Root Finding

Let us consider the the following equation

$$\cos x = x$$

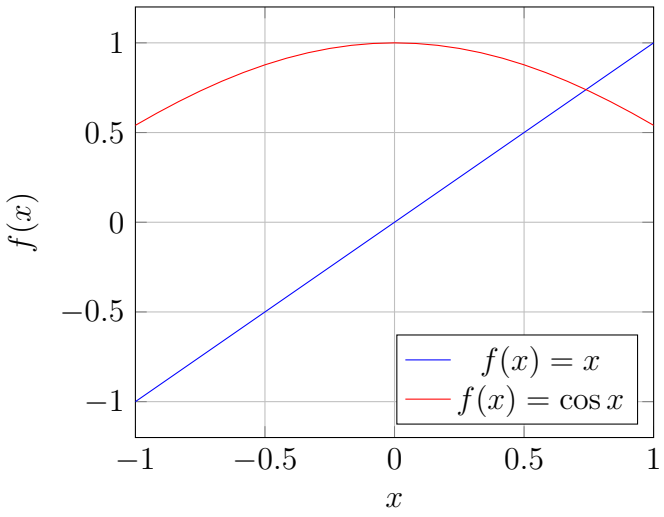
If you try to solve this, it's most likely that you will get in to an infinite loop of trying to do something like

$$\cos x = x \iff \arccos x = x$$

which does not bring you anywhere closer to the real answer. [1]

All the math you learned so far are about function that you can solve by hands. Those equation are sugar-coated. In fact, most problems in real life will give you equations you can't solve by hands.

OK, after a while you may think it might be a good idea to plot it out and see where the two graphs intersect after all that is what it means for a value to be a solution of an equation. The graph you got will look something like this. Of course, you can zoom in as much as you want. This is called graphical method.

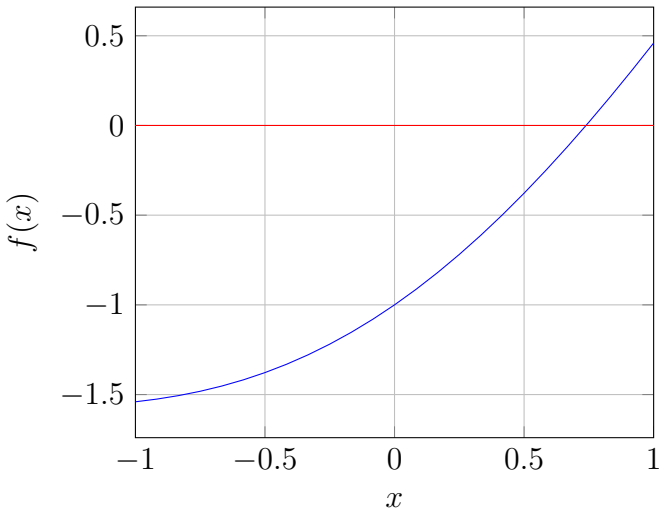


Let us look at this problem from a bit different view. The same problem as above can be stated as finding the solution of

$$x - \cos x = 0$$

Then you have one graph of $f(x) = x - \cos(x)$ and the solution to the above equation is the x-intercept.

[1] There is actually a way to get a really close answer by using what we learned last week but I'll leave that to you as a challenge.

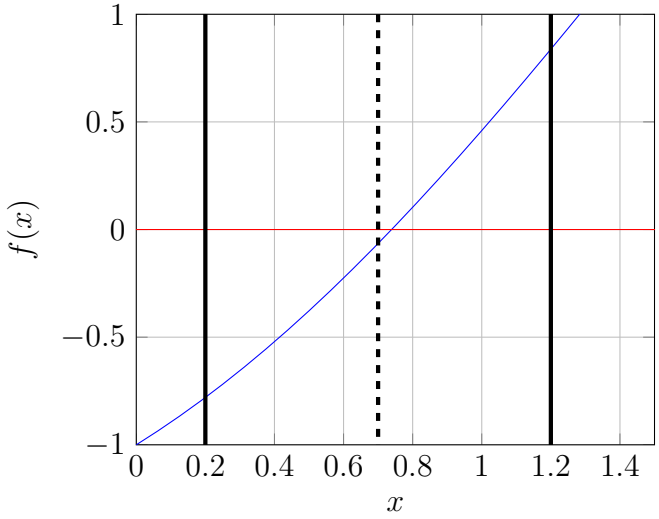


So, instead of using tailored made algorithm for each equation, we can use algorithm for finding x-intercept for any equation we want. We just need to transform it such that it looks like

$$f(x)) = 0$$

Bisection Method

The first mehtod we want to investigate is called the bisection method. The concept is simple if you notice on thing that if you have two point a and b in which $f(a)$ and $f(b)$ has opposite sign then there is a point $c \in [a, b]$ such that $f(c) = 0$.



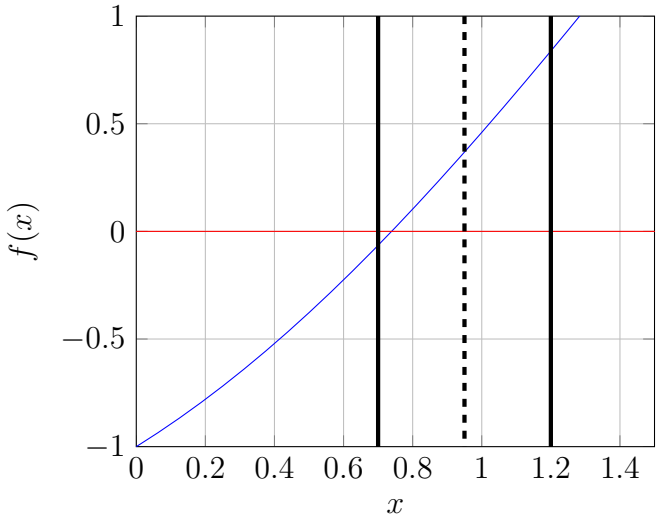
For example in the example above you can see that

$$f(0.2) < 0 \text{ and } f(1.2) > 0,$$

which means that if the function f is continuous it must cut through the x-axis somewhere in between.

If we stop here and make our guess as the center value $x = 0.7$. Then one thing we know is that the real answer will be at most $(1.2 - 0.2)/2 = 0.5$ away from our guess of $x = 0.7$.

There is no reason why we should stop there. We know that at $x = 0.7$, $f(0.7) < 0$. We could use $x = 0.7$ along with $x = 1.2$ as the new set of bracket.



Let us look at the bound on error, it's half of what it was before!. We normally write this as

$$e_{n+1} = \frac{e_n}{2}$$

where e_1 can be found from the original bound. This convergence behavior is called *linearconvergent* for the fact that the error for the next iteration is linear to the error in the previous iteration.

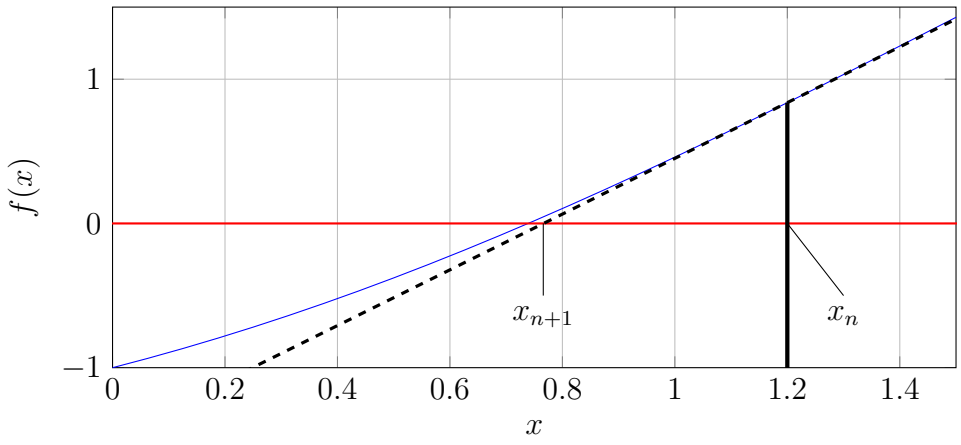
It should be noted that this is the "bound" on error that is getting smaller. The actual error might be bigger or smaller. As you can see in the example, the bound is smaller but you are getting further away from the answer.

This method is super easy to implement in reallife too.You can determine the optimal time to make poach egg using this method.

Newton's Method

Let us do something a bit smarter than bisection. This is the method use for most applications because it converges very fast. The idea behind the method is that if we start our guess at certain point x, we can then make a tangent line and use the x-intercept of the tangent line as our new guess.

You may ask why using the intercept of the tangent line. Let us understand this using picture. Suppose that our initial guess is at $x = 1.2$



You can see that x-intercept is a really good guess. It is a good guess because the function behave almost like a straight line that is it doesn't have much of second derivative. We will explore this fact later.

Before we go any further, let figure out how to find the the next guess. First, we need to find the tangent line at x_n. We know that the tangent line

1. Pass through the point $(x_n, f(x_n))$.
2. Has slope of $f'(x_n)$.

Recall the general form of a straight line

$$y = mx + c$$

We already know $m = f'(x_n)$. So we need to solve for c from the first condition

$$f(x_n) = f'(x_n)x_n + c,$$

This means

$$c = f(x_n) - f'(x_n)x_n$$

But we want the x-intercept that's when $y = 0$

$$0 = mx_{\text{intercept}} + c \Longrightarrow x_{\text{intercept}} = -\frac{c}{m}$$

This means that the our new guess(x_{n+1}) which is the x-intercept of our tangent line is given by

$$x_{n+1} = -\frac{f(x_n) - f'(x_n)x_n}{f'(x_n)}$$
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Similar to the idea of bisection there is no reason to stop here. We could use the x-intercept as the new guess and keep repeating.

Let us understand the convergence behavior of this algorithm. Let us first, recall Taylor expansion we learned.

$$f(x_{n+1}) = f(x_n) + f'(x_n)(x_{n+1} - x_n) + \frac{f''(\xi)}{2!}(x_{n+1} - x_n)^2 \tag{1}$$

where $\xi \in (x_{n+1}, x_n)$.

Truncating the series gives

$$f(x_{n+1}) \approx f(x_n) + f'(x_n)(x_{n+1} - x_n)$$

We want to find x_{n+1} such that when we plug it in to our function it gives us 0. That is our solution. So from our approximation we can just replace $f(x_{n+1})$ with 0 and solve for x_{n+1} in terms of what we know. This yields

$$0 = f(x_n) + f'(x_n)(x_{n+1} - x_n) \tag{2}$$

which means

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

This is exactly what we have from drawing tangent line.

But now we want to know how the bound on error goes as we iterate the algorithm. This can be done by replacing $x_(n+1)$ in Equation by the real root of equation x_r. Let us do that and see what happens.

$$f(x_r) = f(x_n) + f'(x_n)(x_r - x_n) + \frac{f''(\xi)}{2!}(x_r - x_n)^2$$

and since $f(x_r) = 0$ since it's the real root of equation we have

$$0 = f(x_n) + f'(x_n)(x_r - x_n) + \frac{f''(\xi)}{2!}(x_r - x_n)^2 \tag{3}$$

First notice that

$$e_n = |x_r - x_n|$$

So our goal now would be to find the relation between of the first two terms and the $e_n + 1 = |x_r - x_{n+1}|$.

We can do this by subtract Equation 3 by Equation 2. Let us see what happens

$$
\begin{aligned}
0 =& f(x_n) + f'(x_n)(x_r - x_n) + \frac{f''(\xi)}{2!}e_n^2 - f(x_n) - f'(x_n)(x_{n+1} - x_n) \\
=& f'(x_n)(x_r - x_{n+1}) + \frac{f''(\xi)}{2!}e_n^2
\end{aligned}
$$

Thus,

$$
\begin{aligned}
f'(x_n)(x_r - x_{n+1}) &= -\frac{f''(x_n)}{2!}e_n^2 \\
|x_r - x_{n+1}| &= \left|\frac{f''(x_n)}{2f'(x_n)}\right| e_n^2 \\
e_{n+1} &= \left|\frac{f''(x_n)}{2f'(x_n)}\right| e_n^2
\end{aligned}
$$

Let us understand the last expression. This kind of behavior is called *quadratic convergent*. Ignore the constant in front of the right hand side for the moment and let us appreciate how fast this convergent is. Suppose that you are now 0.001 away from the real answer. The bound on error you get from the next one would be$0.001^2 = 0.000001$. Three more correct digit for 1 more iteration.

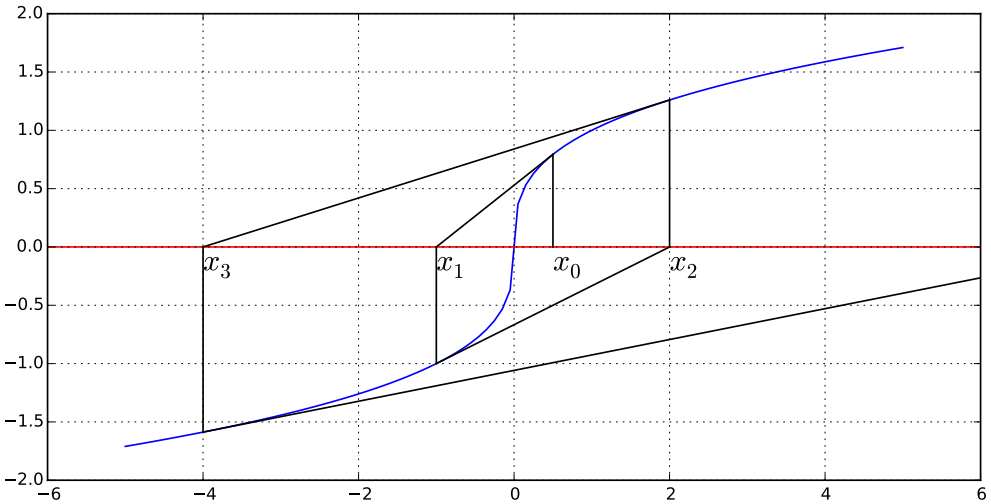
If we compare this against what we got from bisection where the error half every step. It will take you 3 to 4 more iteration to get the next digit while the correct number of digit for newton's method double every iteration.

Caveat

Even though Newton's method converges very fast it comes with the caveat. It doesn't really guarantee to converge. Let us consider the function

$$f(x) = \sqrt[3]{x}$$

The root of this equation is $x = 0$. However, if you try to apply newton's method with any guess you will run in to the following situation of the more iteration you do the further away you are from the answer.



The cause of this behavior is actually the factor in front that we ignore earlier when we show quadratic convergent behavior of this algorithm. Newton method does not guarantee convergence unlike bisection method. We will not go into too much details about this behavior.

So, to summarize Newtons' method. It converges really fast(when it does).

Fun Thing

This is an legendary piece of code in Quake3 arena for calculating inverse square root($1/\sqrt{x}$). The function is needed for calculating normal vector for surfaces. This funciton gets called a lot. So, it is crucial that the calculation is super fast. Taking inverse and calling function squareroot is very slow cpu-wise. Now you are equipped to understand this piece of code. Try to read the following code and figure out how it does the magic.

```
float Q_rsqrt( float number )
{
   long i;
   float x2, y;
   const float threehalfs = 1.5F;

   x2 = number * 0.5F;
   y  = number;
   i  = * ( long * ) &y;              // evil floating point bit level hacking
   i  = 0x5f3759df - ( i >> 1 );      // what the fuck?
   y  = * ( float * ) &i;
   y  = y * ( threehalfs - ( x2 * y * y ) ); // 1st iteration
   // y  = y * ( threehalfs - ( x2 * y * y ) );// 2nd iteration, can be removed

   return y;
}
```

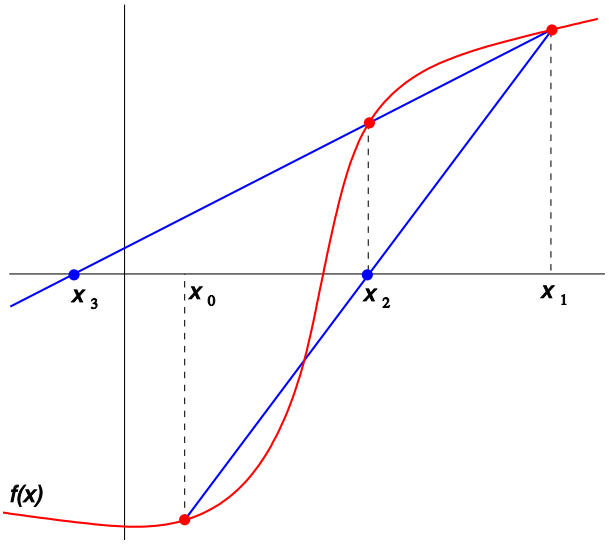
https://en.wikipedia.org/wiki/Fast_inverse_square_root

Secant Method

One of the disadvantage of Newton's method is having to find the first derivative analytically. We can however fix that problem at the cost of a bit slower convergent.

The idea is the following. Suppose you start with 2 guesses x_0 and x_1 (not necessarily with the opposite sign). The next guess x_2 can be found by the x-intercept of the straight line between the two.

The process can be repeated by using x_1 and x_2 and then find the x-intercept (x_3) again.



The convergent of this method(I won't prove it here) is $e_{n+1} = e_n^{\alpha}$ where α is the golden ratio you learned in discrete math.