient function, linear interpolation) to speed up the process as we get increasingly close.