18. Finding roots numerically

Solving equations is a popular task in mathematics. The first type that one meets is algebraic equations. We start by learning how to solve linear equations of the form $ax + b = c$, and by the time we go to high school, we also master the art of solving quadratic equations. Remarkably, there is no more progress afterwards. We do learn some tricks for solving equations with powers a^k and logarithms, but the problems that we applied them to were carefully prepared, just a small change would put them beyond our means. Surprisingly, when it comes to reliably solving various types of equations, we did not progress beyond elementary school knowledge. Perhaps we should clarify that by solving an equation we mean writing its solution using an algebraic formula.

The reason for this lack of progress is simple: algebraic equations are tough. Speaking of polynomials, there are formulas for roots of polynomials of degree three and four, but they are rather unpleasant. Worse, mathematicians proved that it is impossible to have formulas for roots for polynomials of degree five and more. Once we move away from polynomials, things become even worse and even simple equations can defeat us.

Consider the equation $\cos(x) = x$. Just imagining the graphs of $\cos(x)$ and $y = x$ we readily conclude that they intersect, and this equation therefore has some solution. However, we do not have any tool how to get to it in an algebraic manner. In fact, it can be proved that this solution cannot be expressed by an algebraic formula. Consequently, no matter what algebraic tricks we try, we will never reach it. The same story is true for the equation $e^{-x} = x$, which also looks quite simple.

On the other hand, equations play an important role in applications, so we do have to be able to somehow identify their solutions. We therefore turn to numerical mathematics. In order to simplify our work, we start by unifying the setting. Every algebraic equation can be easily rearranged by putting all terms to the left-hand side of the equation, leaving just zero on the right. In other words, it is enough to know how to solve numerically equations of the form $f(x) = 0$. For solutions of such equations we have a special name.

Definition 18.1.
By a **root** of a function f we mean any number r such that $f(r) = 0$.

Some people also say **zero of a function** instead.

How do we find roots of functions numerically? There are several problems one faces. To start with, we are not even able to recognize whether we found a root or not. When a computer tells us that $f(x) = 0$ for some x, then it simply means that the value is smaller than the machine epsilon (see discussion in section 3c). This is useful to know, but it is not such a big problem in this context, because we do not expect to find the actual root anyway.

As is usual in numerical analysis, our aim is to provide some approximation of the root whose distance from the actual root is smaller than some prescribed tolerance $\varepsilon > 0$, which means that we want to control the absolute error $E_r = r - \hat{r}$ of our approximation $\hat{r}$ of r. In fact, one would expect that we focus on relative error, as it relates to reliability of our approximation, but it is customary to work with absolute error here. Typically we want $|E_r| < \varepsilon$, but asking for $|E_r| \leq \varepsilon$ is also possible and there is just a formal difference between the two. This will be the basic setting of this and the following chapters.

It is useful to realize that the functions that we encounter in numerical mathematics need not be given by formulas like those that we meet at school. It is possible that the value has to be measured, in particular it may be an outcome of an experiment. So when we say "substitute a number a into a function f", it may also mean that we start an experiment with some setting set to a and then its outcome determines $f(a)$. If $f(a)$ denotes the mass of tomatoes harvested from a hothouse when the temperature is set to a, then a simple evaluation can last a full season. This is a rather extreme example, but it is good to remember that evaluating a function may not be so easy as we are used to.

We start with a simple question. Given some function f, how do we recognize that it actually does have some root? Even this question we cannot answer to our full satisfaction. There is no procedure that would reliably recognize the existence of a root. But there are some approaches that work reasonably well most of the time, which is a familiar situation in numerical analysis.

So what are the approaches? In the best case we have some information about the shape of the given function. For instance, we may be able to plot its graph. However, this is not reliable. Plotting a function essentially means that we sample it at specific points and then create a curve based on these values. However, imagine a function that is seemingly always positive, but has one really narrow spike reaching down below the x-axis. Unless we are lucky and sample this function in the right place, our plots will never reveal the existence of this spike and we will think that it has no root. Still, if we are able to plot our function, then this is the best start.

Sometimes the function is given by a formula that can be investigated using tools from calculus. Or the function may come from some real-life application and the story that accompanies it gives reason to believe in existence of such a root.

And when all fails, we can simply start sampling the function, that is, start substituting numbers into it (perhaps randomly) and see what happens. What we want to see is change in signs of the values as we go from one place to another. A layperson would then conclude that there must be a root somewhere inbetween, but those who took calculus know that in the world of mathematics there are also weird functions, and that we have to add a crucial assumption to be able to come to this conclusion.

Theorem 18.2.
Let f be a function continuous on some interval $[a,b]$. If the numbers $f(a)$, $f(b)$ have opposite signs, then there must be a root of f in (a,b).

The assumption on signs can be efficiently expressed like this: $f(a)\cdot f(b)<0$. The theorem itself is just an easy consequence of the classical Intermediate value theorem from calculus.

Unfortunately, this theorem is far from perfect. When it happens that the two signs are equal, then we simply cannot make any conclusion, as there may or may not be a root between a and b. Typically we would try to substitute more numbers, but unfortunately there are roots that simply cannot be identified in this way. Imagine the classical parabola, the graph of $f(x)=x^2$. It has a root at the origin, but no matter how hard we try, we never achieve opposite signs when sampling. This shape is typical of roots of even multiplicity. This is the first but not the last time that we see roots of higher multiplicity causing troubles.

However, in many cases this test works fine, and it is often the best tool that we have available.

Example 18.a: Consider the function $f(x)=x^3-x-10$. Does it have a root somewhere?

We try some numbers. My favorite is zero:

$$f(0)=-10.$$

We do not actually care about the exact value, just that it is negative. Now we would like to see some positive value. Since we know how the powers behave, we feel that some larger number should do. Say,

$$f(5)=110.$$

Yes, there must be some root between 0 and 5.

Note that we could deduce the existence of the root also in other ways. For instance, we know that $\lim\limits_{x\to\infty}\big(f(x)\big)=\infty$ and $\lim\limits_{x\to-\infty}\big(f(x)\big)=-\infty$, so as a continuous function it must have a root

somewhere.
△

When we identify a position of some root within a (finite) interval, we say that we **bracketed** it. However, this is not enough, we want to approximate it with a given precision.

The general approach is to try some initial guess, call it x_0. If it is not good enough, we try to make a better guess, call it x_1, using information gained from x_0. Of course, it is a sophisticated guessing. We continue in this way, constructing numbers x_k, until we find one whose absolute error as approximation for the desired root is within the specified tolerance.

Since different customers can have different tolerances. we are interested in procedures that are capable of producing arbitrarily good approximations, in other words, we should be able to let them run as long as needed, until they eventually succeed.

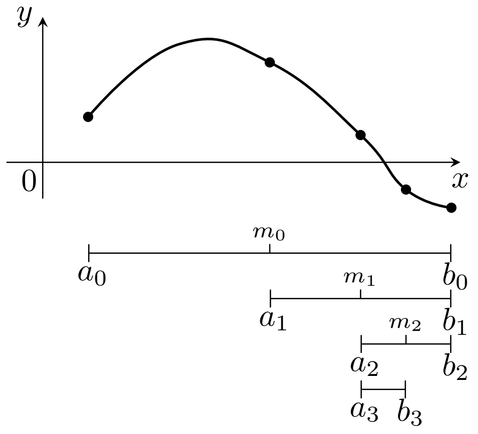
The general setting thus is that we will be interested in **iterative methods** that create (potentially infinite) sequences of numbers x_k. A good process (method) should be able to provide approximations for some root of arbitrary precision, which essentially means that the sequences such a method produces should converge (in the usual mathematical sense) to the desired roots.

There are many iterative methods for finding roots, and they can be grouped according to their nature. We will explore here two major approaches by introducing a representative method for each.

18a. The bisection method (bracketing)

We have the following situation: for a given (continuous) function f we found points a, b such that values f have opposite signs there. We know that this tells us that there must be some root in the interval (a, b). Can we localize it closer?

Here is a natural idea: We take the middle of this interval, call it m, and substitute into f. If the result is zero, then this $m = \frac{1}{2}(a + b)$ is very likely a root (see the remark above about computer computing). However, the chances are low. It is more likely that $f(m)$ is not zero, so it must be either positive or negative. Since the numbers $f(a)$, $f(b)$ have different signs, $f(m)$ cannot obviously have the same sign as both of them. Consequently, either the pair $f(a)$, $f(m)$ has opposite signs, then there is a root in the interval (a, m), or the pair $f(m)$, $f(b)$ has opposite signs and we have a root in (m, b). Either way, we localized the root in an interval that is smaller than the original interval (a, b), so we narrowed it down.



This looks like a good idea, so we continue, splitting the new interval again and choosing the half with opposite signs. To keep track, we introduce names. The starting interval will be called $[a_0, b_0]$ with midpoint m_0, the next one $[a_1, b_1]$ with midpoint m_1, and eventually we construct a (theoretically infinite) sequence of intervals $[a_k, b_k]$ with midpoints m_k, each has half the length of the previous one, and we always have some root in every $[a_k, b_k]$, because we have opposite signs of f at the endpoints.

We wanted to focus on iterative procedures that produce sequences of approximations. To fit withing this category, our procedure should produce some official root approximation at each stage $[a_k, b_k]$. We know that the root is somewhere within this interval, so the natural output for this stage would be the midpoint m_k.

When do we stop? If we know that there is a root in an interval $[a_k, b_k]$ and we declare its midpoint m_k as the official output, then the largest possible error (absolute error) is $\frac{1}{2}|b_k - a_k|$. So we simply wait for this number to drop below the prescribed tolerance ε.

We just developed one of the basic root finding methods.

Algorithm 18a.1.
⟨bisection method for finding root of f⟩
Given: a function f continuous on interval $[a,b]$ and a tolerance ε.
Assumption: $f(a)$ and $f(b)$ have opposite signs.

```
0. Set $a_0 = a$, $b_0 = b$. Let $k = 0$.
1. Assumption: $f(a_k)$ and $f(b_k)$ have opposite signs.
Let $m_k = \frac{1}{2}(a_k + b_k)$.
2. If $f(m_k) = 0$ or $\frac{1}{2}|b_k - a_k| < \varepsilon$ then the algorithm stops, output is $m_k$.
Otherwise:
If $f(a_k)$ and $f(m_k)$ have opposite signs, set $a_{k+1} = a_k$, $b_{k+1} = m_k$,
    increase $k$ by one and go back to step 1.
If $f(m_k)$ and $f(b_k)$ have opposite signs, set $a_{k+1} = m_k$, $b_{k+1} = b_k$,
    increase $k$ by one and go back to step 1.
```
△

Example 18a.a: Consider the function $f(x) = x^3 - x - 10$. Above we identified the interval $[0,5]$ where some root lies. We start the bisection method:
$a_0 = 0$, $b_0 = 5$, we recall that $f(0) < 0$ and $f(5) > 0$.
We find the midpoint $m_0 = 2.5$ and substitute into the function: $f(2.5) = 3.125$, this is positive.
We did not find the root, so we pass to a smaller interval. Since the sign of f changes between 0 and 2.5, the next iteration should be $a_1 = 0$, $b_1 = 2.5$ and the output is $m_1 = 1.25$.
We substitute the midpoint m into the function: $f(1.25) = -9.296875$, this is negative.
We did not find the root, so we pass to a smaller interval. Since the sign of f changes between 1.25 and 2.5, our next iteration will be $a_2 = 1.25$, $b_2 = 2.5$. If we decided to stop here, the official output would be $m_2 = 1.875$ and there is definitely some root within the distance $0.625 = \frac{1}{2}(2.5 - 1.25)$.
△

What are the main properties of this method?
- The bisection method is reliable.
- We have a good control over the error.
- The bisection method is slow.

We will now address these properties, and start with the second one.
Consider a certain stage k in the algorithm. There must be some root in $[a_k, b_k]$, we denote it r. We already observed that the official output m_k has its absolute error $|E_k| = |r - m_k|$ at most $\frac{1}{2}|b_k - a_k|$. This is a relatively rare occasion in numerical analysis, it is a big bonus to have an upper bound for the error that uses data that we actually have available.
There is a group of root-finding methods that are called **bracketing methods**. They all work with nested intervals that keep the root inside, and the control over the position of the root is their major advantage.
Having established the control over the error, we move on. For simplicity, we denote $e_k = \frac{1}{2}(b_k - a_k)$, this is our upper estimate for $|E_k|$. From the way the intervals are constructed we easily see that $e_k = \frac{1}{2}e_{k-1}$, this recursive equality can be applied repeatedly and we obtain

$$|E_k| \le e_k = \frac{1}{2}e_{k-1} = \frac{1}{2}\frac{1}{2}e_{k-2} = \cdots = \frac{1}{2^k}e_0 = \frac{1}{2^{k+1}}(b_0 - a_0).$$

This shows that $E_k \to 0$, so the bisection method is capable of producing root approximations of arbitrary precision. In other words, the sequence $\{m_k\}$ produced by the bisection method always converges to a root.
This brings us to reliability. Can anything go wrong with this process? And the answer is in the negative. Given some tolerance $\varepsilon > 0$ and an initial interval $[a,b]$ on which the given function changes sign, the bisection method necessarily leads us to suitable approximation.

Could there be numerical troubles? There is only one calculation that is a part of the bisection method itself, namely the calculation of the midpoint. Surprisingly enough, we can run into trouble there. If the numbers a_k, b_k get really close, the formula $\frac{1}{2}(a_k + b_k)$ can actually produce a number that is outside of the interval $[a_k, b_k]$!

For instance, consider the interval $[5.3, 5.4]$ and imagine that we work with a two-digit precision. Then $5.3 + 5.4 = 10.7$ gets rounded up to 11, so we get the midpoint $m = 5.5$. This would be fatal, which explains why in implementation the formula $m_k = a_k + \frac{1}{2}(b_k - a_k)$ is preferred, because it is not susceptible to such problems.

Having fixed this, there is nothing else that can go wrong with the bisection method. Of course, we may run into numerical problems when evaluating f, but that is another story, and if that happens, then it will trouble us regardless of what method we use.

In conclusion, apart from possible problems with f, this method is absolutely reliable.

It remains to address its speed. Above we tried to approximate a root of $x^3 - x - 10$. How many iterations would it take before our error approximation drops below a prescribed tolerance? Our requirement is that $|E_k| < \varepsilon$, but we do not have access to the errors. However, we have an upper bound, so we ask for $e_k < \varepsilon$ instead. The inequality $|E_k| < e_k$ shows that we then have $|E_k| < \varepsilon$ automatically.

When will $e_k < \varepsilon$ become true? We can substitute for e_k the formula that we deduced above to obtain $\frac{1}{2^k} e_0 < \varepsilon$. We can solve this inequality for k: $k > \log_2\Bigl(\frac{e_0}{\varepsilon}\Bigr)$. Substituting for e_0 and passing to a slightly large k to simplify the formula we see that $N = \Bigl\lceil \log_2\Bigl(\frac{|b_0 - a_0|}{\varepsilon}\Bigr)\Bigr\rceil$ iterations should do.

This is another advantage of the bisection method: It is predictable. Note that the formula makes sense: The smaller the tolerance, the larger the number of steps. Which brings us to the question of speed: Is this N reasonable or is it too much?

We will shortly develop a general theory for comparing speeds of root-finding methods (section 18d), for now we will do with intuitive reasoning. The natural approach is to ask how the precision of our approximation improves with each step. In fact, we have an answer already, the formula $e_{k+1} = \frac{1}{2} e_k$ tells us that the (upper estimate for) the error gets halved at each iteration. To get a feeling for this result we switch our point of view.

Intuitively, a common measure of a number's reliability is how many valid digits are determined right. We already commented before that the notion of "right digits" is somewhat fuzzy, but it is a useful aid for our intuition. We related this to the relative error, in particular we observed that, roughly speaking, improving accuracy by one digit is related to decreasing its relative error by a factor of ten. How much work does it take if we use the bisection method?

The relation $e_{k+1} = \frac{1}{2} e_k$ seems to lead in the right direction. The decrease by 2 is not enough to gain a new valid digit, but more iterations may help:

$$e_{k+3} = \frac{1}{2^3} e_k = \frac{1}{8} e_k,$$
$$e_{k+4} = \frac{1}{2^4} e_k = \frac{1}{16} e_k.$$

We can see that the desired improvement by the factor 10 is expected to happen somewhere between the third and the fourth iteration. One may argue that we worked with the absolute error here, but just dividing these inequalities by the actual error r yields exactly the same relationships for the relative errors, starting with the basic formula

$$\frac{e_{k+1}}{r} = \frac{1}{2} \frac{e_k}{r}.$$

This supports mathematically that when comparing errors of iterations, it is enough to focus on absolute errors and the conclusions will be valid also for relative errors.

We will now test this observation. Before we make a run of the bisection method, let us make a more practical (and less mathematical) observation. Once the absolute errors drop below 1, we can judge the precision of our approximation by the number of leading zeros in the absolute error. A new leading zero means a new valid digit gained. Again, this relationship is not completely precise, but as a rough guide it will do, and it allows us to judge how things go easily by looking at the outputs.

Example 18a.b: We return to $f(x) = x^3 - x - 10$, considered on the interval $[0, 5]$. We applied the bisection method, with tolerance $\varepsilon = 0.0001 = 10^{-4}$. In the following chart we list the intervals $[a_k, b_k]$, midpoints m_k and test values e_k:

k	a_k	b_k	m_k	e_k
0	0.00000	5.00000	2.50000	2.50000
1	0.00000	2.50000	1.25000	1.25000
2	1.25000	2.50000	1.87500	0.62500
3	1.87500	2.50000	2.18750	0.31250
4	2.18750	2.50000	2.34375	0.15625
5	2.18750	2.34375	2.26562	0.07812
6	2.26562	2.34375	2.30469	0.03906
7	2.30469	2.34375	2.32422	0.01953
8	2.30469	2.32422	2.31445	0.00977
9	2.30469	2.31445	2.30957	0.00488
10	2.30469	2.30957	2.30713	0.00244
11	2.30713	2.30957	2.30835	0.00122
12	2.30835	2.30957	2.30896	0.00061
13	2.30835	2.30896	2.30865	0.00031
14	2.30865	2.30896	2.30881	0.00015
15	2.30881	2.30896	2.30888	0.00008

The first three lines confirm that our previous attempts done by hand agree with the computer run. But now we focus on the last column, namely how many iterations it takes for another zero to appear after the decimal dot in e_k. We can see that we need 3, 4, and 3 iterations. Experiments with other tolerances and other functions and intervals would confirm that the pattern 4-3-3 is typical for the bisection method. If true, it would mean that we can expect 10 more iterations if we ask for improvement in precision by three digits.

As we see above, achieving the tolerance $\varepsilon - 10^{-4}$ required $N = 15$ iterations. When we change the tolerance to $\varepsilon - 10^{-7}$, the algorithm stops after $N = 25$ iterations. This is a very good fit with our expectations.

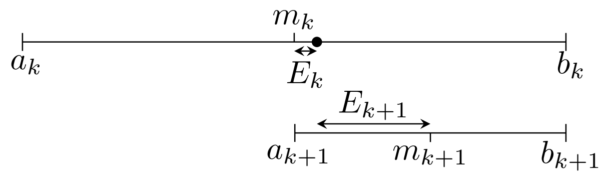
△

The last observation is interesting, What can we expect after ten iterations in theory?

$$e_{k+10} = \frac{1}{2^{10}} e_k = \frac{1}{1024} e_k \approx \frac{1}{1000} e_k.$$

Since 1024 and 1000 are close, we did not make a large error in the last step, so it indeed looks like ten iterations per three new digits is supported also by theory. Consequently, on average, three and one third of iteration are needed on average to gain a new valid digit.

Unfortunately, this conclusion would be true if we established those estimates for actual errors E_k, but we did it for the upper estimates e_k. The actual errors can behave in a different way, they may even increase from stage to stage. The following picture shows how this is possible. The actual root is marked with a dot.



Indeed, we can return to the last experiment above to observe this happening.

Example 18a.c: In example 18a.b we applied the bisection method to the function $f(x) = x^3 - x - 10$ on the interval $[0,5]$. We only listed the estimates e_k for the error, but we actually do know the actual root:

$$r = \frac{1}{3}\sqrt[3]{135+\sqrt{2022}} + \frac{1}{\sqrt[3]{135+\sqrt{2022}}}.$$

By the way, we commented at the beginning of this chapter that formulas for roots of polynomials of the third degree are not exactly friendly. Now you can see what we meant by this.

Anyway, knowing the actual root we can in fact look at the actual errors of our approximations:

k	m_k	e_k	E_k
0	2.50000	2.50000	0.19109
1	1.25000	1.25000	1.05890
2	1.87500	0.62500	0.43390
3	2.18750	0.31250	0.12140
4	2.34375	0.15625	0.03484
5	2.26562	0.07812	0.04328
6	2.30469	0.03906	0.00421
7	2.32422	0.01953	0.01531
8	2.31445	0.00977	0.00554
9	2.30957	0.00488	0.00066
10	2.30713	0.00244	0.00177
11	2.30835	0.00122	0.00055
12	2.30896	0.00061	0.00005
13	2.30865	0.00031	0.00025
14	2.30881	0.00015	0.00009
15	2.30888	0.00008	0.00002

As expected, the actual errors never exceed the upper estimates, so things are as they should. We also see repeatedly that the actual errors increased at some iterations. On the other hand, when we ignore small details and look at the overall pictures, we do see zeros appearing in the errors with roughly the same shape as in the estimates e_k, so the general trend seems to fit.
△

This is a general experience. The estimates e_k form an envelope under which the actual errors must stay, in particular they do have to get smaller at a rate that is, on average, about the same as the theoretical one. In other words, we can really expect about ten iterations for three new valid digits.

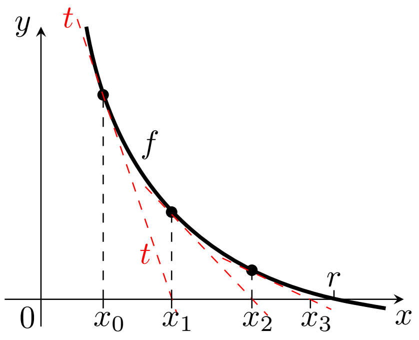
This may not sound bad, but in fact it is rather slow when it comes to problems involving functions that are not so easy to evaluate. We will see soon that much faster methods are available.

This concludes this section. We introduced a method whose strength is its simplicity, reliability and error control (a feature common to all bracketing methods), but we pay for it by low speed. One reason for this is the fact that we actually never use in our decision making any information about the actual shape of the function. We just care about signs at points that are determined by the initial interval $[a_0, b_0]$. If we want a faster algorithm, we have to look at what the function is doing.

18b. The Newton method

We have a function f and we want to approximate its root that hopefully exists.

One possibility is to simply try some x_0. We substitute it into our function f. Unless we are extremely lucky (in which case we are most likely in Las Vegas instead of reading a book on ODEs), $f(x_0)$ is not zero. What next? We would like to get a better approximation using our knowledge of the function.



If our function is differentiable, then we can find the tangent line to it at our point x_0. With a bit of luck this tangent line intersects the x-axis and the picture suggests that this intersection, call it x_1, could be a better approximation for the root. This looks like a good move, so we try it again:

We move by the tangent line at x_1 to a still better approximation x_2. And again the same move, getting x_3. This seems to work, we are getting close.

How will this work mathematically? We will deduce a general formula. We start at a point x_k, and we know $f(x_k)$ and $f'(x_k)$. First we construct the tangent line to the graph of f at x_k. It must pass through the point $[x_k, f(x_k)]$ and it has the slope $k = f'(x_k)$, so we easily write its equation:

$$t: \ y = f(x_k) + f'(x_k)\cdot(x - x_k).$$

Where does it intersect the x-axis? We put 0 for y and solve the resulting equation for x to find out:

$$-f(x_k) = f'(x_k)\cdot(x - x_k) \implies -\frac{f(x_k)}{f'(x_k)} = x - x_k$$
$$\implies x = x_k - \frac{f(x_k)}{f'(x_k)}.$$

This will be our new x_{k+1}. So we have a recursive formula for our procedure and we can write the algorithm. We also need to know when to stop, so we put a test in the algorithm, but we will not worry about it now. This is a very loaded question and we will return to it later, see section 18c.

Algorithm 18b.1.
⟨Newton method for finding root of a function f⟩
Given: a function f continuous on $\mathbb{R}$ and a tolerance ε.

0. Choose x_0. Let $k = 0$.

1. Let $x_{k+1} = x_k - \dfrac{f(x_k)}{f'(x_k)}$.
If $|x_{k+1} - x_k| < \varepsilon$ or $|f(x_{k+1})| < \varepsilon$ then the algorithm stops, output is x_{k+1}.
Otherwise increase k by one and go back to step **1**.
△

Some people call it the Newton-Raphson method, the natural name would be the tangent method.

Example 18b.a: Consider again our favorite test function $f(x) = x^3 - x - 10$. We will approximate its root using the Newton method, as the initial guess we take $x_0 = 1$.

First we find

$$f'(x) = 3x^2 - 1$$

and now we are ready.

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 1 - \frac{1^3 - 1 - 10}{3\cdot 1^2 - 1} = 6.$$

The next step:

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 6 - \frac{6^3 - 6 - 10}{3 \cdot 6^2 - 6} = 6 - \frac{100}{51} = \frac{206}{51}.$$

Obviously, this is a job for a computer, but the idea should be clear now.

It is possible to set up a dedicated iterative formula when applying the Newton method to a specific problem. In our case it would read

$$x_{k+1} = x_k - \frac{x_k^3 - x_k - 10}{3x_k^2 - 1} = \frac{2x_k^3 + 10}{3x_k^2 - 1}.$$

Sometimes such a formula simplifies, making the actual calculations easier compared to the direct application of the Newton formula. One can even develop dedicated procedures for solving specific types of problems. We will return to this idea in section 18g.

In example we asked how many iterations it took to find the desired root with certain tolerances, and we found $N = 15$ for $\varepsilon = 0.0001 = 10^{-4}$ and $N = 25$ for $\varepsilon = 10^{-7}$.

We now start the procedure again, set tolerance to $\varepsilon = 0.0001$, but this time with the Newton method and the initial guess $x_0 = 5$. We obtain the following run.

k	x_k	$f(x_k)$	test
0	5.0000000	110.0000000	110.0000000
1	3.5135135	29.8600281	1.4864865
2	2.6848585	6.6688513	0.8286550
3	2.3615265	0.8082521	0.3233320
4	2.3101450	0.0185680	0.0513815
5	2.3089080	0.0000106	0.0012370
6	2.3089073	0.0000000	0.0000007

Now that was quite fast, $N = 6$ iterations were enough. We will worry about the test column later, but the column with $f(x_k)$ looks interesting, especially the last row. In fact, the computer thinks that we actually found the root at that step.

How many iterations are needed when we increase our demand by lowering the tolerance to 10^{-7}? Remarkably, $N = 6$ is again the answer, because the algorithm thinks that x_6 is also good for this new demand. Comparison with the bisection method (6 versus 25) is inevitable, although it is not completely fair as the two methods did not start from exactly the same starting block, one needing an interval, the other just an initial guess.

However, we do seem a rather clear message. Note that x_5 was not judged good for the error 0.0001, while x_6 is considered good even for 0.0000001. Now we are yet do discuss how the computer came to this conclusion, but if it is reasonably correct, then this would suggest that we gained four more valid digits in just one iteration, something that the bisection method can only dream of.
$\triangle$

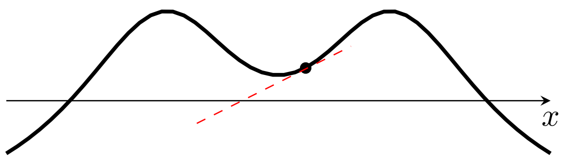
Recall that the main properties of the bisection method were as follows:
- The bisection method is reliable.
- We have a good control over the error.
- The bisection method is slow.

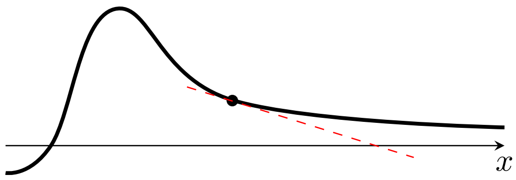
The main properties of the Newton method are as follows:
- The Newton method is unreliable.
- We have no control over the error.
- The Newton method is fast.

The third property sounds good, that is exactly what we wanted, but we paid for it dearly it seems. Each of the points deserves a thorough investigation.

The first is actually the simplest. For starters, just by looking at the formula we can see that we can get in trouble if $f'(x_k) = 0$ for some x_k. Indeed, this would mean that our x_k happened to catch our function at its local extreme, there the tangent line is horizontal and has no intersection with the x-axis. However, this is the least of our worries. For a typical function there is just a small number of local extrema, and we can easily avoid them by slight modification of our initial guess x_0, the resulting run will most likely miss the offending point.

A more serious problem lies in high sensitivity of the Newton method to the shape of the function. One can easily find shapes that totally throw this method off.





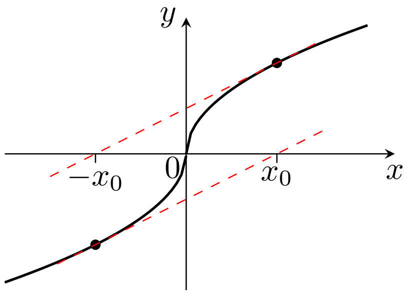
On the left we see one of my favorites, a camel. If our initial guess happens to land in the valley, then it can easily happen that tangent lines will be sending us left and right, never leaving the valley to reach the root on the side. The second shape is equally deadly. If our initial guess falls to the right of the hill, the tangent lines will chase us out to the right, all the way to infinity.

My personal favorite in this context is the function $f(x) = \begin{cases} \sqrt{x}, & x \geq 0; \\ -\sqrt{|x|}, & x < 0. \end{cases}$ Simply put, this is an odd function based on the shape of the standard square root.

Now what happens if we start the Newton method at some $x_0 > 0$? The formula says that

$$x_1 = x_0 - \frac{\sqrt{x_0}}{\frac{1}{2\sqrt{x_0}}} = -x_0.$$

So the tangent line sends us to a symmetric point. Since the graph is odd, that is, symmetric, the next step should send us back. And indeed, the calculation confirms that $x_2 = x_0$. The conclusion is obvious: Unless we start at the root itself, the iteration will keep jumping right and left.



Another problem with the Newton method is that it likes to wander. it may eventually get to the root, but first it can (and often does) take several detours and scenic rides.

Example 18b.b: We return to our favorite test function $f(x) = x^3 - x - 10$. When we used the Newton method to obtain an approximation of the root with the precision $\varepsilon = 0.0001$ and the initial guess $x_0 = 5$, it took $N = 6$ approximations and the run was straightforward.

However, when we change this initial guess to $x_0 = 0$, it will take fairly substantial $N = 22$ iterations. We will not list them all, just the extremes.

k	x_k
0	0.0000000
1	10.0000000
2	−6.6555183
6	0.5299389
7	−65.3843761
17	4.1968304

Starting with x_{17}, the iterates became a decreasing sequence converging to the root.

Incidentally, while $x_0 = 0.2$ leads to a run requiring $N = 18$ iterations, the choice $x_0 = 0.3$ can do with just $N = 11$.

△

Which brings us to another interesting feature of the Newton method: Extreme sensitivity to the initial guess.

Example 18b.c: This time we make an exception and consider the function

$$f(x) = x + \frac{1}{23}x^2 - \frac{1}{5}x^3 + \frac{31}{2*(x-5)^2+1}.$$

It has three roots:

$$r_1 = -1.938...$$
$$r_2 = -0.541...$$
$$r_3 = 5.370...$$

We will apply the Newton method with tolerance set to the traditional $\varepsilon = 0.001$ with eleven closely spaced initial guesses. The chart shows to which root the resulting runs converge, and the number of iterations required before the algorithm terminated. In the last row we show some interesting values the run visited if the run was not straightforward.

x_0	4.0	4.1	4.2	4.3	4.4	4.5	4.6	4.7	4.8	4.9	5.0
r	r_3	r_1	r_3	r_3	r_1	r_1	r_3	r_2	r_3	r_3	r_3
N	11	19	9	9	9	17	16	8	7	7	5
	19.9	−49.8		15.5		−41.7				17.0	

The table speaks for itself. The run with $x_0 = 46$ is especially intriguing. We did not list any extreme values of iterates, and yet we claim that it took 16 iterations. In fact, the iterates were oscillating between about 4 and about 5 for a while before settling on the right approximation. △

The practical lesson here is that the Newton method is an algorithm that requires supervision. In particular, for all iterative methods it pays to implement some safety stops: The algorithm stops automatically after a certain number of iterations, and the algorithm stops when numbers involved get too large. When an algorithm stops prematurely due to conditions like these, the operator looks at the sequence produced by this algorithm and makes a judgement whether the algorithm is just slow, but a longer run would produce a satisfactory approximation, or something fishy is going on.

Still, at least pictures suggest that in some situations the Newton algorithm works reliably. Indeed, the right combination of monotonicity, concavity and initial guess can provide good convergence. Here is an example of a theorem that fairly popular in textbooks.

Theorem 18b.2.
Let f be a function on an interval $[a,b]$ such that $f(a) \cdot f(b) < 0$. Assume that f is twice continuously differentiable on (a,b) and $f' \neq 0$, $f'' \neq 0$ on (a,b).
If $x_0 \in (a,b)$ is chosen so that $f(x_0) \cdot f''(x_0) > 0$, then the sequence $\{x_n\}$ generated by the Newton method converges to a root $r \in [a,b]$ of f.

Let's try to decipher it. We have a function that changes signs at endpoints, therefore there is a root in (a,b). The fact that derivatives are not allowed to be zero means that both derivatives have to settle on one of the signs for the whole interval $[a,b]$. In other words, the function has to decide what kind of monotonicity and concavity it wants to have and stick with it on that interval. Note that this is not so restrictive. Functions do change their monotonicity and concavity, but usually only at a few places, and generally consist of large stretches where these assumptions are true. When we get really close to some root, then there is a good chance that the function behaves as needed, unless the root is actually a local extreme or an inflection point. In other words, roots of higher multiplicity cause trouble again.

Put in plain words, the Newton method may behave badly globally, but once it gets close to the root, there is a good chance that things will go well.

However, the second paragraph tells us that one has to be a bit lucky, too, as the initial guess has to fit well with the shape of the function. For instance, if it is concave up, then the initial guess should be positive.

This theorem is in fact not the most practical statement, in particular it may be faster to simply try a run of Newton's method than verifying all these assumptions. The main message is that it really makes most sense to use the Newton method when we are already close to a root, which is a strategy that we will return to later.

Now we address the other two properties. Both of them are in fact rather important, raise some general questions and lead to general notions. We therefore dedicate special sections to them.

18c. Stopping conditions

Here is the situation: We have a function f, we used the Newton method to estimate its root, and after a while we arrived at an approximation x_k. Should we stop or continue? We should stop when the distance between the approximation x_k and the root r is smaller than the given tolerance $\varepsilon > 0$, but unfortunately, this is exactly the kind of information that we do not have. From knowing f and f' at x_k (or the previous points) we cannot reliably estimate the distance from the root.

This is not just a problem of the Newton method. If we use a method that is not bracketing, then we only have information about the function f at points x_k, x_{k-1}, ... , and there is no mathematical tool that would reliably derive the desired distance from the root based on this.

So how do people stop their algorithms? There are three popular **stopping conditions**. Assume that an iterative algorithm produces a sequence $\{x_k\}$. We can use one or more of the following to stop it:

- $|x_k - x_{k-1}| < \varepsilon$ (absolute difference)
- $\dfrac{|x_k - x_{k-1}|}{|x_k|} < \varepsilon$ (relative difference)
- $|f(x_k)| < \varepsilon$ (value difference)

The epsilon in the inequalities is typically the supplied tolerance ε, but it may be also another value, as we may want to modify the given precision to achieve a better run.

In some books, the first and the second condition are called the absolute error and the relative error. However, this is highly misleading, as the two conditions have nothing in common with the error of the approximation x_k. So why do we use them? Wishful thinking.

The absolute difference test is based on the idea that if the successive approximations almost do not change, then we are probably close to the root itself. Unfortunately, there is no good reason why this should be true, and it is easy to find examples of functions for which this fails spectacularly. Still, this is probably the most popular stopping condition, and we will see some indications that it can have something in common with the actual error.

For one such indication we return to the bisection method. We stopped the run based on the test $\frac{1}{2}|b_k - a_k| < \varepsilon$. However, note that this is exactly the distance between the official output m_k (the midpoint) and the endpoints of this interval. Now this interval came as a half of the previous one, so one of these endpoints must actually be the previous midpoint m_{k-1}. It follows that bisection stops when $|m_k - m_{k-1}| < \varepsilon$, that is, the absolute difference is the natural stopping condition for the bisection method.

When the root is a large number, then also the approximations become large and insisting on small absolute error can be counterproductive. After all, we are interested in relative error anyway. So when our iteration seems to be taking us to large numbers, it is a good idea to use the relative difference test. In fact, it is the natural error to consider, so perhaps we should be using it always. However, it is more difficult to calculate, and for most cases it is reasonably comparable to the absolute error. Moreover, it can behave very badly if used with very small numbers.

Now we pass to the third stopping test. It is based on the idea that if the function is really small somewhere, then the root should not be far. Again, one can easily imagine situations when this would fail, very flat functions being the culprits. However, if we can prevent our function from being flat, then we do have some control over the position of the error.

Fact 18c.1.
Let f be a function and $r \in \mathbb{R}$ its root. Assume that there is a neighborhood U of r on which f is differentiable and $|f'| \geq m_1$ on U for some $m_1 > 0$.
Then for $\hat{r} \in U$ we have the estimate $|r - \hat{r}| \leq \frac{1}{m_1}|f(\hat{r})|$.

Proof: Since both r and $\hat{r}$ are in U, then the closed interval I with endpoints $r, \hat{r}$ (in the right order) is a subset of U. Consequently, the function f is continuous on I and differentiable on its interior, which means that we can apply the Mean value theorem to it. We learn that

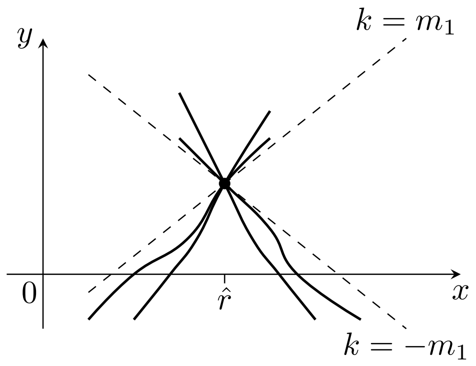
$$\frac{f(r) - f(\hat{r})}{r - \hat{r}} = f'(\xi)$$

for some $\xi \in I$. Applying our assumption we obtain

$$\frac{|f(r) - f(\hat{r})|}{|r - \hat{r}|} = |f'(\xi)| \geq m_1.$$

Now we realize that r is a root of f, so $f(r) = 0$, and rearrange the inequality to obtain the desired estimate.

□



This theorem links the function value $f(x_k)$ with the error $E_k = r - x_k$, which is very nice. The key idea is the control of the derivative. The inequality $|f'| \geq m_1$ prevents f from being too flat around r, as it forces f to grow or decrease at least as fast as the rate m_1.

What are the chances if this assumption being true? Assuming that f' is continuous, then everything depends on $f'(r)$. Indeed, if $f'(r) \neq 0$, then by continuity there must be some neighborhood U of r on which the derivative is bounded away from zero. The only bad case is therefore the one when $f'(r) = 0$, that is, when r is a root of higher multiplicity. Here we go again.

We conclude that if a root r is simple, then there is a neighborhood where we have a link between the value of the function and the distance from the root. If we want to have an approximation with error bounded by ε, we simply wait until $|f(x_k)| < \varepsilon \cdot m_1$. Unfortunately, we rarely know this m_1, so in applications this is not as useful and we are back to hoping that our stopping conditions behave reasonably.

To summarize our exploration, we have three stopping conditions to choose from, but none works reliably, and even if there are some theoretical results in the right direction, we are often unable to use them due to lack of data. Sometimes people ask for more conditions to be true before they stop, typically they combine the absolute error test with the value test, but even that is no guarantee that we did not stop too soon while our approximation is still too far from the root.

Interestingly, there is a simple test that can guarantee that our approximation is good enough. One could call it a "three value test".

Algorithm 18c.2.
⟨locating a root with given tolerance⟩
Given: A function f, tolerance $\varepsilon > 0$ and a number $\hat{r}$.

1. Evaluate $f(\hat{r}-\varepsilon)$, $f(\hat{r})$, $f(\hat{r}+\varepsilon)$. If their signs are not the same, then there is a root r of f satisfying $|r-\hat{r}|<\varepsilon$.
$\triangle$

Again, this test is not foolproof. What if the three numbers have the same sign? Then in general we do not know anything. It may be that there is no root, but it could also happen that there is a root but the function changes too quickly to show it. Then we have to investigate some more, for instance we can artificially lower the ε in our stopping condition to force the algorithm to run longer. We get a (hopefully) better approximation and test it again.

Of course, if it is a root of even multiplicity (think of the parabola), then this test will always fail, but when it comes to simple roots it usually performs rather well. In particular, if we have a reason to believe that the function we investigate is monotone around the root (for instance because we can analyze its derivative), then we can rely on this test entirely. Once all the signs are equal in a monotone function, then we can be sure that there is no root hiding in there.

Returning to the Newton method, we see that in algorithm 18b.1 we actually offer the absolute difference test and the value test for stopping the run.

18c.3 Remark on residual: When we present some solution obtained using tools of numerical mathematics, we can rarely determine its actual error. However, in many settings there is a parameter that we can evaluate. Often the thing that we are trying to find is supposed to serve some purpose. For instance, when we are trying to integrate some function f, then the result (an antiderivative F) is supposed to serve the purpose that when we differentiate it, we get f. When we are trying to find $\sqrt{A}$, we are looking for a number that is supposed to give A when squared.

In such situations we can ask by how much the proposed solution missed its mark. For instance, if somebody claims to have an antiderivative F of a given function f, then $f-F'$ tells us by how much it missed its purpose. This quantity is called a **residual**, or a **residuum**. People also talk of residual functions, residual numbers, residual vectors, depending on the setting.

The advantage is that it can be evaluated easily, unlike the error of the proposed solution. For instance, given some approximation x of $\sqrt{13}$, we would be hard pressed to find the exact error. On the other hand, we can easily calculate its residual $13-x^2$. This is a useful concept that we will revisit later. In particular, in many situations a relationship may be found between a residual and the actual error, and a natural requirement for any iterative procedure is that it makes the residual converge to zero.

How does it apply to this chapter? The purpose of a root r of a function$f$ is to make this function equal to zero. Therefore, given some root approximation $\hat{r}$, its residual would be $0-f(\hat{r})$. In fact, above we proved a theorem that relates the size of this residual $|f(\hat{r})|$ to the error. Returning to the topic of this section, the value stopping condition can be seen as a condition that works with the residual. This is another justification why it may be a good idea to use it.
$\triangle$

18d. Order of method (order of error)

Intuitively we would judge the speed of a root-finding algorithm based on how quickly it produces valid digits. This is related to the question how often does the relative error get divided by ten, and since we always divide by the same root r in the relative error, we simply want to know how frequently does the absolute error get divided by ten during a typical algorithm run.

Mathematically, we are in some way to measure how fast sequences go to zero. The natural idea is to compare two successive terms. We already worked with such a relationship when investigating the error of bisection method. We actually had an equality there, but that is a rare situation, usually we are happy with an inequality. The message we take from the bisection method investigation is therefore this inequality: $e_{k+1}\frac{1}{2}e_k$. It obviously forces the sequence e_k to go to zero.

Generally, we can talk about sequences x_k that converge to zero and about comparing the speeds at which they do so. It seems natural to consider relationships of the form $|x_{k+1}| \leq c|x_k|$ and focus on the influence of c. Obviously, the smaller the c, the faster the decrease to zero. However, it turns out that making c small is not powerful enough, and we should focus on making the “1” in the exponent bigger. Which “1”? The relationship above can be written as $|x_{k+1}| \leq c|x_k|^1$.

Generally, we can consider relationships of the form $|x_{k+1}| \leq c|x_k|^q$, and it turns out that the influence of q is incomparably stronger than the influence of c.

Example 18d.a: Here we will compare the trends of four sequences that tend to zero. They all start with $x_1 = 0.0001 = 10^{-4}$, but x_{k+1} depends on x_k in different ways.

x_{k+1}	x_1	x_2	x_3	x_4
$\frac{1}{2}x_k$	0.0001	0.00005	0.000025	0.0000125
$\frac{1}{5}x_k$	0.0001	0.00002	0.000004	0.0000008
$x_k^{1.5}$	0.0001	0.000001	0.000000001	0.000000000000316...
x_k^2	0.0001	0.00000001	0.0000000000000001	0.00000000000000000000000000000001

Recall our interpretation that zeros after the decimal dot correspond to the number of digits in our approximation that are correct. The first row is as expected for the bisection method.

The second row can be imagined as coming from some better method that improved the constant c to $\frac{1}{5}$. For instance, we may be dividing intervals into five parts instead of two. There is some improvement in how quickly new digits appear, we get roughly a new digit at each iteration in our example, although it will not last, $x_4 = 0.00000016$.

So it is better than the previous relationship, but the next row shows that this improvement is weak compared to what can be achieved by increasing the power q in the comparison. This is where the real gains are made.

In the last row, the last number is 10^{-32}. Of course we do not normally write such numbers in this way, but here it sends a clear message: The speed of convergence is spectacular. In fact, the number of zeros after the decimal dot at least doubles at each iteration. It would be really nice if errors behaved like that in our methods.

We will return to this comparison in example .

$\triangle$

The moral of the story is that if we want to force a sequence to go to zero fast, we should focus on q in the relationship $|x_{k+1}| \leq c|x_k|^q$, the value of c is of secondary importance.

Once we know how to rate sequences convergent to zero, we can rank all convergent sequences: We simply look at the behavior of the “errors”. That is, if a sequence x_k converges to some x_∞, then we look for relationship of the form $|x_\infty - x_{k+1}| \leq c|x_\infty - x_k|^q$

Definition 18d.1.
Consider a sequence $\{x_k\}$ that converges to some x_∞.
We say that it converges with order (or rate) of convergence $q > 0$ if there is C such that $|x_\infty - x_{k+1}| \leq C|x_\infty - x_k|^q$ for all k.

There are other definitions of this notion around, but they are equivalent to this one. Note that when we say “for all k”, we mean for all k used in indexing the given sequence. In fact, we do not care much, because the rate of convergence is decided at the tail of the sequence anyway, the first terms do not matter.

Note also that this notion is hierarchic. If a sequence converges for a certain rate q, then it also converges with all rates p satisfying $p < q$. Since more is better, we always try to determine the largest possible q and consider this to be the right rate of convergence for that sequence.

This notion is often called the Q-rate of convergence to distinguish it from another, similar notion. Why do we need two? recall that we plan on applying it to errors E_k of our approximations. However, in the case of bisection we were not able to establish any relationship; rather, we established an appropriate relationship for upper estimates of the errors. This can also happen in other examples, and a notion was created to take care of this situation.

Definition 18d.2.
Consider a sequence $\{x_k\}$ that converges to some x_∞.
We say that it converges with R-order (or R-rate) of convergence $q > 0$ if there is a sequence $\{e_k\}$ of upper estimates of $|x_\infty - x_k|$, that is, $|x_\infty - x_k| \leq e_k$ for all k, such that it Q-converges to zero.

Obviously, if a sequence converges with a certain Q-rate q, then it also R-converges with the same rate, because we can simply take $e_k = |x_\infty - x_k|$. However, the converse is not true, so mathematically, these two notions are not equivalent.

However, in numerical mathematics we often do not worry about this distinction, simply because we usually do not have much choice. For some methods, we can establish a relationship between errors, and we naturally use the first definition. For methods like bisection we have no choice but to use the other. We will therefore simply talk about rate of convergence.

Now we apply this to iterative methods for finding roots. These produce sequences, but we know that these need not always converge. So when we want to assign some ranking to methods, we have to take into account only convergent runs. But even then things are not so simple, because we already noted that roots of higher multiplicity cause troubles. Moreover, usually we need some assumptions on functions f to be able to deduce anything. We will therefore also disregard cases that cause trouble when ranking methods.

The definition that follows is not completely rigorous. Some authors avoid this by simply not assigning any order to methods, and talk only of sequences generated by them. However, many authors do find it useful to talk about order of method, as it sends a direct message.

Definition 18d.3.
Consider a certain iterating method for finding roots functions. We say that it is a **method of order** q, or that it has **error of order** q, where $q > 0$, if it satisfies one of the following condition:
Whenever this method produces a sequence $\{x_k\}$ converging to a certain root r of a function f, this root is simple and the function sufficiently smooth, then $\{x_k\}$ converges to r with rate q.

Note that it does not really make much sense to consider orders smaller than 1. Moreover, for methods of order 1 we actually have to focus on the constant c (note that each sequence is allowed its own c), we have to insist that there is a common upper bound $C < 1$ for all these c. In other words, all sequences $\{x_k\}$ produced by a method of order one must satisfy $|r - x_{k+1}| \leq C|r - x_k|$ with this common $C < 1$. We also say that these methods are of linear order.

Our analysis in section 18a has proved the following statement.

Theorem 18d.4.
The bisection method is of linear order (order 1).

Actually, with the bisection method there are no roots and functions to avoid, the rate of convergence is general and reliable (and small).

As we saw in the example above, we would prefer to have a rate of convergence greater than 1. How about the Newton method? Remarkably, while we cannot determine the errors, it is actually possible to establish a relationship between them. Formally, one gets a result about Q-rate of convergence.

Theorem 18d.5.
Let r be a root of a function f that is twice continuously differentiable on some neighborhood U of r and for which there are $m_1, M_2 > 0$ such that $|f'| \geq m_1$ and $|f''| \leq M_2$ on U.
Consider a sequence $\{x_k\}$ generated by the Newton method. Then for all $x_k, x_{k+1} \in U$ we have estimates

$$|r - x_{k+1}| \leq \frac{M_2}{m_1}|r - x_k|^2 \qquad \text{and} \qquad |r - x_{k+1}| \leq \frac{M_2}{2m_1}|x_{k+1} - x_k|^2.$$

The technical assumptions are satisfied in cases when the function f is twice continuously differentiable at r and the root r is simple (then $f'(r) \neq 0$). This perfectly fits with the restrictions described above and we get the following statement.

Theorem 18d.6.
The Newton method is of order 2.

For roots of higher multiplicity the Newton method generates sequences that converge linearly. This is the third time that we see roots of higher multiplicity being obtuse.

Note the second formula in the theorem. It actually offers a control over the error of approximation using the knowledge of the last two iterations, which is great. Unfortunately, it can be used only if we know those bounds m_1, M_1, which is often not the case.

Since this theorem is important, one would expect to see its proof. Since it is rather technical, we prefer to leave it to chapter so that we can focus on crucial ideas here. Another reason is that the proof suggests an interesting modification of the Newton method that just fits there.

We now now that the Newton method is as fast as the best sequence in example . This is great. On the other hand, it sometimes does not converge at all. This is a tradeoff, and there are strategies that use the good properties while largely avoiding the bad ones. One particular strategy is to first narrow down the location of the root using a more reliable method, for instance the bisection method. Once we have the error markedly smaller than 1, there is a high probability that the Newton method will go directly to the root, and it does it with a lightning speed.

One particular advantage of the Newton method is its ability to create iterating schemes. We will show some in the section 18g. On the other hand, one serious drawback is its reliance on the knowledge of derivative. When the function f is known only experimentally, we do not have this information available.

18d.7 Remark: In remark 18c.3 we introduced the notion of residual. Sometimes we can establish a relationship $r_{k+1} \leq C r_k^p$ for residuals corresponding to sequences generated by a certain method. This may be a useful information that sheds further light on how such a method performs. We will see such an example in section 18g.
△

18e. The secant method

Therefore there is a big demand for so-called derivative-free methods. Of course, the bisection method is one, but we want something faster.

One interesting idea is to start with the fast Newton method, and ask whether we could do something with the derivative in the formula. Since we are in the world of numerical mathematics here, we naturally think back to chapter 4, where we approximated derivatives numerically. In particular, we had a simple method that only needed the knowledge of f at two points. Now when we run the Newton method, we do get to know f at many points x_k. So here is the idea: At stage k, we have x_k, but also x_{k-1} from the previous stage. We can use these two to approximate the derivative at x_k:

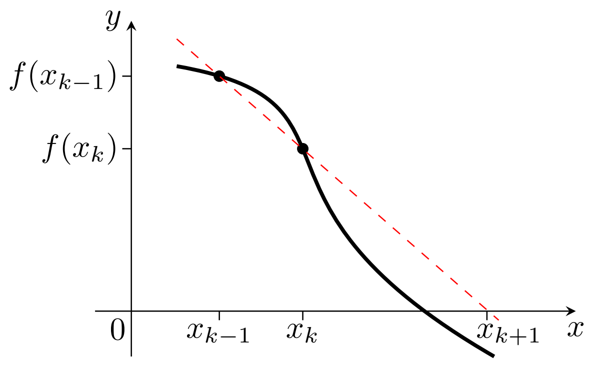
$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

and use it in the Newton formula. We get

$$\begin{aligned} x_{k+1} &= x_k - \frac{f(x_k)}{\frac{f(x_k)-f(x_{k-1})}{x_k - x_{k-1}}} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})} \\ &= \frac{x_k(f(x_k) - f(x_{k-1})) - f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})} \\ &= \frac{f(x_k)x_{k-1} - f(x_{k-1})x_k}{f(x_k) - f(x_{k-1})}. \end{aligned}$$

This looks like an interesting iterative formula. However, note one fundamental difference. Our previous iterative methods only used the immediate present to produce the future iteration. Symbolically, $x_k \mapsto x_{k+1}$. Our new method uses the present and also goes one step into the past to produce a new iterate: $x_{k-1}, x_k \mapsto x_{k+1}$. This is not anything bad, but this tells us that in order to start, this procedure actually needs two initial guesses x_0, x_1.

There is actually another thing that we can try when we know a function f at two points. We may take the corresponding points on the graph, connect them with a line and check where it intersects the x-axis. If the picture is any guide, this new point should be a better estimate for the root of this function.



The secant line has the slope

$$k = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}},$$

and thus the equation

$$s: y = f(x_k) + k(x - x_k) = f(x_k) + \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}(x - x_k).$$

Where does it intersect the x-axis?

We set y equal to zero and solve for x, obtaining

$$x = \frac{f(x_k)x_{k-1} - f(x_{k-1})x_k}{f(x_k) - f(x_{k-1})}.$$

Remarkably, this is the same formula as above. Now we have two reasons to think that this could be a good idea, let's give it a name.

Algorithm 18e.1.
⟨secant method for finding root of a function f⟩
Given: a function f continuous on $\mathbb{R}$ and a tolerance ε.

0. Choose x_0, x_1. Let $k = 1$.

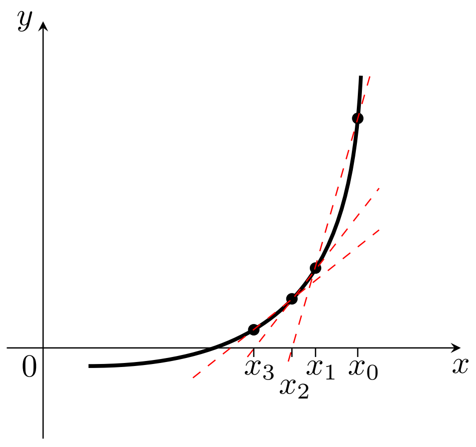
1. Let $x_{k+1} = \dfrac{x_{k-1}f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})}$.
If $|x_{k+1} - x_k| < \varepsilon$ or $|f(x_{k+1})| < \varepsilon$ then algorithm stops, output is x_{k+1}.

Otherwise increase k by one and go back to step **1**.
$\triangle$

What can we expect of this method?

- The secant method is unreliable.
- We have no control over the error.
- The secant method is quite fast.

Regarding the first observation, as an approximation of the Newton method, this new methods shares its bad traits. We may have trouble with division by zero (and we cure it by adjusting the initial guesses like with the Newton method), it can travel around before settling down, and some shapes are not good, namely the ones that we saw for the Newton method, like the camel one. When in a bad situation, the secant method may oscillate or run away. However, just like for the Newton method, also here we can identify some shapes that make the secant method work, like the one on the right. We see a nice monotone sequence being generated there, which is a nice bonus.



Theorem 18e.2.
Let f be a function continuous on an interval $[a,b]$. Assume that it is concave-up or concave-down on $[a,b]$ and $f(a)f(b)<0$. Then for arbitrary values x_0,x_1 chosen so that $f(x_0)>0$, $f(x_1)>0$ for the concave-up case, or $f(x_0)<0$, $f(x_1)<0$ for the concave-down case, the sequence $\{x_k\}$ generated by the secant method converges to a root $r\in(a,b)$ of f.

Try to draw a picture capturing the meaning of this theorem, it is a good exercise.

Since the secant method is not a bracketing method, we again have no control over the error, so we are reduced to guessing when to stop this algorithm, and we arrive at the same stopping conditions as described above. Generally speaking, also this method requires supervision.

Regarding the speed, by approximating the derivative we actually lost a bit, but not that much.

Theorem 18e.3.
Let f be a function twice continuously differentiable on some neighborhood U of its root r. Let $\{x_k\}$ be a sequence generated by the secant method such that $x_k\to r$. If the root r is simple, then there is a constant $K>0$ and $N\in\mathbb{N}$ such that $|r-x_{k+1}|\le K|r-x_k|^{\varphi}$ for $k\ge N$, where $\varphi=\frac{1+\sqrt{5}}{2}$.
If the root is of higher multiplicity, then the convergence is linear.

Note that $\alpha\approx 1.618$, so this convergence rate is definitely better than linear, but not as good as quadratic. Note that as usual, roots of higher multiplicity cause troubles.

Corollary 18e.4.
The secant method is of order $\varphi=\frac{1+\sqrt{5}}{2}$.

Now what kind of a number is this φ? Actually, it is the famous golden ratio, famous for its role in arts, appearing in nature at unexpected places and perhaps, according to some, even capturing the substance of the universe. People of esoteric bend should definitely use the secant method to find their roots.

What can we expect of it in practice?

Example 18e.a: In example we explored sequences that all started with the same e_1 and grew at different rates. One of them had rate of convergence 1.5, which is close to the rate of the secant method, so it gives us some idea of what to expect.

Here we will try it again, this time with the right order. To this end we will look at the sequence that starts with $e_1 = 0.01$ and given by the recursive formula $e_{k+1} = e_k^{\varphi}$. However, this time we will not list the values e_k, but the number of zeros after the decimal dot for them. In this way we will get a fairly good idea of how fast the secant method improves its approximations and generates correct digits.

k:	2	3	4	5	6	7	8	9
zeros	3	5	8	13	22	35	58	93

Especially the first four numbers should look familiar. It is the Fibonacci sequence. The next number is different, but you can observe that most of the time, numbers of zeros are the sum of the previous two, so the number of zeros does behave almost like the Fibonacci sequence. Actually, this sequence is closely related to the golden ratio, so once there is one, the other is not far. Still, it is a nice observation.
△

Just like the Newton method, the secant method combines advantages with disadvantages. However, in the following section we will see that it may be even better than it seems.

18f. Practical considerations

What did we learn? We have the bisection method: the lumbering giant that takes on average three and a third iterations to gain one correct digit, but is bulletproof reliable and yields a guaranteed error estimate.

On the other hand we have the lightning fast Newton method that doubles the number of correct digits with each iteration, but that only if it happens to converge and we happen to get close to the root to start with. Now take your choice.

What people usually do is to combine them. First they fish around for the root, trying numbers until they hit on opposite signs, then they narrow the root down to a small interval using bisection, it is enough to get an interval shorter than 1, which can be done reasonably quickly. And then they switch to the Newton method to quickly obtain very good approximation. Which they cannot confirm, but readers of this book know our test 18c.2 can do the trick. Note that this approach works only if we have the derivative available to us. If we don't, then we can use the secant method as the finisher.

There are several topics that were not addressed yet. We leave some of them to the next chapter, but the notion of order deserves a closer look before we leave this chapter.

There are several problems with the definition of order. We start with the following idea. Imagine a new, lightning-fast method, there each step looks like this:

1. Given x_k, we look at the tangent line to the graph of f at x_k and call its intersection with the x-axis y_t for "temporary".
2. We construct a tangent line to the graph of f at y_t and define x_{k_1} as its intersection with the x-axis.

You can see the trick: We take two steps of the Newton method and pretend that it is just one step. How fast is this new method? Consider some simple root r and a run that converges to it. Let E_k be the error of x_k. Since y_t was actually constructed using the Newton method, the error of y_t should satisfy $|E_t| \leq c|E_k|^2$. Similarly, the error of x_{k+1} should satisfy $|E_{k+1}| \leq c|E_t|^2$. Putting these two inequalities together we obtain $|E_{k+1}| \leq c^3|E_k|^4$. Voila, we have a method of order 4.

Now mathematicians would not do anything like this, they are interested in knowledge, not in upmanship, but this shows that the mathematical notion of order is not as solid as it could be. Is it possible to actually find some reliable measure of speed?

There is another concern. Some methods do not proceed uniformly, but mix "faster" and "slower" steps. How do we judge their speed then? It is possible to develop a notion of an "average rate of convergence".

Consider some sequence $\{x_k\}$ that converges to zero with known order of convergence q. We will check how it behaves after several steps:

$$\begin{aligned}
|x_{k+1}| &\le c|x_k|^q,\\
|x_{k+2}| &\le c|x_{k+1}|^q \le c(c|x_k|^q)^q = c^{q+1}|x_k|^{q^2},\\
|x_{k+3}| &\le c|x_{k+2}|^q \le c(c^{q+1}|x_k|^{q^2})^q = C|x_k|^{q^3},\dots
\end{aligned}$$

In general, $|x_{k+n}| \le c|x_k|^{q^n}$. This can be used to determine average rates of convergence over several iterations. If we can find a constant p so that $|x_{k+n}| \le c|x_k|^p$, then the average order is $q = \sqrt[n]{p}$.

Now it is time to put things together. When we run an algorithm, its speed does not depend on number of iterations, because "iterations" are just ways of writing ideas. The speed depends on the number of calculations and other things that the computer must do (comparisons, checking on tests and such). Typically, the most work is spent on evaluating the function f and perhaps its derivatives. In some cases this can be extreme, for instance when the value has to be determined by an experiment that takes considerable time.

It therefore makes sense to measure performance of algorithms based on the number of evaluations that are needed. This does not have an official name, but it is of extreme importance to people who actually calculate things. We can call it the practical order. It is very useful and we will make good use of it in the next chapter. How do our three basic methods fare?

The bisection method: In every iteration we need just one function evaluation, namely $f(m_k)$. Then we are comparing this with signs at the endpoints of $[a_k, b_k]$, but this information as already calculated in previous iterations and we keep it stored somewhere. We conclude that each "mathematical" iteration requires one evaluation, and therefore the mathematical order coincides with the practical one.

The secant method: In every iteration we need just one function evaluation, namely $f(x_k)$. For x_{k+1} we also need $f(x_{k-1})$, but we keep it from the previous iteration for sure. Again, the mathematical order coincides with the practical one.

The Newton method: For every iteration we need to find $f(x_k)$ and $f'(x_k)$, that is, two function evaluations. This means that while the improvement $|E_{k+1}| \le c|E_k|^2$ happens in one mathematical iteration, from practical point of view this actually represents two steps, and the practical order is therefore lower. If we denote this practical order q, we have $|E_{k+1}| \le c|E_k|^{q^2}$ by our above observation. Therefore $q^2 = 2$ and the practical order of the Newton method is $p = \sqrt{2} = 1.41...$, which is even lower than the secant method.

This is interesting, and it can be compared in another way. Consider a certain approximation x_k of a root with error E_k. If we apply the Newton method, this error improves to $c|E_k|^2$ (approximately). However, it will take two function evaluations, which means that the secant method can make two iterations during the same time. It can therefore improve the error to $C|E_k|^{\varphi^2}$, and $\varphi^2 \approx 2.6$. This is significantly better, even if we take into account that the constant C can be larger than the constant c for the Newton method.

We thus see an interesting picture. While on paper the Newton method is faster, in practice the secant method runs faster, and it also does not need any derivative as a bonus. Thus it actually makes sense to use the secant method rather than the Newton method.

So why is the Newton method so popular? First, when the function we investigate is simple to evaluate, then the service of iterations starts being important, and the Newton method does run faster. This is especially true if we do not substitute to the Newton formula as given, but first create a dedicated iterating scheme for the given function. Determining x_{k+1} then means just a substitution of x_k into an expression that can be significantly simpler than the ratio $\frac{f(x_k)}{f'(x_k)}$.

In fact, the ability of the Newton method to prepare iteration schemes of high practical utility, as we will see in the next section, is the second reason why it is very useful.

Finally, the Newton method is very versatile. It can be readily generalized to more dimensional settings, in fact once we are in a situation when some kind of differentiation makes sense, then the Newton method has a good chance of working, which can include even infinitely-dimensional situations.

Now the reader probably wonders: Where is the obligatory section on numerical stability? The beauty of iterative methods is that we do not have to worry much about them. The only significant source of errors is the evaluation of f itself. Usually we worry about propagation of such errors in our calculations. However, here all methods are supposed to improve the approximations that they produce. This means that even if some approximation $x-k$ is determined with a small numerical error, this will be remedied when x_{k+1} is created. In a way, iterative methods are self-correcting.

Thus we can expect that numerical errors in x_k should not differ much from the base roundoff error of the system. This is typically negligible compared to the desired precision ε, because in general it does not make sense to ask for a precision that is close to the base precision of the system.

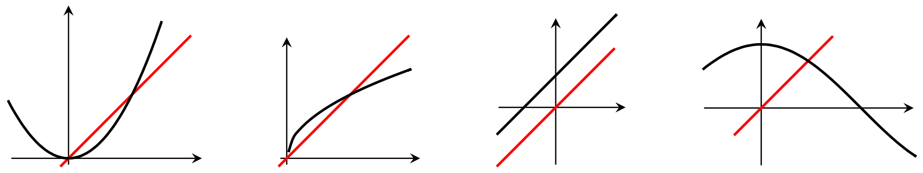
19. Finding fixed points numerically

We return to the problem of solving algebraic equations numerically, this time from another angle. Every algebraic equation can be written in the form $\varphi(x) = x$. For instance, the quadratic equation $x^2 - 3x + 2 = 0$ can be written as $x = \frac{1}{3}(x^2 + 2)$, now $\varphi(x) = \frac{1}{3}(x^2 + 2)$. We had a name for numbers r satisfying $f(r) = 0$. There is also a name for numbers solving the new kind of equation.

Definition 19.1.
Let φ be a function.
By a **fixed point** of φ we mean any number x_f satisfying $\varphi(x_f) = x_f$.

In the previous chapter we used the equation $\cos(x) = x$ as an inspirational example of a simple equation that cannot be solved analytically. We had to rewrite it as $\cos(x) - x = 0$ so that we can use tools that we learned there. However, now we can treat it as it came, as an equation asking for a fixed point of the function $\varphi(x) = \cos(x)$.

Most people associate functions with their graphs. In this setting, the fixed-point equation $\varphi(x) = x$ can be interpreted as a problem of finding the intersection of two graphs, the graph of $y = \varphi(x)$ and the graph of $y = x$.

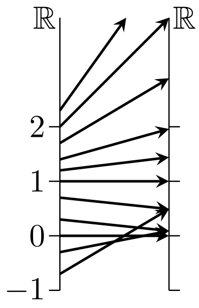


Just looking at the pictures we make an educated guess that the functions $\varphi(x) = x^2$ and $\varphi(x) = \sqrt{x}$ have fixed points $x_f = 0$ and $x_f = 1$, while the function $\varphi(x) = x + 0.5$ has no fixed point at all. Finally, we see that $\varphi(x) = \cos(x)$ has a fixed point.

This point of view has its uses and we will return to it. However, the substance of the notion of a fixed point is better appreciated when we see functions as mappings that send numbers from one set to another.

Example 19.a: Here is an example for $\varphi(x) = x^2$:

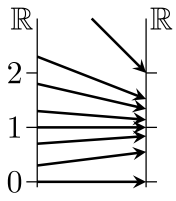
The picture suggests that regions above $x = 1$ get shifted up and stretched by this mapping, while regions between -1 and 1 get somehow folded to appear between 0 and 1 and pulled towards 0.



It is quite obvious that there are only two numbers that are preserved under this transformation, namely $x_f = 0$ and $x_f = 1$. That is the substance of being a fixed point.

Similarly, when we view x^2 as a number-sending process, then we see exactly these two numbers that get sent to themselves.

Here we see a similar picture for the function $\varphi(x) = \sqrt{x}$.



We know that this function makes numbers larger than 1 smaller, while numbers between 0 and 1 get larger.

We see the same fixed points as in the previous example, $x_f = 0$ and $x_f = 1$.

△

This point of view allows us to appreciate one of the great advantages of the notion of a fixed point: It is a very general idea that can be applied to virtually any mapping that maps a set into itself.

Example 19.b: Consider the set M of all invertible (regular, nonsingular) square matrices of all dimensions. This is just a set, it does not have any algebraic structure, because square matrices of different dimensions cannot be added or multiplied.

We consider the mapping φ: $M \mapsto M$ defined as $\varphi(A) = A^{-1}$. Does it have any fixed points?

We are looking for matrices such that $A^{-1} = A$. There is definitely one, namely the identity matrix E_n for any n. However, there are more. In linear algebra these are called involutory matrices and quite a lot is known about them, in particular they include orthogonal matrices, which is a pretty important group.

Simple involutary matrices can be obtained from identity matrices by changing signs of entries or by permuting rows.

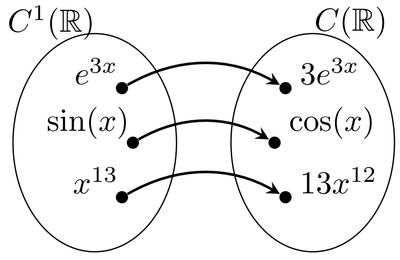
△

Example 19.c:

Consider the mapping D that sends functions to their derivatives. Formally, $D[f] = f'$. For instance, $D[\sin(x)] = \cos(x)$.

On the other hand, we cannot apply this mapping to the function $|x|$, unless we decide to work on, say $(13, \infty)$. Obviously we need a more precise specification of what we mean.

To make ourselves clear, we choose as our starting set (domain) the set $C^1(\mathbb{R})$ of all real functions on $\mathbb{R}$ that are differentiable and their derivatives are continuous. Then this mapping D can accept elements of this set as arguments. The outputs no longer need to have a derivative, but they are continuous, so the target set (co-domain) could be the set $C(\mathbb{R})$ of all continuous real functions on $\mathbb{R}$. We therefore have this setup:



$$D:\ C^1(\mathbb{R}) \mapsto C(\mathbb{R}); \qquad D:\ f \mapsto f'.$$

Now definitely the function $|x|$ does not belong to the domain of D.

Every student who passed introductory calculus knows that this mapping D has two fixed points (which means functions now), namely e^x and 0 (a constant function). There are no other functions like this.

As we will see shortly, we prefer the mapping to have its domain and co-domain identical when talking about fixed points, or at least the co-domain should be a subset of the domain. This is not the case here, and it is caused by the fact that by differentiating a function we in a way deprive it of its properties - it loses one order of derivative, it may even loose continuity. Practically speaking, if we try to be more restrictive with the co-domain so that it becomes a part of the domain, we are in turn forced to be more restrictive in the domain. For instance, if we wanted to make sure that the outputs are differentiable, we would have to start with twice-differentiable functions, so the co-domain would not be a part of the domain again.

There is a way to get out of this vicious circle, we can consider the set $C^\infty(\mathbb{R})$ of functions that have derivatives of all orders on $\mathbb{R}$. Derivatives of such functions still keep this property, so we get the following nice picture:

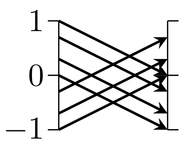
$$D:\ C^\infty(\mathbb{R}) \mapsto C^\infty(\mathbb{R}); \qquad D:\ f \mapsto f'.$$

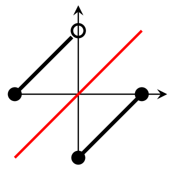
The space $C^\infty(\mathbb{R})$ is quite rich, for instance it includes polynomials, the exponential function, and sine and cosine. On the other hand, the function $x^{4/3}$ does not belong there. It is defined on $\mathbb{R}$ and continuous derivative $\frac{4}{3}x^{1/3}$ there, but its second derivative (namely $\frac{4}{9}\,\frac{1}{x^{2/3}}$) exists only when $x \neq 0$.

△

Can we identify situations when a function must have a fixed point? One way to prevent a function from moving things around too much is to require that it maps some interval into itself. But this is not enough. For instance, the function x^2 maps $(0, 1)$ onto $(0, 1)$, but does not have a fixed point there. This inspires us to focus just on closed intervals. However, that need not be enough.

Consider the function $\varphi(x)$ defined as $\varphi(x) = x - 1$ for $x \geq 0$, while $\varphi(x) = x + 1$ for $x < 0$. Then φ maps the interval $[-1, 1]$ into itself, but it has no fixed point there.

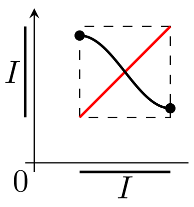




However, the action of this function is disjointed at places. When we restrict our attention only to reasonably smooth acting functions, then it is not possible to send an interval into itself without leaving some point in its place.

Theorem 19.2.
Let φ be a function on some closed bounded interval I.
If $\varphi[I] \subseteq I$ and φ is continuous on I, then φ has a fixed point $x_f \in I$.

Geometrically, this statements seems obvious. The condition $\varphi[I] \subseteq I$ means that the graph must be in the square $I \times I$. Since φ is defined on the whole closed interval I, then its graph must start somewhere on the left edge of that square and end somewhere on the right edge. And since the graph is uninterrupted by continuity, it must cross the diagonal somewhere, even if it were at one of the endpoints.



Proof: As a closed interval, I can be written as $[a, b]$ for some $a < b$. Consider the auxiliary function $h(x) = \varphi(x) - x$. Since both $\varphi(x)$ and x are continuous functions on I, so is the function h. Because φ maps I into I, we must have $\varphi(a) \in I$ and $\varphi(b) \in I$, in particular $\varphi(a) \geq a$ and $\varphi(b) \leq b$. Thus

$$h(a) = \varphi(a) - a \geq a - a = 0,$$
$$h(b) = \varphi(b) - b \leq b - b \leq 0.$$

This means that $h(b) \leq 0 \leq h(a)$ and h is continuous, hence by the Intermediate value theorem the value 0 must be attained at some point c in I. But $h(c) = 0$ means $\varphi(c) = c$ as needed.

□

Example 19.d: One of the motivational examples in the previous chapter was the simple yet difficult equation $\cos(x) = x$. This time we do not have to rewrite it into another form and treat it as it is: a fixed-point type of problem.

The function cos definitely maps the interval $[-1, 1]$ into itself, so it must have some fixed point there by the above fact.

△

19a. (Plain) iteration

So much for the theory, but how do we find this fixed point? The default approach is to use an iterative procedure, that is, we will produce a potentially infinite sequence of approximations x_k that will hopefully converge to some fixed point x_f. The way in which these approximations are produced may seem weird at the first sight, but it can be remarkably efficient. Note that the coming algorithm has the same problems with error control as the Newton method, the second method and many others, so we have to resort to the usual stopping conditions, with their advantages and disadvantages.

Algorithm 19a.1.
⟨iteration for fixed points⟩
Given: A function φ and tolerance $\varepsilon > 0$.

```
0. Choose a number $x_0$. Let $k = 1$.
1. Let $x_{k+1} = \varphi(x_k)$.
If $|x_{k+1} - x_k| < \varepsilon$ or $|f(x_{k+1})| < \varepsilon$ then algorithm stops, output is $x_{k+1}$.
Otherwise increase $k$ by one and go back to step 1.
```

People also call this a plain iteration or a simple iteration, because there will be more sophisticated versions coming up. We offered the usual suspects for stopping the algorithm, and as usual, for larger x_k people sometimes use the relative stopping condition. How does the algorithm work?

Example 19a.a: Consider the problem $\cos(x) = x$. To find the fixed point we decide to use the plain iteration, and we start at, say, $x_0 = 0$.
Then $x_1 = \cos(x_0) = \cos(0) = 1$. So far so good.
Next, $x_2 = \cos(x_1) = \cos(1)$. Fine.
Next, $x_3 = \cos(x_2) = \cos(\cos(1))$.
Next, $x_4 = \cos(x_3) = \cos(\cos(\cos(1)))$.
Perhaps we could ask a calculator for help. In fact it is simple, we just put in 1, make sure that it is switched to radians and then just keep pressing the cos button and read out the answers (unless you have a modern calculator where you have to type in an expression, good for you):

$$0.,\ 1.,\ 0.540...,\ 0.857...,\ 0.654...,\ 0.793...,\ 0.701...,\ 0.764...$$

and after a while

$$0.7392...,\ 0.7390...,\ 0.7391...$$

Remarkably, this seems to converge to the solution we found in the previous chapter.
△

Example 19a.b: How about the two powers?
We start with $\varphi(x) = \sqrt{x}$ and the initial guess 0.5. Pressing the $\sqrt{\ }$ button repeatedly yields the following numbers:

$$0.5,\ 0.71...,\ 0.84...,\ 0.92...,\ 0.96...,\ 0.978...,\ 0.989...,\ 0.994...$$

and even a pessimist would start thinking that this looks like a sequence that converges to 1, which is a fixed point as we observed above.
Try the same procedure with $x_0 = 2$ to convince yourself that the resulting sequence again converges to 1.
Now we look at $\varphi(x) = x^2$. We try again $x_0 = 0.5$. Hitting the x^2 button repeatedly produced numbers

$$0.5,\ 0.25,\ 0.0625,\ 0.0039...,\ 0.000015...$$

that seems to go to zero.
After all, if $x_0 = \frac{1}{2}$, then $x_1 = x_0^2 = \left(\frac{1}{2}\right)^2 = \frac{1}{4}$, $x_2 = x_1^2 = \left(\frac{1}{4}\right)^2 = \frac{1}{16}$, $x_3 = x_2^2 = \left(\frac{1}{16}\right)^2 = \frac{1}{256}$, this really looks like this sequence should converge to zero, and quite fast.
On the other hand, if we start with $x_0 = 2$, then the resulting sequence goes $x_1 = 2^2 = 4$, $x_2 = 4^2 = 16$, $x_3 = 16^2 = 256$, $x_4 = 256^2 = 65536$, this looks like a divergent sequence.
So it seems that the plain iteration can work, but not always (actually, when I saw this for the first time I found the former more surprising).
△

Example 19a.c: We commented that the notion of fixed point can be very general. Recall the differential operator D that we now consider to map C^∞ to C^∞. We will apply the plain iteration to this mapping, with $x_0 = x^3 + 13x$.

Then $x_1 = D[x_0] = [x^3 + 13x]' = 3x^2 + 13$, $x_2 = D[x_1] = [3x^2 + 13]' = 6x$, $x_3 = D[x_2] = [6x]' = 6$, $x_4 = [6]' = 0$, $x_5 = [0]' = 0$, ...

Even without knowing how convergence is done for functions, this looks like a sequence of functions that converges to the fixed point of D, the zero constant function.

What if we start with $x_0 = e^{x/3}$? Then $x_1 = [e^{x/3}]' = 3e^{x/3}$, $x_2 = [3e^{x/3}]' = 9e^{x/3}$, $x_3 = [9e^{x/3}]' = 27e^{x/3}$, in general $x_k = \frac{1}{3^k}e^{x/3}$.

There are several possible ways to define convergence of functions, but the most popular ones (pointwise convergence, uniform convergence on bounded intervals) would agree that the sequence of functions $\big\{\frac{1}{3^k}e^{x/3}\big\}$ actually converges to the zero constant function.

So things seem to work quite well even in this abstract setting.

On the other hand, taking $x_0 = \sin(x)$ does not get us far, obviously:

$$x_1 = \cos(x),\ x_2 = -\sin(x),\ x_3 = -\cos(x),\ x_4 = \sin(x), \dots$$

We briefly return to the example with matrices. Starting with a matrix $x_0 = A$, the plain iteration produces the sequence $x_1 = A^{-1}$, $x_2 = (A^{-1})^{-1} = A$, $x_3 = A^{-1}$, ... that does not get us anywhere. So in this example the plain iteration does not help much.

△

The experiments suggest that the (plain) iteration can work, but only sometimes. We need to investigate this closer, and start by confirming that if such an iteration converges, then it already must yield the right object.

Theorem 19a.2.
Let φ be a function. For some $x_0 \in \mathbb{R}$, consider the sequence defined recursively by $x_{k+1} = \varphi(x_k)$ for $k \in \mathbb{N}_0$.
If $x_k \to x_f$ and φ is continuous at x_f, then $\varphi(x_f) = x_f$.

Proof: The recursive equality can be seen as an equality between two sequences, $\{x_{k+1}\}$ and $\{\varphi(x_k)\}$. If their terms are equal, then the limit must also be the same:

$$\lim_{k\to\infty}(x_{k+1}) = \lim_{k\to\infty}\big(\varphi(x_k)\big).$$

If $\{x_k\}$ converges to x_f, then also the shifted sequence $\{x_{k+1}\}$ must converge there, which settles the left-hand side. On the right we recall one theorem from calculus: If $\varphi(x)$ is continuous at the limit point x_f of $\{x_k\}$, then we can move the limit operation inside the function. We obtain

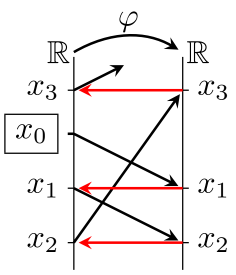
$$x_f = \varphi\big(\lim_{k\to\infty}(x_k)\big) \implies x_f = \varphi(x_f).$$

We confirmed that the limit point is also a fixed point of φ.

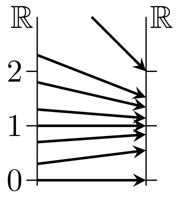
□

In words, we proved that if this procedure outputs some number, then this number is what we want. In the theory of algorithms, this type of result is called partial correctness. It leaves open the question of actually making this process produce some result. We would like to identify situations when the iteration converges. We start by interpreting our iteration as seen in the arrow-type picture.

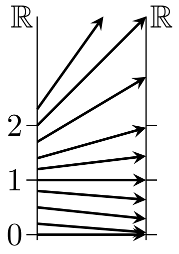
We start with some x_0 in the domain and it get sent to x_1 in the co-domain on the right. However, to obtain x_2 we need to substitute x_1 into φ, so we have to move x_1 identically back into the domain. Only then we can send it again, this time to x_2, and the whole process repeats itself.
Now that we understand how iteration works, we will revisit the examples that we started with.



Here we see the action of $\varphi(x) = \sqrt{x}$. Imagine that we start with some x_0 between 0 and 1. We move to the right and the square root shifts us up, then we jump back left on the same level, then we go right and a bit up again, jump left, and as we keep doing this, we imagine that we follow something like a spiral winding up towards 1. Similarly, if we start such a spiral from $x_0 > 1$, then we will spiral down towards 1. We would guess that if we start our iteration with any positive x_0, then this iteration takes us to the fixed point 1. If we start with $x_0 = 0$, we obtain the constant sequence $x_k = 0$ that clearly converges to the fixed point 0.



The picture for $\varphi(x) = x^2$ suggests that if we start iterating with x_0 between 0 and 1, then the resulting sequence should go to 0. On the other hand, if we start with $x_0 > 1$, then the iteration starts spiralling up beyond control.
It would seem that the only way to reach the fixed point 1 is the constant iteration $x_k = 1$.
We are ready to formulate some intuitive conclusions. If we want the iteration to take us successfully to a certain fixed point x_f, then we need for the function arrows to somehow point toward x_f.



We would like to express this idea mathematically. The fact that arrows somehow gravitate towards each other means that the distance between the outcomes should be smaller than the distance between starting points. Mathematically,

$$|\varphi(x) - \varphi(y)| < |x - y|.$$

This is not a bad idea, but one has to be careful. We surely saw in analysis that a limit process can turn sharp inequalities into equality. In this particular case, if we just asked for this property, it could happen that depending where we look, the quantities on the left and right can get arbitrarily close to each other, which would cause troubles. We therefore need to be more strict and insist that the decrease in mutual distances when being sent by a mapping has some minimal size.

Definition 19a.3.
Let φ be a function on an interval I. We say that it is **contractive** there, or that it is a **contraction**, if there exists $q < 1$ such that for all $x, y \in I$ we have

$$|\varphi(x) - \varphi(y)| \leq q \cdot |x - y|.$$

The constant q serves as a separation. We can see this better if we rewrite this condition in the following way:

$$\frac{|\varphi(x) - \varphi(y)|}{|x - y|} \leq q.$$

If we just asked for $\frac{|\varphi(x)-\varphi(y)|}{|x-y|} < 1$, then it could happen that the ratio approaches 1 arbitrarily close, which would not do. We use $q < 1$ to keep this ratio away from 1 by at least some positive distance.
It turns out that this condition is just what was needed.

Theorem 19a.4. (Banach's fixed-point theorem)
Let φ be a contractive function with coefficient q on a closed bounded interval I such that $\varphi[I] \subseteq I$.
Then there exists exactly one solution x_f of the equation $\varphi(x) = x$ in I. Moreover, for all choices of $x_0 \in I$ the sequence given by $x_{k+1} = \varphi(x_k)$ converges to x_f and satisfies

$$|x_f - x_{k+1}| \leq q|x_f - x_k| \quad \text{and} \quad |x_f - x_{k+1}| \leq \frac{q}{1-q}|x_{k+1} - x_k|.$$

This theorem supplies us with everything that we may ask for. We get convergence for any starting point x_0, and also an interesting information. The first inequality actually compares errors of approximations x_k. It reads $E_{k+1} \leq c|E_k|$, so the method of fixed point iteration is of order 1. This does not look too good, but as we will see below, this can be significantly improved.

The second inequality shows that $|E_{k+1}| \leq c|x_{k+1} - x_k|$, so we have a connection between the absolute difference (a popular stopping condition) and the error of approximation. As usual, we do not always have the information needed for a practical application of these formulas, but it is a nice thing to have anyway.

The Banach fixed point theorem is actually a very strong result, because it works not just for functions, but in general for mappings between metric spaces. We commented before that the notion of fixed point is very general, and this theorem shows that this idea can be taken to its full fruition.

Proof: We already proved that for any continuous function φ: $I \mapsto I$ there must be some fixed point x_f in I.

Could there be more? Assume that we get fixed points x_f and y_f in I. Since they are fixed points, we can replace x_f with $\varphi(x_f)$ and similarly with y_f, then we use the fact that φ is a contraction. We obtain

$$|x_f - y_f| = |\varphi(x_f) - \varphi(y_f)| \leq q|x_f - y_f|.$$

Rewriting this inequality we get

$$0 \leq q|x_f - y_f| - |x_f - y_f| = (q-1)|x_f - y_f|.$$

The contraction assumption means that $q < 1$, so the number $q - 1$ is negative. We divide the inequality:

$$0 \geq |x_f - y_f|.$$

HOwever, as an absolute value, the expression on the right can never be negative, so $|x_f - y_f| = 0$, that is, $x_f = y_f$. There can be only one fixed point.

Now take some $x_0 \in I$ and consider the sequence x_k generated by the iteration. Using the replacement $x_f = \varphi(x_f)$ and the definition of contraction we get

$$|x_f - x_{k+1}| = |\varphi(x_f) - \varphi(x_k)| \leq q|x_f - x_k|, \qquad (*)$$

which proves the first inequality in the statement.

It is also a recursive formula that can be applied repeatedly.

$$|x_f - x_k| \leq q|x_f - x_{k-1}| \leq q \cdot q|x_f - x_{k-2}| \leq \cdots$$
$$\cdots \leq q^k|x_f - x_0|.$$

Since $|q| < 1$, the geometric sequence q^k tends to 0, therefore $|x_f - x_k| \to 0$, that is, $x_k \to x_f$.

Finally, we return to the inequality $(*)$:

$$|x_f - x_{k+1}| =\leq q|x_f - x_k| = q|x_f - x_{k+1} + x_{k+1} - x_k|$$
$$\leq q|x_f - x_{k+1}| + q|x_{k+1} - x_k|.$$

We subtract $q|x_f - x_{k+1}|$ from both sides, rewrite the inequality and obtain

$$(1-q)|x_f - x_{k+1}| \le q|x_{k+1} - x_k|$$
$$\implies |x_f - x_{k+1}| \le \frac{q}{1-q}|x_{k+1} - x_k|$$

as claimed.

□

It would be nice if we could recognize contractions easily. There is such a tool.

Theorem 19a.5.
Assume that function φ defined on an interval I has a continuous derivative on the interior I^O of I.
If there is $q < 1$ such that $|\varphi'(t)| \le q$ on I^O, then φ is a contraction on I with coefficient q.

Proof: Take any $x, y \in I$, for simplicity we may assume that $x < y$. Then $[x,y] \subseteq I$ and $(x,y) \subseteq I^O$, so φ is continuous on the former interval and differentiable on the latter. In other words, we can apply the Mean Value theorem on $[x,y]$ to learn that

$$\frac{\varphi(x) - \varphi(y)}{x - y} = \varphi'(\xi).$$

for some $\xi \in (x,y)$. Applying our assumption we obtain

$$\frac{|\varphi(x) - \varphi(y)|}{|x - y|} = |\varphi'(\xi)| \le q$$

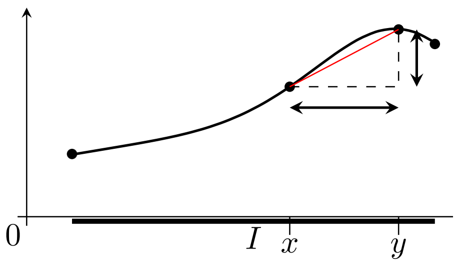
and the proof is complete.

□

This brings us back to the second point of view that we tried. How do contraction look in the usual graph point of view of real functions? We need to control the ratios

$$\frac{|\varphi(x) - \varphi(y)|}{|x - y|}$$

that we know well:



A function φ is a contraction on an interval I if all the secant lines as in the picture have slopes smaller (by some definite separation) than 1. In short, the function should be rather flat. As we know well, this can be controlled by the derivative, so the above result makes sense.
Returning to our examples, the graph of $\sqrt{x}$ is flat around 1, so we are not surprised to see contraction working there. On the other hand, the graph of x^2 grows fast around 1, that explains a lot.
However, note that the Banach contraction theorem is just an implication. The success or failure of iteration owns a lot to geometric configuration of the graph. To appreciate this we now look closer how the iteration actually appears in a graph of a function.

Example 19a.d: Consider the equation $\cos(x) = x$. The natural interval I is $[-1,1]$, then $\varphi(x) = \cos(x)$ indeed maps I into I. Is it a contraction there?

We check on the derivative: $|\varphi(x)| = |\sin(x)|$. Since the sine function is increasing on $[-1,1]$, we get an upper bound $|\varphi(x)| \leq \sin(1) = q$. Since this q is smaller than 1, the mapping $\cos(x)$ is a contraction on $[-1,1]$. Therefore there is some fixed point x_f in $[-1,1]$ and for every $x_0 \in [-1,1]$, the plain iteration produces a sequence that converges to x_f.
Note that if we just choose any $x_0 \in \mathbb{R}$, then $x_1 = \cos(x_0) \in [-1,1]$, and we can think of the process that follows as if it started there. Thus the resulting sequence also necessarily converges to x_f. We conclude that the plain iteration also works if we consider $\cos(x)$: $\mathbb{R} \mapsto \mathbb{R}$, although it is not a contraction there. Indeed, if we take x, y closer and closer to $\frac{\pi}{2}$, then the ratio $\frac{|\varphi(x)-\varphi(y)|}{|x-y|}$ approaches 1 arbitrarily close.
$\triangle$

Similarly, we know that the derivative $\left[\sqrt{x}\right]'$ approaches infinity as $x \to 0^+$, so the function $\sqrt{x}$ is definitely not a contraction on intervals $[a,b]$ for $b > 1$ and a close to zero, but as we observed, the plain iteration produces sequences that tend to the fixed point 1 on such sets.
We see that the contraction test is not perfect, but it is the best tool that we have if we want to guess whether iteration has a good chance of converging. We check on the derivative, and if it is small, we know that things look good. But we also know that even if the derivative may be large, the iteration is still worth trying, because me may get lucky.

We conclude this introductory section by revisiting the problem of order. The inequalities that we obtained suggest that the flatter the function (the smaller the derivative), the better behavior we can expect from iteration. The smallest derivative is zero, then the function is constant, which is very flat indeed.

Theorem 19a.6.
Let φ be a function that is at least twice differentiable on some neighborhood of its fixed point x_f. If $\varphi'(x_f) = 0$, then for every sequence $\{x_f\}$ generated by (plain) iteration that converges to x_f there is $c > 0$ such that

$$|x_f - x_{k+1}| \leq c|x_f - x_k|^2.$$

This means that the iteration then behaves as if it were of order 2, which is very nice, indeed. We can also see from the proof that if, moreover, $\varphi''(x_f) = 0$, then the iteration behaves as a method of order three. However, while the former can be often achieved using the relaxation approach described below, with the second derivative we can only hope for a lucky coincidence.

19b. Relaxation

Relaxation is a certain procedure that can be used to (try to) improve convergence of iteration. The motivations is based on our observations above. Given an equation of the form $\varphi(x) = x$, we would appreciate if the function φ was flat. We can flatten it by multiplying that equation using some (small but) non-zero number traditionally denoted λ:

$$\lambda\varphi(x) = \lambda x.$$

That looks good, but unfortunately this is no longer a problem of the right type. We need to see x on the right. Dividing the equation by λ would fix this, but that would get us back to square one. Instead we will try another way to fix problems. There is a certain quantity missing on the right, so we will add it:

$$\lambda\varphi(x) + (1-\lambda)x = \lambda x + (1-\lambda)x$$
$$\implies \lambda\varphi(x) + (1-\lambda)x = x.$$

We obtained a new problem of the right type. Since all the operations were reversible (we never take $\lambda = 0$), this new problem has exactly the same solutions as before. We are in fact asking for fixed points of a new function

$$\varphi_\lambda(x) = \lambda\varphi(x) + (1-\lambda)x,$$

and by our observation, it has the same fixed points as the original $\varphi(x)$.
This approach is called relaxation and it is an idea that can be applied also in other settings. For us the important thing is that we can apply our usual iterative procedure to this new problem, and iterate with $\varphi_\lambda(x)$.
What are our expectations here? We expected the function $\lambda\varphi(x)$ to be flatter, but now we add another function $(1-\lambda)x$ to it, which changes the situation. There are two interesting interpretations. Both start with the idea of a weighted average.
Given two quantities A, B, we find the average of them using the formula $\frac{1}{2}A + \frac{1}{2}B$. It combines the two influences in such a way that both have the same impact. Sometimes we may want to consider one of the two influences more important, but to balance things out, we then have to downrate the other. Mathematically, we consider expressions of the form $\alpha A + \beta B$, where we require that $\alpha + \beta = 1$. This is called a weighted average. We can use the restriction to get rid of one parameter and write such a weighted average as $\alpha A + (1-\alpha)B$. We can imagine that α is a slider that moves our attention from one to another in varying degrees. Of course, we could also use the formula $(1-\beta)A + \beta B$.
Now we are ready to return to our relaxed iteration. It uses the following formula:

$$x_{k+1} = \lambda\varphi(x_k) + (1-\lambda)x_k.$$

We can see this as a weighted average of two iterations and λ is an indicator of our confidence in the original iteration. If we are happy with it, we can simply put $\lambda = 1$ and then $\varphi_\lambda(x) = \varphi(x)$, that is, we iterate $x_{k+1} = \varphi(x_k)$.
The other iteration in the mix is $x_{k+1} = x_k$ and it always produces constant and hence convergent sequence. So if we are not quite happy with the convergence of the original iteration, we can put some weight (or more weight) on this perfectly convergent iteration. However, we should not overdo it, because for $\lambda = 0$ we get the convergent procedure $x_{k+1} = x_k$ that has, unfortunately, nothing in common with our original problem.

Example 19b.a: We return to our equation $\cos(x) = x$. We will try to find an approximation of its solution with precision $\varepsilon = 0.0001$. To have a fair comparison, we will always use $x_0 = 1$ and the absolute difference stopping condition with this given ε.
To get some benchmark, we rewrite our equation as $\cos(x) - x = 0$ and apply the Newton method (of order two). It found its result after $N = 4$ iterations.
Now we try the plain fixed point iteration $x_{k+1} = \cos(x_k)$, and the procedure needed $N = 24$ iteration for comparable result. This was to be expected from a method of general order one.
Relaxed iteration uses the formula $x_{k+1} = \lambda\cos(x_k) + (1-\lambda)x_k$. We try several values for λ:

$\lambda =$	0.9	0.8	0.7	0.6	0.5	0.4	0.3
$N =$	14	9	6	4	7	10	14

This is a fairly typical picture. We see that the rate of convergence of iteration can significantly improve with the right relaxation, and when the relaxation parameter λ is about 0.6, iteration becomes comparable to the Newton method. Can it get even better? Trying values like 0.61 we find that not really, the four iteration benchmark is the best. This is also fairly typical, although there are cases when the relaxed iteration can even beat the Newton method.
$\triangle$

Algorithm 19b.1.
⟨relaxation for fixed point iteration⟩
Given: Function φ.
0. Define $\varphi_\lambda(x) = \lambda\varphi(x) + (1-\lambda)x$, where λ is the relaxation parameter.

1. Choose some value for λ, and proceed with the iteration method applied to this φ_λ.

2. If you are happy with the way iterations go, complete the iteration and obtain x_k, an approximation of fixed point of φ.

Otherwise repeat **1.** with different λ.

$\triangle$

Now why would anyone want to do that, given that after the first try we already have an approximation of our root? There is a situation where relaxation might help. Imagine that we repeatedly look for roots of functions that are very similar. Then it makes sense to expect (or hope) that if some relaxation parameter λ works for one, it will also work for others. So every time we look for a root, we try a different value of λ, and after a while we find what relaxation works well for the problems of similar type that will come afterwards.

If the function $\varphi(x)$ is given by a formula, we can also try to determine a suitable relaxation parameter λ before we actually start iterating. Recall that the general aim is to make the function

$$\varphi_\lambda(x) = \lambda\varphi(x) + (1-\lambda)x$$

very flat, to make its derivative as small as possible. The smallest derivative is zero, which leads to a natural requirement that $\varphi'_\lambda(x)$ should be zero. That is, we want

$$\lambda\varphi'(x) + (1-\lambda) = 0.$$

But we have two unknowns there, λ and x, which brings us to the natural question: Where do we want our φ_λ to be flat? If we want this to be true around some point x_c, then we get

$$\lambda\varphi'(x_c) + (1-\lambda) = 0 \implies \lambda = \frac{1}{1-\varphi'(x_c)}.$$

The natural location x_c is the root itself, because then we will get, according to the theorem above, a quadratic convergence. However, we do not know the root.

We can often at least estimate the location of the root, then we can optimize λ for that location. Another interesting idea is to optimize λ at each step of our iteration; that is, for every x_k we use the λ_k that would make the next iterative step best possible, by making φ_λ fast right where we need it most. On the other hand, such dynamic adjustment of λ requires extra calculations, which could slow down the algorithm and thus offset gains.

Example 19b.b: We return to the problem $\cos(x) = x$. We can guess by plotting the graph that this root is not far from 0.7, so we ask for $\varphi'_\lambda(0.7) = 0$. Since

$$\varphi'_\lambda(x) = -\lambda\sin(x) + (1-\lambda),$$

we deduce as above that the best relaxation parameter is

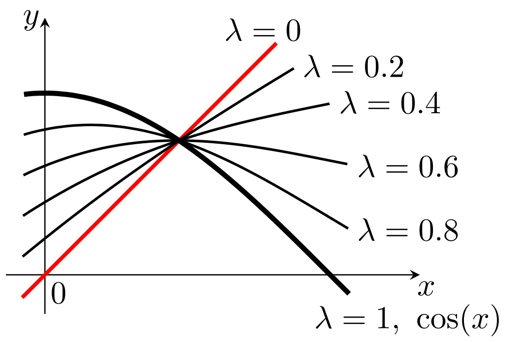
$$\lambda = \frac{1}{\sin(0.7)+1} \approx 0.61.$$

This matches rather well the results of our experiments above.

$\triangle$

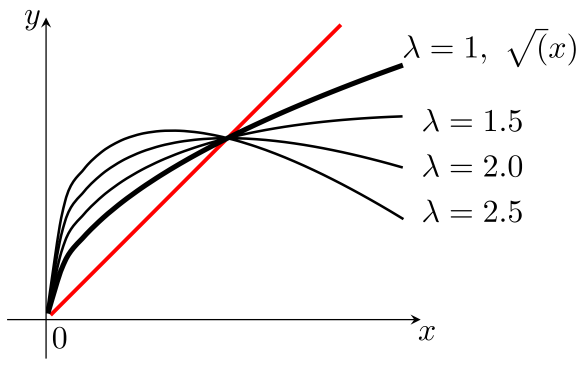
We talked about the relaxed function φ_λ creating a weighted average of two iterations, but we can also view the function itself as a weighted average of two functions, and to take this one step further, we can imagine that the graph of φ_λ is an average of graphs of $y = \varphi(x)$ and $y = x$. We can imagine that we have a slider, with $\lambda = 1$ we see the graph of the function $\varphi(x)$, and as we move the slider to 0, the graph morphs gradually into the graph of $f(x) = x$.

In the following picture we show how this works with the graph of $\varphi(x) = \cos(x)$.



Note that for $\lambda = 0.6$ we get a graph that is seemingly horizontal at the point of intersection, that is, at the fixed point. This, graphically, explains why this choice of parameter led to such a fast convergence.
When we ponder this example, we may get an insight that when a function φ is decreasing, then the best λ will be between 0 and 1. If the function φ is increasing but not as fast as $y = x$, then we would like to bend the graph somewhat downwards, which does not happen when we take intermediate versions of the compromise. We have to move in the direction of the graph of φ and then beyond, that is, the value of λ should be greater than 1, which is actually allowed.

Example 19b.c: Consider the function $\varphi(x) = \sqrt{x}$.
We observed earlier that iteration with a positive starting value should lead us to the fixed point $x_f = 1$. This pictures suggest that for λ around 2, the corresponding φ_λ looks rather flat around this fixed point. We will now make a series of experiments with common initial guess $x_0 = 2$, common tolerance and stopping condition, and different values of λ to test this. The following chart shows the umber of iterations necessary for the algorithm to stop. We include also the case $\lambda = 1$, that is, the plain iteration.



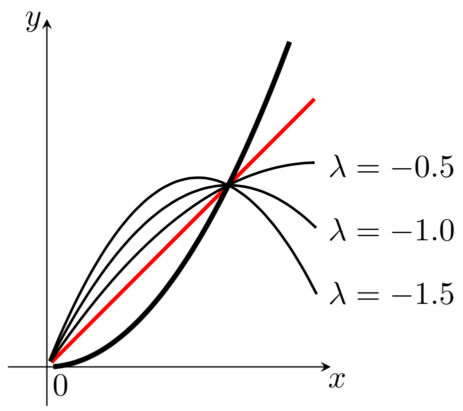
$\lambda =$	0.8	1.0	1.2	1.4	1.6	1.8	2.0	2.2
$N =$	17	13	10	8	6	5	4	5

The results match our expectations remarkably well. As expected, trying $\lambda < 1$ made actually the situation worse, as we actually bent the graph up more, so it got steeper.
Incidentally, the Newton method under the same conditions stops after $N = 4$ iterations.
$\triangle$

When a function grows faster than $y = x$, then we need to pull it towards $y = x$ and then even beyond, suggesting that negative values of λ may be of help.

Example 19b.d: Consider the function $\varphi(x) = x^2$.
We observed earlier that iteration with a starting value greater than 1 leads to a divergent iteration. This pictures suggest that for λ around -1, the corresponding φ_λ looks rather flat around the fixed point x_f. We will again try to see how experiments fit with this expectation, we will use the initial guess $x_0 = 1.5$.

$\lambda =$	0.2	0.0	-0.2	-0.4	-0.6	-0.8	-1.0	-1.2
$N =$	N	X	30	15	9	7	5	7



For $\lambda > 0$ we have divergence as expected. For $\lambda = 0$ the iteration does not make sense, as then it has nothing to do with our problem, we simply work with the formula $x_{k+1} = x_k$.

It is the negative lambdas that are interesting, and again we see the behavior that we expected. And again, we matched the performance of the Newton method.
It should be noted that the function φ_λ is curving rather sharply down as we move to the right, and definitely does not look like a contraction there. Indeed, if we try the starting point $x_0 = 2$, then we get divergent iterations for $\lambda < -1$, and the best performance is $N = 10$ for $\lambda \approx -0.98$, while the Newton method can still do with five iterations with this starting point. So this time we did not match this performance, but we did achieve success by saving iteration that originally diverged.
Another interesting feature of $x_0 = 2$ is that when we set $\lambda = -1$, the iteration immediately jumps to $x_f = 0$, the other fixed point, and naturally stays there, never getting to $x_f = 1$.
$\triangle$

Now we should have a good idea of what relaxation does for us. Before we move on, an important remark should be made. When we try to make φ_λ small somewhere, it only affects its shape around the point where we do it. If the iteration moves us elsewhere, we may be in trouble. This is related to the fact that making φ_λ into a contraction locally does not allow us to appeal to the conclusions of the Banach contraction theorem.
For that we would have to identify an interval I that is mapped by φ_λ into itself, which may be already quite a task given that the range most likely also depends on λ. Then we would have to show that φ_λ has a small derivative globally, on the whole interval I. This is a pretty serious work. We will therefore use the local flatness as an indication of what to expect, but we should be aware that this is nothing more than that, and that the actual iteration may behave differently.

19c. Roots, fixed points and relaxation

Obviously, the root approach and the fixed-point approach are closely related. Indeed, every equation of the form $f(x) = 0$ can be rewritten as $f(x)+x = x$, so the problem of finding a root of f can be replaced by the problem of finding a fixed point of $\varphi(x) = f(x)+x$. Conversely, $\varphi(x) = x$ can be written as $\varphi(x) - x = 0$, so finding a fixed point of φ can be done by finding a root of $f(x) = \varphi(x) - x$. To sum it up, finding a root and finding a fixed point is just two facets of the same problem.
Thus, given some algebraic equation, we can choose in which way we can approach it. The root approach offers high performance methods, on the other hand, the fixed-point approach seems somewhat more flexible.
This flexibility can be observed already when rewriting our equation. In particular, consider a root problem $f(x) = 0$. We changed it above into a fixed-point problem by adding x to each side. The corresponding iteration function $\varphi(x) = f(x) + x$ will be considered the **standard transformation** in this book.
Some people actually prefer a slight modification. before adding x they multiply the original equation by -1, arriving at the iteration function $\varphi(x) = x - f(x)$. There isn't much to choose from between these two approaches, and statistically they are about equally efficient. I chose the former as the standard in order to complete a nice circle. Given an equation $f(x) = 0$, we create the fixed-point version as $f(x)+x = x$, and the natural transformation into a root problem mentioned above (subtracting x) returns us back to where we started: $f(x) = 0$. But the version $x - f(x)$ also has its charm.
Whichever way we decide to play it, in many cases this does not make much difference as we prefer to do something else than the standard anyway.

Example 19c.a: Consider our traditional test problem $x^3 - x - 10 = 0$. We will try to find the solution using the fixed point approach, with initial guess $x_0 = 3$.

a) Standard transformation: Adding x to each side we obtain

$$x^3 - 10 = x \implies \varphi(x) = x^3 - 10.$$

What can we expect from this iteration? We have $\varphi'(x) = 3x^2$, where should we investigate it? We find by bracketing that the root is between 2 and 3, perhaps a bit closer to 2 judging from function values, so we will look at φ around 2 to have nice numbers. We have $\varphi'(2) = 12$. This indicates a very steep function, definitely very far from being a contraction. We thus feel that the standard iteration is perhaps not the best approach here.
If we did not know the rough location of the root, it would make sense to at least look at the situation at the place where the iteration starts, in our case at $x_0 = 3$. We get $\varphi'(3) = 27$, this is even worse.
Numerical experiment confirms that the iteration diverges, and quite badly at that.
Can it be salvaged using relaxation? We could use the formula for optimal λ that was deduced above, my preference is to remember the idea and apply it to problems. Here we go.
Relaxation means that we use the iterative function

$$\varphi_\lambda(x) = \lambda(x^3 - 10) + (1 - \lambda)x.$$

Its derivative is $3\lambda x^2 + (1 - \lambda) = 1 - \lambda(1 - 3x^2)$, and we want this to be zero when $x = 2$. We get

$$\lambda = \frac{1}{1 - 3 \cdot 2^2} = -\frac{1}{11} \approx -0.09.$$

Experiments show that with this λ the relaxed iteration actually converges to the right root, and with our traditional testing tolerance $\varepsilon = 0.001$ stops after $N = 8$ iterations. For comparison, the Newton method can make it in four, but we are glad to be able to salvage this divergent situation here.
If we decide to optimize our λ based on the starting point $x_0 = 3$, for instance because we would not know the location of the root, we would get $\lambda = -\frac{1}{26} \approx -0.28$. The iteration converges, this time with $N = 15$ iterations. This is not so surprising, we optimized φ_λ around 3, but the iteration moved elsewhere soon. Still, not knowing the location of the root, this would be the best we could do, and it did work.
However, there are other alternatives.

b) We can rewrite $x^3 - x - 10 = 0$ as $x^3 = x + 10$, then $x = \frac{x+10}{x^2}$ and we have a different fixed-point version of our question, this time $\varphi(x) = \frac{x+10}{x^2}$. What can we expect of it?
$\varphi'(x) = -\frac{x+20}{x^3}$, we are interested in $|\varphi'(3)| = \frac{23}{27}$. This is less than 1, so the situation looks hopeful. We try, and it turns out that this iteration also diverges, and quite badly at that. What happened? We quickly move elsewhere, and there the situation is worse. In particular, note that $|\varphi'(2)| = \frac{11}{4} = 2.75$, so φ stops being a contraction as we move towards the root.
However, now the derivatives are not as bad as in the previous attempt, so this time relaxation could be more successful. We will work with

$$\varphi_\lambda(x) = \lambda\frac{x + 10}{x^2} + (1 - \lambda)x,$$

and $\varphi'_\lambda(3) = 0$ happens for

$$\lambda = \frac{1}{1 + \frac{3+20}{3^3}} = \frac{27}{50} = 0.54.$$

Now the iteration converges, and stops after $N = 14$ iterations. Again, we saved the day.
By the way, now we know that the root is about 2.3. If we try to optimize λ for this location, we get $\lambda \approx 0.35$. Then the iteration stops after $N = 4$, and we matched the speed of the Newton method.
This version of $\varphi(x)$ is a good opportunity to note that we would actually prefer not to work with it when we have a viable alternative, because the x^2 in the denominator could lead to a division-by-zero error. The chances are small, but there will be other formulas with comparable or even better performance that do not have this problem.

c) We start with the previous rewrite $x^3 = x + 10$, but then simply write $x = (x + 10)^{1/3}$ and this time this fixed-point problem uses $\varphi(x) = (x + 10)^{1/3}$.

We have $\varphi'(x) = -\frac{1}{3}(x+10)^{-2/3}$, in particular $|\varphi'(3)| = \frac{1}{3}13^{-2/3} \approx 0.06$. This is very close to zero, so we are very hopeful. This way of rewriting the problem seems great right out of the box. Indeed, the plain iteration stops after $N = 5$ steps, which is just one step short of perfection. Relaxation is probably not worth our time.
This makes the main point of this example: Usually we can transform the given equation into a fixed-point problem in many ways, and often there is one that works just great without any further tinkering. In other words, the way in which we rewrite the problem may have greater impact than our attempts to save an unpleasant iteration using relaxation.
So let's practice some more.

d) We can rewrite $x^3 - x - 10 = 0$ as $x^3 - x = 10$, that is, $x(x^2 - 1) = 10$. The we can write $x = \frac{10}{x^2-1}$ and we have another version of fixed-point problem, this time $\varphi(x) = \frac{10}{x^2-1}$. We will be hoping that we never get $x_k = \pm 1$ in our runs, but the chances are small.
$\varphi'(x) = -\frac{20x}{(x^2-1)^2}$, we are interested in $|\varphi'(3)| = \frac{15}{16}$. This is less than 1, but only barely. We will try a run.
Interestingly, after a while the iteration started to flip between two distinct values (about -10.099 and 0.099) with just tiny changes.
Being so close to convergence just asks for relaxation. The optimal λ to make

$$\varphi_\lambda(x) = \lambda\frac{10}{x^2-1} + (1-\lambda)x$$

flat around $x_0 = 3$ is $\lambda = \frac{16}{31} \approx 0.52$.
Now the iteration converges, and stops after $N = 37$, which is quite a lot, this happens. Trying 0.6 we actually get a divergent oscillation, so we are playing it close to the edge here. On the other hand, smaller lambdas improve the runs. Optimizing around $x = 2.3$ yields $\lambda \approx 0.29$ and a run of respectable $N = 5$ steps.

e) Starting with $x(x^2 - 1) = 10$ again we can go to $x^2 - 1 = \frac{10}{x}$, then to $x^2 = 1 + \frac{10}{x}$ and finally to $x = \sqrt{1 + \frac{10}{x}}$.
The iterative function $\varphi(x) = \sqrt{1 + \frac{10}{x}}$ now features a double jeopardy, apart from division by zero (which will hopefully never happen) we now also have to worry about a negative argument under the root.
We have $\varphi'(x) = \frac{-5}{x^2\sqrt{1+\frac{10}{x}}}$, and around our initial guess we have $|\varphi'(3)| = \frac{5\sqrt{3}}{9\sqrt{13}} \approx 0.27$. This is significantly less than 1, so we are hopeful. Just to be on the safe side we check on what is happening towards the root: $|\varphi'(2)| = \frac{5}{4\sqrt{6}} \approx 0.51 < 1$. We are really hopeful.
The plain iteration converged after $N = 12$ steps, which is pretty good for an out-of-the-box iteration.
To finish this off, we optimize the shape of

$$\varphi_\lambda(x) = \lambda\sqrt{\frac{10}{x} + 1} + (1-\lambda)x$$

around $x_0 = 3$, obtaining $\lambda \approx 0.79$. The corresponding relaxed iteration stops after $N = 5$ steps, which is almost the best one can get. Optimizing for $x = 2.3$ we could probably improve this to $N = 4$, we leave it to the reader.
$\triangle$

As we just saw, given an algebraic equation, we can transform it into a fixed-point problem in many ways, some good, some not so good, and often we can significantly improve our runs by relaxation.

19c.1 Some connections

We will conclude this chapter with exploration of relaxation as applied to root-finding problems.

We start with a problem of the form $f(x)=0$. Applying the standard approach, we arrive at fixed-point iteration with $\varphi(x)=f(x)+x$. How would a relaxed iteration look like then?

$$\begin{aligned}\varphi_\lambda(x)&=\lambda\varphi(x)+(1-\lambda)x=\lambda\big(f(x)+x\big)+(1-\lambda)x\\&=\lambda f(x)+x.\end{aligned}$$

This is a very nice formula.

Now we try it differently. We may prefer not to remember formulas, but understand the ideas instead. The relaxation went as follows: We started with the original fixed-point equation and multiplied it by λ, then adjusted. Let us try the same process with the root equation:

$$\begin{aligned}f(x)&=0\\\lambda f(x)&=0\qquad \Big/ +x\\\lambda f(x)+x&=x\end{aligned}$$

We obtained a fixed point problem with $\varphi_\lambda(x)=\lambda f(x)+x$, exactly as before. So in the end it does not matter which approach we use, we end up with the same situation.

We may notice that the alternative standard transformation to a fixed-point problem $\varphi(x)=x-f(x)$ corresponds to our relaxation with $\lambda=-1$. But more can be done. How about this.

We start with the equation $f(x)=0$, and now we divide it by $-f'(x)$, assuming that it is not zero. We get

$$\begin{aligned}-\frac{f(x)}{f'(x)}&=0\qquad \Big/ +x\\x-\frac{f(x)}{f'(x)}&=x.\end{aligned}$$

We have $\varphi(x)=x-\frac{f(x)}{f'(x)}$. If we set up iteration for this approach, we obtain the iterating formula $x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)}$. Looks familiar? It should, it is the Newton formula from chapter 18. Wow! Turns out the Newton formula is just a special case of the fixed-point approach to finding roots, and how well it worked for us.

The last interesting point: COnsider the standard relaxed fixed-point iteration, that is, with

$$\varphi_\lambda(x)=\lambda f(x)+x.$$

Then $\varphi'(x)=\lambda f'(x)+1$. If we want to optimize around a certain point x_c, we want to have $\lambda f'(x_c)+1=0$, that is, $\lambda=-\frac{1}{f'(x_c)}$.

We now return to one idea that we entertained before but did not pursue it. What if we try to optimize the relaxation parameter at every step of iteration, thus making sure that all our iteration steps are as good as they could be? For x_k we get the optimal parameter $\lambda_k=-\frac{1}{f'(x_k)}$, our iteration then proceeds like this:

$$\begin{aligned}x_{k+1}&=\varphi_{\lambda_k}(x_k)=\lambda_k f(x_k)+x_k\\&=-\frac{1}{f'(x_k)}f(x_k)+x_k=x_k-\frac{f(x_k)}{f'(x_k)}.\end{aligned}$$

We again obtain the Newton formula. We conclude that the Newton method is actually the standard fixed point iteration, relaxed with relaxation parameter optimized at every step to provide the best possible performance. No wonder the Newton method is so fast.