CHAPTER 10

Zeros of Functions

An important part of the maths syllabus in secondary school is equation solving. This is important for the simple reason that equations are important — a wide range of problems can be translated into an equation, and by solving the equation we solve the problem. We will discuss a couple of examples in section 10.1.

In school you should have learnt to solve linear and quadratic equations, some trigonometric equations, some equations involving logarithms and exponential functions, as well as systems of two or three equations. Solving these equations usually follow a fixed recipe, and it may well be that you managed to solve all the equations that you encountered in school. For this reason you may believe that the problem of solving equations is — a solved problem.

The truth is that most equations cannot be solved by traditional pencil-and-paper methods. And even for equations that can be solved in this way, the expressions may become so complicated that they are almost useless for many purposes. Consider for example the equation

$$x^3 - 3x + 1 = 0. \tag{10.1}$$

The Norwegian mathematician Niels Henrik Abel proved that all polynomial equations of degree less than five can be solved by extracting roots, so we know there is a formula for the solutions. The program Mathematica will tell us that there are three solutions, one real and two complex. The real solution is given by

$$\frac{-20\sqrt[3]{\dfrac{3}{-9+\sqrt{12081}}} + \sqrt[3]{2\left(-9+\sqrt{12081}\right)}}{6^{2/3}}.$$

Although more complicated than the solution of a quadratic equation, this is not so bad. However, the solution becomes much more complicated for equations of degree 4. For example, the equation

$$x^4 - 3x + 1 = 0$$

has two complex and two real solutions, and one of the real solutions is given by

$$\begin{aligned}&\frac{\sqrt{\sqrt[3]{81-\sqrt{5793}}+\sqrt[3]{81+\sqrt{5793}}}}{2\sqrt[6]{2}\sqrt[3]{3}}\\ &\qquad + \frac{1}{2}\sqrt{\frac{1}{3}\left(-\sqrt[3]{\frac{3}{2}\left(81-\sqrt{5793}\right)}-\sqrt[3]{\frac{3}{2}\left(81+\sqrt{5793}\right)}\right.}\\ &\qquad\qquad\qquad \overline{\left.+\frac{18\sqrt[6]{2}\sqrt[3]{3}}{\sqrt{\sqrt[3]{81-\sqrt{5793}}+\sqrt[3]{81+\sqrt{5793}}}}\right)}\end{aligned}$$

(the square root in the second line extends to end of the third line).

In this chapter we are going to approach the problem of solving equations in a completely different way. Instead of looking for exact solutions, we are going to derive numerical methods that can compute approximations to the roots, with whatever accuracy is desired (or possible with the computer resources you have available). In most situations numerical approximations are also preferable for equations where the exact solutions can be found. For example the given root of the cubic equation above with 20 correct digits is -0.099900298805472842029, while the given solution of the quartic equation is 1.3074861009619814743 with the same accuracy. For most purposes this is much more informative than the large expressions above.

10.1 The need for numerical root finding

In this chapter we are going to derive three numerical methods for solving equations: the Bisection method, the Secant method and Newton's method. Before deriving these methods, we consider two practical examples where there is a need to solve equations.

10.1.1 Analysing difference equations

In chapter 6 we studied difference equations and saw that they can easily be simulated on a computer. However, we also saw that the computed solution may be completely overwhelmed by round-off errors so that the true solution is completely lost. Whether or not this will happen depends on the size of the roots of the characteristic equation of the difference equation. As an example, consider the difference equation

$$16x_{n+5} + 5x_{n+4} - 70x_{n+3} - 24x_{n+2} + 56x_{n+1} - 16x_n = 0$$

whose characteristic equation is

$$16r^5 + 5r^4 - 70r^3 - 24r^2 + 56r + 16 = 0.$$

It is impossible to find exact formulas for the roots of this equation. However, by using numerical methods like the ones we are going to derive in this chapter, one quite easily finds that the five roots are (with five-digit accuracy)

$$r_1 = -1.7761, \quad r_2 = -1.0985, \quad r_3 = -0.27959, \quad r_4 = 0.99015, \quad r_5 = 1.8515.$$

From this we see that the largest root is $r_5 \approx 1.85$. This means that regardless of the initial values, the computed (simulated) solution will eventually be dominated by the term r_5^n.

10.1.2 Labelling plots

A completely different example where there is a need for finding zeros of functions is illustrated in figure 10.1 which is taken from chapter 9. This figure has nine labels of the form $n = 2k - 1$ for $k = 1, \ldots, 9$, that are placed either directly above the point where the corresponding graph intersects the horizontal line $y = 2$ or below the point where the graph intersects the line $y = -2$. It would be possible to use an interactive drawing program and place the labels manually, but this is both tedious and time consuming, and it would be difficult to place all the labels consistently. With access to an environment for producing plots that is also programmable, it is possible to compute the exact position of the label.

Consider for example the label $n = 9$ which is to be placed above the point where the Taylor polynomial of $\sin x$, expanded about $a = 0$, intersects the line $y = 2$. The Taylor polynomial is given by

$$p(x) = x - \frac{x^3}{6} + \frac{x^5}{720} - \frac{x^7}{5040} + \frac{x^9}{362880}$$

so the x-value at the intersection point is given by the equation $p(x) = 2$, i.e., we have to solve the equation

$$x - \frac{x^3}{6} + \frac{x^5}{720} - \frac{x^7}{5040} + \frac{x^9}{362880} - 2 = 0.$$

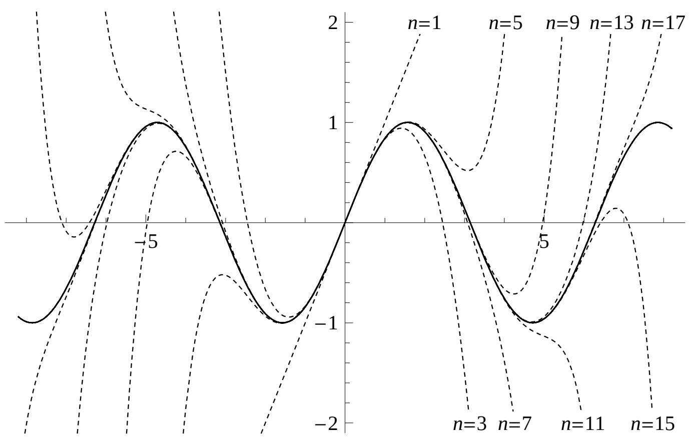

Figure 10.1. Plot with automatically placed labels.

This equation may have as many as nine real solutions, but from the plot we see that the one we are interested in is close to $x = 5$. Numerical methods for finding roots usually require a starting value near the root, and in our case it is reasonable to use 5 as starting value. If we do this and use a method like one of those derived later in this chapter, we find that the intersection between $p(x)$ and the horizontal line $y = 2$ is at $x = 5.4683$. This means that the label $n = 9$ should be drawn at the point with position $(5.4683, 2)$.

The position of the other labels may be determined similarly. In fact, this procedure may be incorporated in a program with a loop where k runs from 1 to 9. For each k, we determine the Taylor polynomial p_{2k-1} and plot it, compute the intersection x_k with $y = (-1)^{k+1}2$, and draw the label $n = 2k - 1$ at $\big(x_k, (-1)^{k+1}2\big)$. This is exactly how figure 10.1 was produced, using Mathematica.

10.2 The Bisection method

There are a large number of numerical methods for computing roots of equations, but the simplest of all is the *Bisection method.* Before we describe the method, let us review a basic mathematical result which forms the basis for the method.

10.2.1 The intermediate value theorem

The mean value theorem is illustrated in figure 10.2a. It basically says that if a function is positive at one point and negative at another, it must be zero somewhere in between.

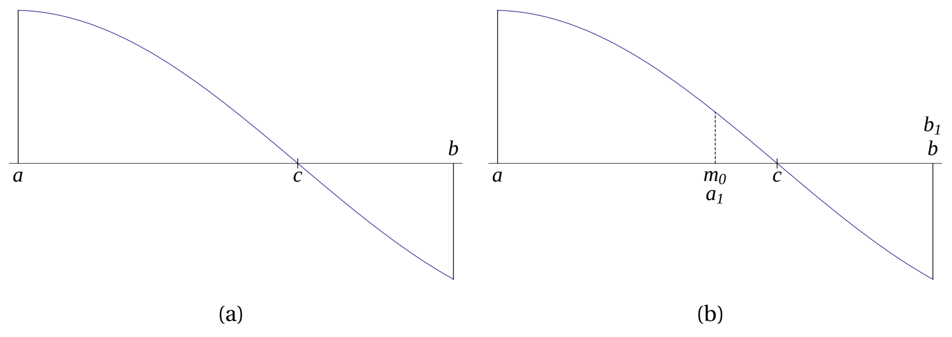

Figure 10.2. Illustration of the mean value theorem (a), and the first step of the Bisection method (b).

Theorem 10.1 (Intermediate value theorem)**.** *Suppose f is a function that is continuous on the interval $[a,b]$ and has opposite signs at a and b. Then there is a real number $c \in (a,b)$ such that $f(c) = 0$.*

This result seems obvious since f is assumed to be continuous, but the proof, which can be found in a standard calculus book, is not so simple. It may be easier to appreciate if we try to work with rational numbers only. Consider for example the function $f(x) = x^2 - 2$ on the interval $[a,b] = [0,2]$. This function satisfies $f(0) < 0$ and $f(2) > 0$, but there is no rational number c such that $f(c) = 0$. The zero in this case is of course $c = \sqrt{2}$, which is irrational, so the main content of theorem is basically that there are no gaps in the real numbers.

10.2.2 Derivation of the Bisection method

The intermediate value theorem only tells that f must have a zero, but it says nothing about how to find this zero. However, based on the theorem it is easy to devise a method for finding good approximations to the zero.

To start with, we know that f has opposite signs at the two ends of the interval $[a,b]$. Our aim is to find a new interval $[a_1,b_1]$, which is smaller than $[a,b]$, such that f also has opposite signs at the two ends of $[a_1,b_1]$. But this is not difficult: We use the midpoint $m_0 = (a+b)/2$ of the interval $[a,b]$ and compute $f(m_0)$. If $f(m_0) = 0$, we are very happy because we have found the zero. If this is not the case, the sign of $f(m_0)$ must either be equal to the sign of $f(a)$ or the sign of $f(b)$. If $f(m_0)$ and $f(a)$ have the same sign, we set $[a_1,b_1] = [m_0,b]$; if $f(m_0)$ and $f(b)$ have the same sign, we set $[a_1,b_1] = [a,m_0]$. The construction is illustrated in figure 10.2b.

The discussion in the previous paragraph shows how we may construct a new interval $[a_1,b_1]$, with a width that is half that of $[a,b]$, with the property that

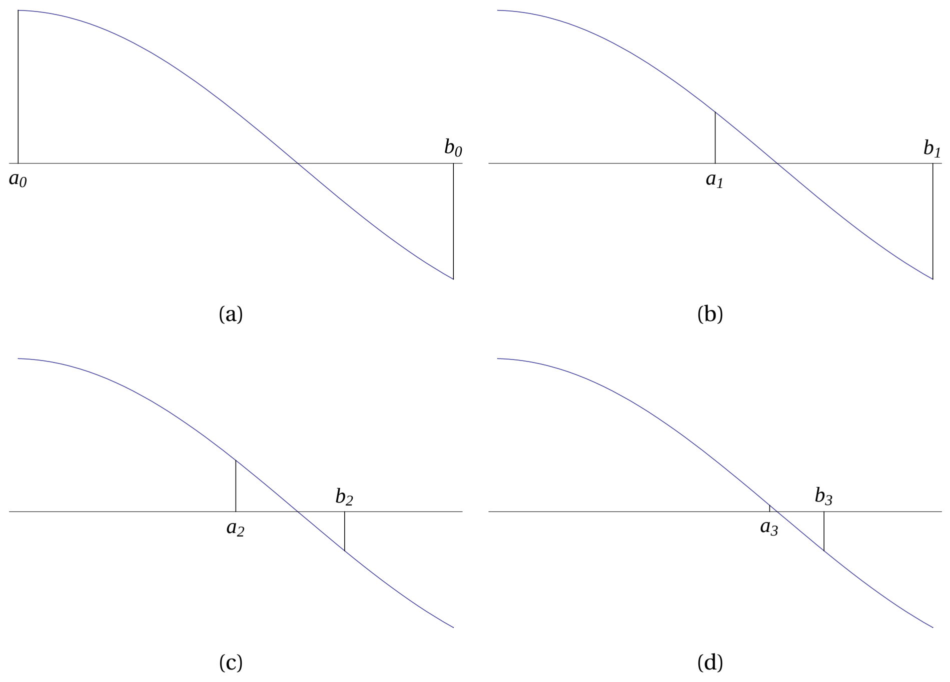

Figure 10.3. The first four steps of the bisection algorithm.

f is also guaranteed to have a zero in $[a_1, b_1]$. But then we may of course continue the process in the same way and determine another interval $[a_2, b_2]$ that is half the width of $[a_1, b_1]$, and such that f is guaranteed to have a zero in $[a_2, b_2]$. This process can obviously be continued until we hit a zero or the interval has become so small that the zero is determined with sufficient accuracy.

Algorithm 10.2 (Bisection method)**.** *Let f be a continuous function that has opposite signs at the two ends of the interval $[a, b]$. The following algorithm computes an approximation m_N to a zero $c \in (a, b)$ after N bisections:*

```
$a_0 := a$;
$b_0 := b$;
**for** $i := 1, 2, \dots, N$
    $m_{i-1} := (a_{i-1} + b_{i-1})/2$;
    **if** $f(m_{i-1}) = 0$
        $a_i := b_i := m_{i-1}$;
    **if** $f(a_{i-1}) f(m_{i-1}) < 0$
        $a_i := a_{i-1}$;
```

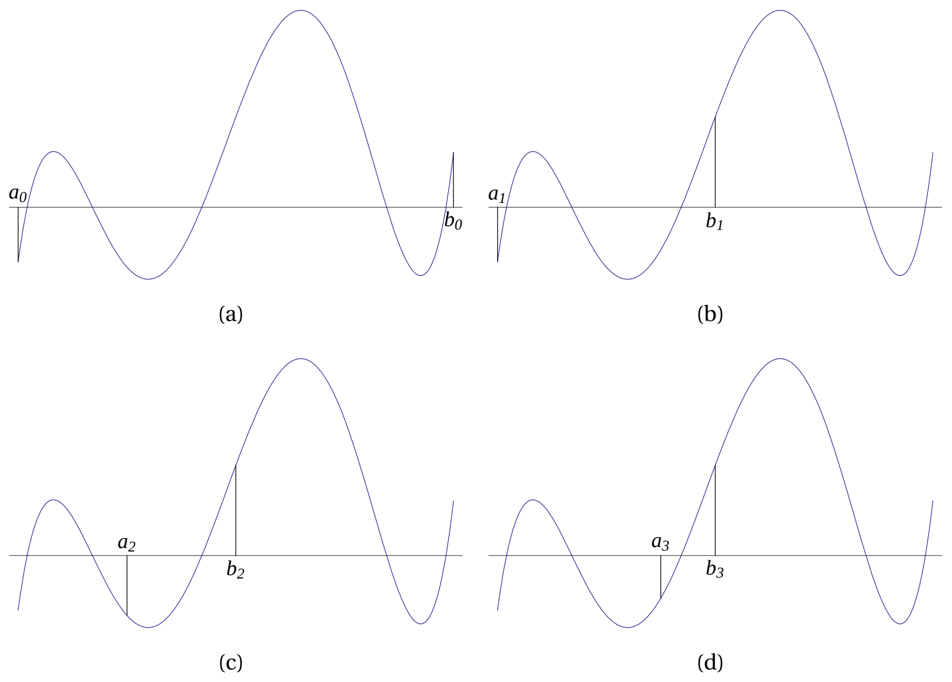

Figure 10.4. The first four steps of the bisection algorithm for a function with five zeros in the initial interval.

```
            $b_i := m_{i-1};$
        **else**
            $a_i := m_{i-1};$
            $b_i := b_{i-1};$

    $m_N := (a_N + b_N)/2;$
```

This algorithm is just a formalisation of the discussion above. The for loop starts with an interval $[a_{i-1}, b_{i-1}]$ with the property that $f(a_{i-1})f(b_{i-1}) < 0$. It usually produces a new interval $[a_i, b_i]$ of half the width of $[a_{i-1}, b_{i-1}]$, such that $f(a_i)f(b_i) < 0$. The exception is if we hit a zero c, then the width of the interval becomes 0. Initially, we start with $[a_0, b_0] = [a, b]$.

The first four steps of the Bisection method for the example in figure 10.2 are shown in figure 10.3. An example where there are several zeros in the original interval is shown in figure 10.4. In general, it is difficult to predict which zero the algorithm zooms in on, so it is best to choose the initial interval such that it only contains one zero.

10.2.3 Error analysis

Algorithm 10.18 does N subdivisions and then stops, but it would be more desirable if the loop runs until the error is sufficiently small. In order to do this, we need to know how large the error is.

If we know that a function f has a zero c in the interval $[a,b]$, and we use the midpoint $m = (a+b)/2$ as an approximation to the zero, what can we say about the error? The worst situation is if the zero is far from the midpoint, and the furthest from the midpoint we can get, is a or b in which case the error is $(b-a)/2$. This gives the following lemma.

Lemma 10.3. *Suppose f is a function with a zero c in the interval $[a,b]$. If the midpoint $m = (a+b)/2$ of $[a,b]$ is used as an approximation to c, the error is at most*

$$|c-m| \le \frac{b-a}{2}.$$

This simple tool is what we need to estimate the error in the Bisection method. Each bisection obviously halves the width of the interval, so the error is also halved each time.

Theorem 10.4. *Suppose f is a function with only one zero c in the interval $[a,b]$ and let $\{m_i\}$ denote the successive midpoints computed by the Bisection method. After N iterations, the error is bounded by*

$$|c-m_N| \le \frac{b-a}{2^{N+1}}. \tag{10.2}$$

As N tends to infinity, the midpoints m_N will converge to the zero c.

Here we have emphasised that f should have only one zero in $[a,b]$. If there are several zeros, an estimate like (10.2) still holds for one of the zeros, but it is difficult to say in advance which one.

This simple result allows us to control the number of steps necessary to achieve a certain error ϵ. For in order to ensure that the error is smaller than ϵ it is clearly sufficient to demand that the upper bound in the inequality (10.2) is smaller than ϵ,

$$|c-m_N| \le \frac{b-a}{2^{N+1}} \le \epsilon.$$

The second inequality can be solved for N by taking logarithms. This yields

$$\ln(b-a) - (N+1)\ln 2 \le \ln\epsilon$$

which leads to the following observation.

Observation 10.5. *Suppose that f has only one zero in $[a,b]$. If the number of bisections in the Bisection method is at least*

$$N \ge \frac{\ln(b-a) - \ln\epsilon}{\ln 2} - 1 \tag{10.3}$$

the error will be at most ϵ.

A simple word of advice: Do not try and remember the formula (10.3). It is much better to understand (and thereby remember) how it was derived.

Example 10.6. Suppose we want to find the zero $\sqrt{2}$ with error smaller than 10^{-10} by solving the equation $f(x) = x^2 - 2$. We have $f(1) = -1$ and $f(2) = 2$, so we can use the Bisection method, starting with the interval $[1,2]$. To get the error to be smaller than 10^{-10}, we know that N should be larger than

$$\frac{\ln(b-a) - \ln\epsilon}{\ln 2} - 1 = \frac{10\ln 10}{\ln 2} - 1 \approx 32.2.$$

Since N needs to be an integer this shows that $N = 33$ is guaranteed to make the error smaller than 10^{-10}. If we run algorithm 10.18 we find

$$\begin{aligned} m_0 &= 1.50000000000, \\ m_1 &= 1.25000000000, \\ m_2 &= 1.37500000000, \\ &\vdots \\ m_{33} &= 1.41421356233. \end{aligned}$$

We have $\sqrt{2} \approx 1.41421356237$ with eleven correct digits, and the actual error in m_{33} is approximately 4.7×10^{-11}. ∎

Recall that when we are working with floating-point numbers, the relative error is a better error measure than the absolute error. The relative error after i iterations is given by

$$\frac{|c - m_i|}{|c|}.$$

From the inequality (10.2) we have an upper bound on the numerator. Recall also that generally one is only interested in a rough estimate of the relative error. It is therefore reasonable to approximate c by m_i.

Observation 10.7. *After i iterations with the Bisection method, the relative error is approximately bounded by*

$$\frac{b-a}{|m_i|2^{i+1}}. \tag{10.4}$$

One may wonder if it is possible to estimate beforehand how many iterations are needed to make the relative error smaller than some given tolerance, like we did in observation 10.5 for the absolute error. This would require some a priori estimate of the zero c or the approximation m_i, which is hardly possible.

Recall from observation 5.3 that if the relative error in an approximation $\tilde{c}$ to c is of magnitude 10^{-m}, then c and $\tilde{c}$ have roughly m decimal digits in common. This is easily generalised to the fact that if the relative error is roughly 2^{-m}, then c and $\tilde{c}$ have roughly m binary digits in common. Observation 10.7 shows that the relative error in the Bisection method is roughly halved during each iteration (the variation in m_i will not vary much in magnitude with i). But this means that the number of correct bits increases by one in each iteration. Since 32-bit floating-point numbers use 24 bits for the significand and 64-bit floating-point numbers 54 bits, we can make the following observation.

Observation 10.8. *The number of correct bits in the approximations to a zero generated by the Bisection method increases by 1 per iteration. With 32-bit floating-point numbers, full accuracy is obtained after 24 iterations, while full accuracy is obtained after 54 iterations with 64-bit floating-point iterations.*

10.2.4 Revised algorithm

If we look back on algorithm 10.18, there are several improvements we can make. We certainly do not need to keep track of all the subintervals and midpoints, we only need the last one. It is therefore sufficient to have the variables a, b and m for this purpose. More importantly, we should use the idea from the previous section and let the number of iterations be determined by the requested accuracy. Given some tolerance $\epsilon > 0$, we could then estimate N as in observation 10.5. This is certainly possible, but remember that the absolute error may be an inadequate measure of the error if the numbers are far from 1 in size. Instead we should use the relative error. We use an integer counter i and the expression in (10.4) (with N replaced by i) to estimate the relative error. We stop the computations when i becomes so large that

$$\frac{b-a}{|m_i|2^{i+1}} \le \epsilon.$$

This condition becomes problematic if m_i should become 0. We therefore use the equivalent test

$$\frac{b-a}{2^{i+1}} \leq \epsilon |m_i|$$

instead. If m_i should become 0 for an i, this inequality will be virtually impossible to satisfy, so the computations will just continue.

Algorithm 10.9 (Revised Bisection method)**.** *Let f be a continuous function that has opposite signs at the two ends of the interval $[a,b]$. The following algorithm attempts to compute an approximation m to a zero $c \in (a,b)$ with relative error at most ϵ, using at most N bisections:*

```
$i = 0$;
$m := (a+b)/2$;
$abserr := (b-a)/2$;
while $i \leq N$ and $abserr > \epsilon|m|$
    if $f(m) = 0$
        $a := b := m$;
    if $f(a)f(m) < 0$
        $b := m$;
    else
        $a := m$;
    $i := i+1$;
    $m := (a+b)/2$;
    $abserr := abserr/2$;
```

In the while loop we have also added a test which ensures that the while loop does not run forever. This is good practice to ensure that your program does not enter an infinite loop because some unforeseen situation occurs that prevents the error from becoming smaller than ϵ.

What is called 'abserr' in the algorithm is of course not the exact absolute error, but rather the estimate in theorem 10.4. The estimate is halved for each iteration and this is used in the computation in the algorithm.

It must be emphasised that algorithm 10.9 lacks many details. For instance, the algorithm should probably terminate if $|f(m)|$ becomes small in some sense, not just when it becomes zero. And there is no need to perform more iterations than roughly the number of bits in the significand of the type of floating-point numbers used. However, the basic principle of the Bisection method is illustrated by the algorithm, and the extra details belong to the area of professional software development.

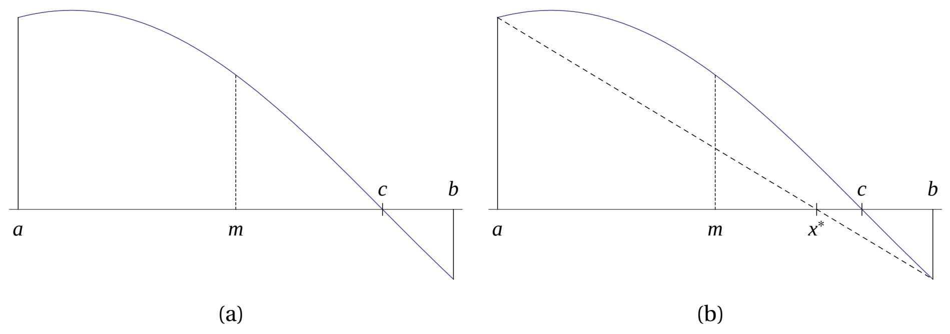

Figure 10.5. An example of the first step with the Bisection method (a), and the alternative approximation to the zero provided by the secant (b).

10.3 The Secant method

The Bisection method is robust and uses only the sign of $f(x)$ at the end points and the successive midpoints to compute an approximation to a zero. In many cases though, the method appears rather unintelligent. An example is shown in figure 10.5a. The values of f at a and b indicate that the zero should be close to b, but still the Bisection method uses the midpoint as the guess for the zero.

10.3.1 Basic idea

The idea behind the Secant method is to use the zero of the secant between $\big(a, f(a)\big)$ and $\big(b, f(b)\big)$ as an approximation to the zero instead of the midpoint, as shown in figure 10.5b.

Idea 10.10 (Secant idea)**.** *Let f be a continuous function, let a and b be two points in its domain, and let*

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

be the secant between the two points $\big(a, f(a)\big)$ and $\big(b, f(b)\big)$. The Secant method uses the zero

$$x^* = b - \frac{b - a}{f(b) - f(a)} f(b) \tag{10.5}$$

of the secant as an approximation to a zero of f.

We observe that the secant is symmetric in the two numbers a and b, so the

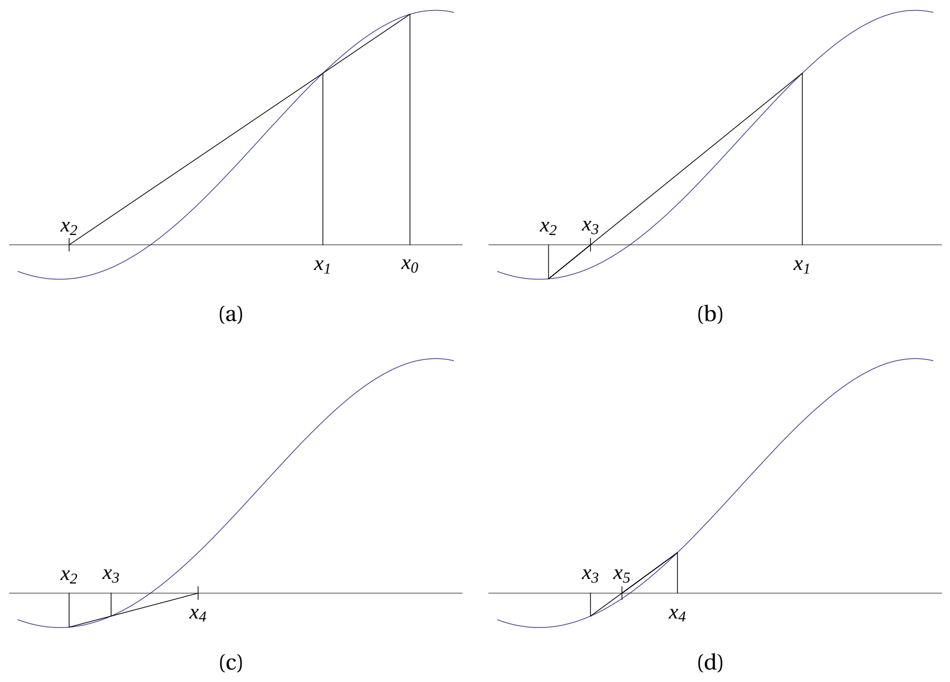

Figure 10.6. An example of the first four steps of the Secant method.

formula (10.8) may also be written

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

In the Secant method it is convenient to label a and b as $x_0 = a$ and $x_1 = b$ and denote the zero x^* by x_2. We are then in a position where we may repeat the formula: From the two numbers x_0 and x_1, we compute the approximate zero x_2, then from the two numbers x_1 and x_2 we compute the approximate zero x_3, from x_2 and x_3 we compute the approximate zero x_4, and so on. This is the basic Secant method, and an example of the first few iterations of the method is shows in figure 10.6. Note how the method quite quickly zooms in on the zero.

Algorithm 10.11 (Basic Secant method)**.** *Let f be a continuous function and let x_0 and x_1 be two given numbers in its domain. The sequence $\{x_i\}_{i=0}^N$ given by*

$$x_i = x_{i-1} - \frac{x_{i-1} - x_{i-2}}{f(x_{i-1}) - f(x_{i-2})} f(x_{i-1}), \quad i = 2, 3, \dots, N, \tag{10.6}$$

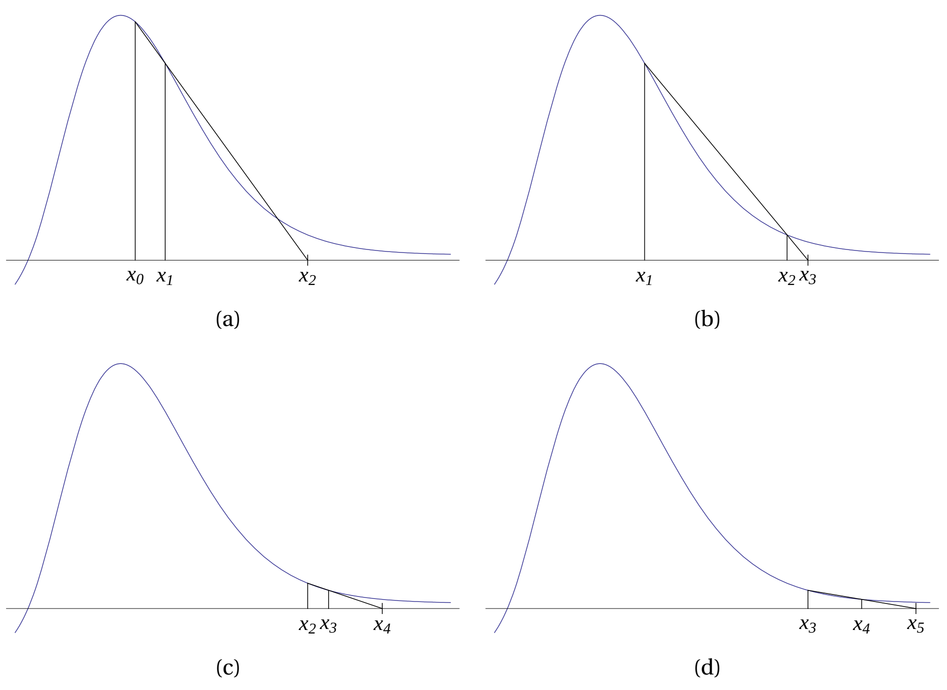

Figure 10.7. An example where the Secant method fails.

will in certain situations converge to a zero of f.

It is important to realise that unlike the Bisection method, the Secant method may fail. One such example is shown in figure 10.7. The problem here is that the two starting values are too far away from the zero to the left in the plot, and the algorithm gets stuck in the area to the right where the function is small, without ever becoming 0. This explains the expression "will in certain situations converge" in algorithm 10.11.

10.3.2 Testing for convergence

Algorithm 10.11 provides the basis for the secant algorithm. However, rather than just do N iterations, it would be more useful to stop the iterations when a certain accuracy has been attained. This turns out to be more difficult for the Secant method than for the Bisection method since there is not such an explicit error estimate available for the Secant method.

The Secant method produces a sequence of approximations x_0, x_1, ... to a zero, and we want to decide when we are within a tolerance ϵ of the zero. We will often find ourselves in this kind of situation: Some algorithm produces a sequence of approximations, and we want to check whether we have convergence.

When we come close to the zero, the difference between successive approximations will necessarily become small. If we are working with the absolute error, it is therefore common to use the number $|x_n - x_{n-1}|$ as a measure of the absolute error at iteration no. n. If we want to stop when the absolute error is smaller than ϵ, the condition then becomes $|x_n - x_{n-1}| \leq \epsilon$.

Usually, it is preferable to work with the relative error, and then we need an estimate for the zero as well. At step n of the algorithm, the best approximation we have for the zero is the latest approximation, x_n. The estimate for the relative error at step n is therefore

$$\frac{|x_n - x_{n-1}|}{|x_n|}.$$

To test whether the relative error is less than or equal to ϵ, we would then use the condition $|x_n - x_{n-1}| \leq \epsilon |x_n|$. We emphasise that this is certainly not exact, and this kind of test cannot guarantee that the error is smaller than ϵ. But in the absence of anything better, this kind of strategy is often used.

Observation 10.12. *Suppose that an algorithm generates a sequence $\{x_n\}$. The absolute error in the terms x_n is then often estimated by $|x_n - x_{n-1}|$, and the relative error by $|x_n - x_{n-1}|/|x_n|$. To test whether the relative error is smaller than ϵ, the condition*

$$|x_n - x_{n-1}| \leq \epsilon |x_n|$$

is often used.

When computing zeros of functions, there is one more ingredient that is often used. At the zero c we obviously have $f(c) = 0$. It is therefore reasonable to think that if $f(x_n)$ is small, then x_n is close to a zero. It is easy to construct functions where this is not the case. Consider for example the function $f(x) = x^2 + 10^{-30}$. This function is positive everywhere, but becomes as small as 10^{-30} at $x = 0$. Without going into further detail, we therefore omit this kind of convergence testing, although it may work well in certain situations.

10.3.3 Revised algorithm

The following is a more detailed algorithm for the Secant method, where the test for convergence is based on the discussion above.

Algorithm 10.13 (Revised Secant method)**.** *Let f be a continuous function, and let x_0 and x_1 be two distinct initial approximations to a zero of f. The following algorithm attempts to compute an approximation z to a zero with relative error less than $\epsilon < 1$, using at most N iterations:*

```
$i = 0$;
$xpp := x_0$;
$xp := z := x_1$;
$abserr := |z|$;
while $i \le N$ and $abserr \ge \epsilon|z|$
    $z := xp - f(xp)(xp - xpp)/\big(f(xp) - f(xpp)\big)$;
    $abserr := |z - xp|$;
    $xpp := xp$;
    $xp := z$;
```

Since we are only interested in the final approximation of the root, there is no point in keeping track of all the approximations. All we need to compute the next approximation z, is the two previous approximations which we call xp and xpp, just like in simulation of second order difference equations (in fact, the iteration (10.6) in the Secant method can be viewed as the simulation of a non-linear, second-order, difference equation). Before we enter the while loop, we have to make sure that the test of convergence does not become true straight-away. The first time through the loop, the test for convergence is $|z| \ge \epsilon|z|$ which will always be true (even if $z = 0$), since ϵ is assumed to be smaller than 1.

10.3.4 Convergence and convergence order of the Secant method

So far we have focused on the algorithmic aspect of the Secant method, but an important question is obviously whether or not the sequence generated by the algorithm converges to a zero. As we have seen, this is certainly not always the case, but if f satisfies some reasonable conditions and we choose the starting values near a zero, the sequence generated by the algorithm will converge to the zero.

Theorem 10.14 (Convergence of the Secant method)**.** *Suppose that f and its first two derivatives are continuous in an interval I that contains a zero c of f, and suppose that there exists a positive constant γ such that $|f'(x)| > \gamma > 0$ for all x in I. Then there exists a constant K such that for all starting values x_0 and x_1 sufficiently close to c, the sequence produced by the Secant method* *will converge to c and the error $e_n = c - x_n$ will satisfy*

$$|e_n| \leq K|e_{n-1}|^r, \quad n = 2, 3, \ldots, \tag{10.7}$$

where

$$r = \frac{1}{2}(1+\sqrt{5}) \approx 1.618.$$

We are not going to prove this theorem which may appear rather overwhelming, but let us comment on some of the details.

First of all we note the assumptions: The function f and its first two derivatives must be continuous in an interval I that contains the zero c. In addition $|f'(x)|$ must be positive in this interval. This is always the case as long as $f'(c) \neq 0$, because then $f'(x)$ must also be nonzero near c. However, the Secant method works even if $f'(c) = 0$, it will just require more iterations than in the case when $f'(c)$ is nonzero.

The other assumption is that the starting values x_0 and x_1 are "sufficiently close to c". This is imprecise, but means that it is in fact possible to write down precisely how close x_0 and x_1 must be to c.

Provided the assumptions are satisfied, theorem 10.14 guarantees that the Secant method will converge to the zero. However, the inequality (10.7) also says something about how quickly the error goes to zero. Suppose that at some stage we have $e_k = 10^{-1}$ and that K is some number near 1. Then we find that

$$\begin{aligned}
e_{k+1} &\lesssim e_k^r = 10^{-r} \approx 10^{-1.618} \approx 2.41 \times 10^{-2},\\
e_{k+2} &\lesssim e_{k+1}^r \lesssim e_k^{r^2} = 10^{-r^2} \approx 2.41 \times 10^{-3},\\
e_{k+3} &\lesssim 5.81 \times 10^{-5},\\
e_{k+4} &\lesssim 1.40 \times 10^{-7},\\
e_{k+5} &\lesssim 8.15 \times 10^{-12},\\
e_{k+6} &\lesssim 1.43 \times 10^{-18}.
\end{aligned}$$

This means that if the size of the root is approximately 1, and we manage to get the error to become 0.1, it will be as small as 10^{-18} (machine precision with 64-bit floating-point numbers) only six iterations later.

Observation 10.15. *When the Secant method converges to a zero c with $f'(c) \neq 0$, the number of correct digits increases by about 62 % per iteration.*

Example 10.16. Let us see if the predictions above happen in practice. We test the Secant method on the function $f(x) = x^2 - 2$ and attempt to compute the zero $c = \sqrt{2} \approx 1.41421356237309505$. We start with $x_0 = 2$ and $x_1 = 1.5$ and obtain

$$\begin{aligned}
x_2 &\approx 1.42857142857142857, & e_2 &\approx 1.4 \times 10^{-2},\\
x_3 &\approx 1.41463414634146341, & e_3 &\approx 4.2 \times 10^{-4},\\
x_4 &\approx 1.41421568627450980, & e_4 &\approx 2.1 \times 10^{-6},\\
x_5 &\approx 1.41421356268886964, & e_5 &\approx 3.2 \times 10^{-10},\\
x_6 &\approx 1.41421356237309529, & e_6 &\approx 2.4 \times 10^{-16}.
\end{aligned}$$

This confirms the claim in observation 10.15. ∎

10.4 Newton's method

We are going to study a third method for finding roots of equations, namely Newton's method. This method is quite similar to the Secant method, and the description is quite similar, so we will be brief.

10.4.1 Basic idea

In the Secant method we used the secant as an approximation to f and the zero of the tangent as an approximation to the zero of f. In Newton's method we use the tangent of f instead.

Idea 10.17 (Newton's method)**.** *Let f be a continuous, differentiable function, let a be a point in its domain, and let*

$$T(x) = f(a) + f'(a)(x - a)$$

be the tangent of f at a. Newton's method uses the zero

$$x^* = a - \frac{f(a)}{f'(a)} \tag{10.8}$$

of the tangent as an approximation to a zero of f.

Newton's method is usually iterated, just like the Secant method. So if we start with $x_0 = a$, we compute the zero x_1 of the tangent at x_0. Then we repeat and compute the zero x_2 of the tangent at x_1, and so on,

$$x_n = x_{n-1} - \frac{f(x_{n-1}}{f'(x_{n-1})}, \quad n = 1, 2, \ldots \tag{10.9}$$

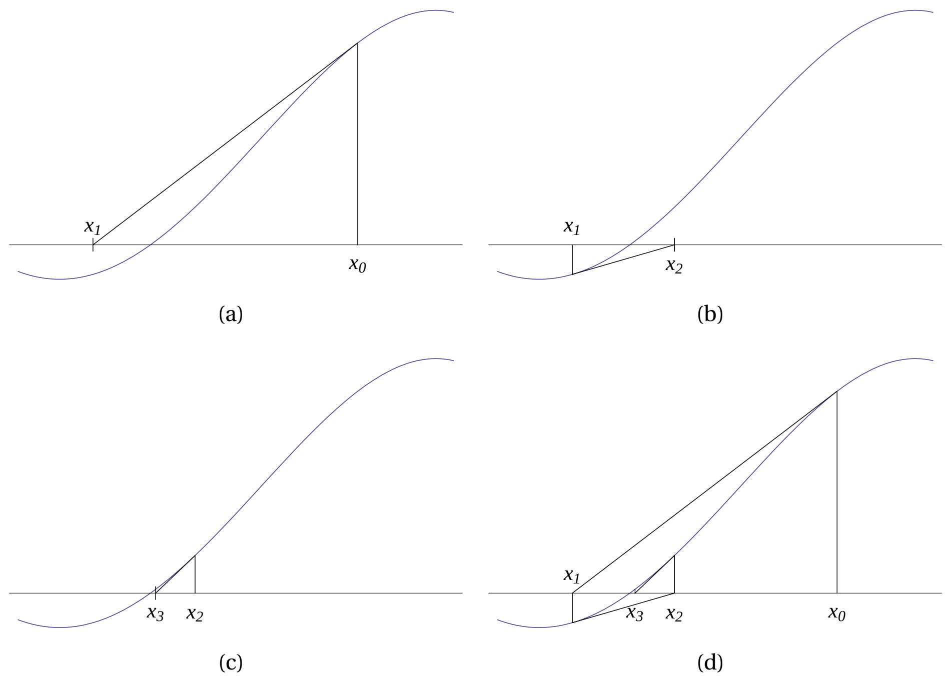

Figure 10.8. An example of the first three steps of Newton's method (a)–(c). The plot in shows a standard way of illustrating all three steps in one figure.

The hope is that the resulting sequence $\{x_n\}$ will converge to a zero of f. Figure 10.8 illustrates the first three iterations with Newton's method for the example in figure 10.6.

An advantage of Newton's method compared to the Secant method is that only one starting value is needed since the iteration (10.9) is a first-order (nonlinear) difference equation. On the other hand, it is sometimes a disadvantage that an explicit expression for the derivative is required.

10.4.2 Algorithm

Newton's method is very similar to the Secant method, and so is the algorithm. We measure the relative error in the same way, and therefore the stopping criterion is also exactly the same.

Algorithm 10.18 (Newton's method)**.** *Let f be a continuous, differentiable function, and let x_0 be an initial approximation to a zero of f. The following algorithm attempts to compute an approximation z to a zero with relative*

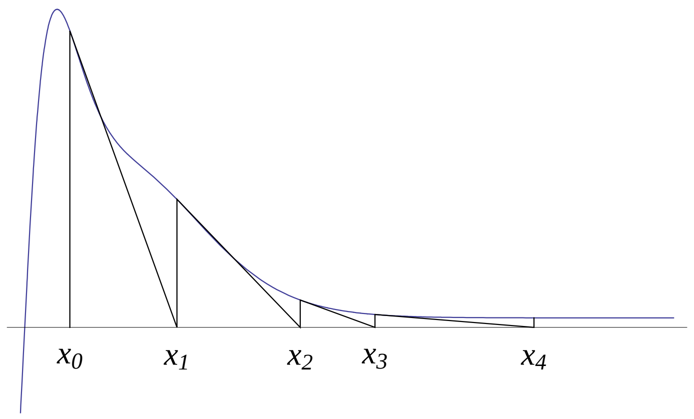

Figure 10.9. An example where Newton's method fails to converge because of a bad starting value.

error less than $\epsilon < 1$, using at most N iterations:

```
$i = 0$;
$xp := z := x_0$;
$abserr := |z|$;
while $i \le N$ and $abserr \ge \epsilon|z|$
    $z := xp - f(xp)/f'(xp)$;
    $abserr := |z - xp|$;
    $xp := z$;
```

What may go wrong with this algorithm is that, like the Secant method, it may not converge, see the example in figure 10.9. Another possible problem is that we may in rare cases get division by zero in the first statement in the while loop.

10.4.3 Convergence and convergence order

The behaviour of Newton's method is very similar to that of the Secant method. One difference is that Newton's method is in fact easier to analyse since it is a first-order difference equation. The equivalent of theorem 10.14 is therefore easier to prove in this case. The following lemma is a consequence of Taylor's formula.

Lemma 10.19. *Let c be a zero of f which is assumed to have continuous derivatives up to order 2, and let $e_n = x_n - c$ denote the error at iteration n in Newton's method. Then*

$$e_{n+1} = e_n^2 \frac{f''(\xi_n)}{2f'(x_n)}, \tag{10.10}$$

where ξ_n is a number in the interval (c, x_n) (the interval (x_n, c) if $x_n < c$).

Proof. The basic relation in Newton's method is

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

If we subtract the zero c on both sides we obtain

$$e_{n+1} = e_n - \frac{f(x_n}{f'(x_n)} = \frac{e_n f'(x_n) - f(x_n)}{f'(x_n)}. \tag{10.11}$$

Consider now the Taylor exansion

$$f(c) = f(x_n) + (c - x_n)f'(x_n) + \frac{(c - x_n)^2}{2} f''(\xi_n),$$

where ξ_n is a number in the interval (x_n, c). Since $f(c) = 0$, this may be rewritten as

$$-f(x_n) + (x_n - c)f'(x_n) = \frac{(c - x_n)^2}{2} f''(\xi_n).$$

If we insert this in (10.11) we obtain the relation

$$e_{n+1} = e_n^2 \frac{f''(\xi_n)}{2f'(x_n)},$$

as required. ∎

Lemma 10.19 is the basis for proving that Newton's method converges. The result is the following theorem.

Theorem 10.20. *Suppose that f and its first two derivatives are continuous in an interval I that contains a zero c of f, and suppose that there exists a positive constant γ such that $|f'(x)| > \gamma > 0$ for all x in I. Then there exists a constant K such that for all initial values x_0 sufficiently close to c, the sequence produced by Newton's method will converge to c and the error $e_n = x_n - c$ will satisfy*

$$|e_{n+1}| \le K|e_n|^2, \quad n = 1, 2, \dots \tag{10.12}$$

where K is some nonzero constant.

We will not prove this theorem, just comment on a few details. First of all we note that the assumptions are basically the same as the assumptions in the similar theorem 10.14 for the Secant method. The essential condition is that $f'(c) \neq 0$. Without this, the method still works, but the convergence is very slow.

The inequality (10.12) is obtained from (10.10) by taking the maximum of the expression $f''(x)/f'(y)$ on the right in (10.12), for all x and y in the interval I. If $f'(c) = 0$ this constant will not exist.

When we know that Newton's method converges, the relation (10.12) tells us how quickly it converges. If at some stage we have obtained $e_k \approx 10^{-1}$ and $K \approx 1$, we see that

$$\begin{aligned} e_{k+1} &\approx e_k^2 \approx 10^{-2}, \\ e_{k+2} &\approx e_{k+1}^2 \approx 10^{-4}, \\ e_{k+3} &\approx e_{k+2}^2 \approx 10^{-8}, \\ e_{k+4} &\approx e_{k+3}^2 \approx 10^{-16}. \end{aligned}$$

This means that if the root is approximately 1 in size and we somehow manage to reach an error of about 10^{-1}, we only need four more iterations to reach machine accuracy with 64-bit floating-point numbers. This shows the power of the relation (10.12). An algorithm for which the error satisfies this kind of relation is said to be *quadratically convergent*.

Observation 10.21. *When Newon's method converges to a zero c for which $f'(c) \neq 0$, the number of correct digits doubles per iteration.*

Let us end by redoing example 10.16 with Newton's method and checking observation 10.21 on a practical example.

Example 10.22. The equation is $f(x) = x^2 - 2$ which has the solution $c = \sqrt{2} \approx 1.41421356237309505$. If we run Newton's method with the initial value $x_0 = 1.7$, we find

$$\begin{aligned} x_1 &\approx 1.43823529411764706, & e_2 &\approx 2.3 \times 10^{-1}, \\ x_2 &\approx 1.41441417057620594, & e_3 &\approx 2.4 \times 10^{-2}, \\ x_3 &\approx 1.41421357659935635, & e_4 &\approx 2.0 \times 10^{-4}, \\ x_4 &\approx 1.41421356237309512, & e_5 &\approx 1.4 \times 10^{-8}, \\ x_5 &\approx 1.41421356237309505, & e_6 &\approx 7.2 \times 10^{-17}. \end{aligned}$$

We see that although we only use one starting value, which is further from the root than best of the two starting values used with the Secant method, we still end up with a smaller error than with the Secant method after five iterations. ∎

10.5 Summary

We have considered three methods for computing zeros of functions, the Bisection method, the Secant method, and Newton's method. The Bisection method is robust and works for almost any kind of equations and any kind of zeros, but the convergence is relatively slow. The other two methods converge much faster when the root c is simple, i.e., when $f'(c) \neq 0$. The Secant method is then a bit slower than Newton's method, but it does not require knowledge of the derivative of f.

If $f'(c) = 0$, the Bisection method still converges with the same speed, as long as an interval where f has opposite signs at the ends can be found. In this situation the Secant method and Newton's method are not much faster than the Bisection method.

A major problem with all three methods is the need for starting values. This is especially true for the Secant method and Newton's method which may easily diverge if the starting values are not good enough. There are other, more advanced, methods available which converge as quickly as Newton's method, without requiring precise starting values.