LECTURE 2: Locating roots of equations

In this chapter, we consider the task of solving equations numerically. The problem can be written out in the following form:

$$f(x) = 0$$

where all the terms of the equation have been moved to the left hand side, thus giving us the problem of finding the zero or zeros of a function f (assuming that there is only one independent variable x). Despite the apparent notational simplicity, this problem can often be difficult to solve. In many case, the success depends on having a good first guess for the solution. Throughout the chapter, we emphasize the importance of never using numerical routines as "black boxes" without understanding them (and under what circumstances they may fail).

1 Introduction

Why is locating roots of equations important?

For example, consider a certain electrical circuit in which the voltage V and the current I are related by two equations of the form

$$\begin{aligned} I &= a(e^{bV} - 1) \\ c &= dI + V \end{aligned}$$

where a, b, c and d are constants. Combining the two equations gives

$$c = ad(e^{bV} - 1) + V$$

and the solution to this equation is required. This can be rewritten in the form

$$f(V) = ad(e^{bV} - 1) + V - c = 0$$

The solution to the problem is now given by the root of the equation above, or equivalently, by the zero of f.

There is a large collection of methods available for locating zeros of functions. The three most useful ones are considered in this course: the bisection method, Newton's method, and the secant method. The fundamental strategy behind all these methods is to replace a difficult problem with a string of simpler ones and carry out an iterative process, expecting the solutions to converge to the solution of the original problem.

2 Bisection method

2.1 Intermediate-value theorem of continuous functions

Theorem. If the function f is continuous on the closed interval $[a,b]$, and if $f(a) \leq y \leq f(b)$ or $f(b) \leq y \leq f(a)$, then there exists a point c such that $a \leq c \leq b$ and $f(c) = y$.

We can use the intermediate value theorem in our task of locating roots of equations. Let f be a continuous function that has values of opposite signs at the two ends of the interval $[a,b]$. This means that

$$f(a)f(b) < 0$$

We can now say that the root is *bracketed* in the interval $[a,b]$. It follows that there must exist a number r that satisfies the two conditions: $a < r < b$ and $f(r) = 0$. Thus we know that at least one root must lie in this interval. This reasoning is utilized in the bisection algorithm.

Once we know that an interval contains a root, there are several procedures for locating the root. These procedures have varying degrees of speed and reliability. Unfortunately, the methods that are guaranteed to converge are often very slow, while the quick ones are less certain and may fail to find the correct answer.

Therefore, it is important to understand the strengths and weaknesses of the various methods and learn to recognize situations in which problems may arise.

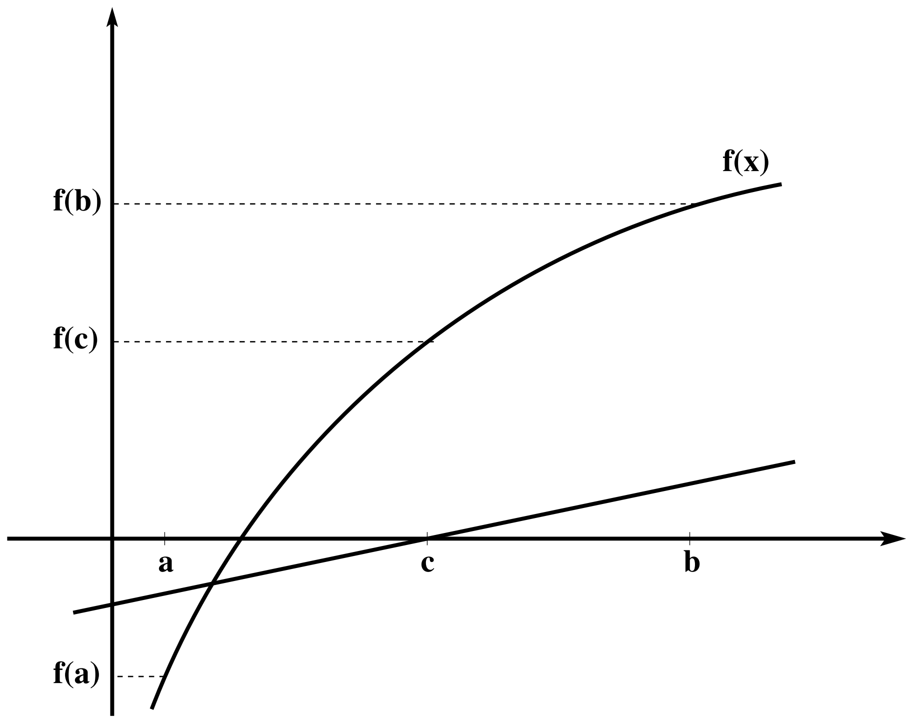


Figure 1: Illustration of the bisection method.

2.2 Bisection algorithm

The bisection algorithm is one that cannot fail (assuming that we know reasonable initial values). The idea is simple. A function is known to have a zero at some interval because it is continuous and changes sign. Evaluate the function at the interval's middle point and examine the sign. Use the midpoint to replace whichever limit has the same sign. This procedure is illustrated in Fig. 1.

After each iteration the bounds containing the root decrease by a factor of two. Eventually, the root will be found with a given precision. If the interval contains more than one root, the bisection method will find one of them.

To formulate this into code, we use the following notation. The interval $[a,b](a<b)$ is needed as an input. Set $u=f(a)$ and $v=f(b)$. The numbers u and v must satisfy: $uv<0$. At each iteration step, the interval is divided in half. The new interval is taken to be the one in which the solution is found.

The algorithm can be written as follows:

```
i.   If $b-a \le tol_1$, stop.
ii.  Set $c=(a+b)/2$.
     Calculate $w=f(c)$.
     If $|w| \le tol_2$, stop.
iii. If $wu<0$, set $b \leftarrow c$.
     Else set $a \leftarrow c$.
     Go to i.
```

There are two conditions on which to stop: if we are sufficiently close to the root (the interval $b-a$ is close to zero) or if the value of the function is sufficiently small ($|f| \approx 0$). In practice, we also need to assign a maximum for the number of iteration loops.

Example

Consider the function $f(x)=x^2-2$ as an example. The following piece of C code uses the bisection method for locating the root of the equation $f(x)=0$.

```
 /* Step i. */
 if(b-a <= tol1) break;

/* Step ii. */
 c = 0.5*(a+b);
 w = func(c);
 if(fabs(w) <= tol2) break;

/* Step iii. */
 if(w*u < 0) {
   b = c;
   v = w;
 }
 else {
   a = c;
   u = w;
 }
```

The solution is obtained in 21 iterations. (The starting interval is $[-1.1,2.1]$.) Output of the program:

```
i     c         f(c)
0  0.500000 -1.750000
1  1.300000 -0.310000
2  1.700000  0.890000
3  1.500000  0.250000
4  1.400000 -0.040000
5  1.450000  0.102500
6  1.425000  0.030625
7  1.412500 -0.004844
8  1.418750  0.012852
9  1.415625  0.003994
10 1.414062 -0.000427
11 1.414844  0.001783
12 1.414453  0.000678
13 1.414258  0.000125
14 1.414160 -0.000151
15 1.414209 -0.000013
16 1.414233  0.000056
17 1.414221  0.000022
18 1.414215  0.000004
19 1.414212 -0.000004
20 1.414214 -0.000000
```

Figure 2 shows the function $f(x) = x^2 - 2$ and a plot of the output of the program (plotted using MATLAB).

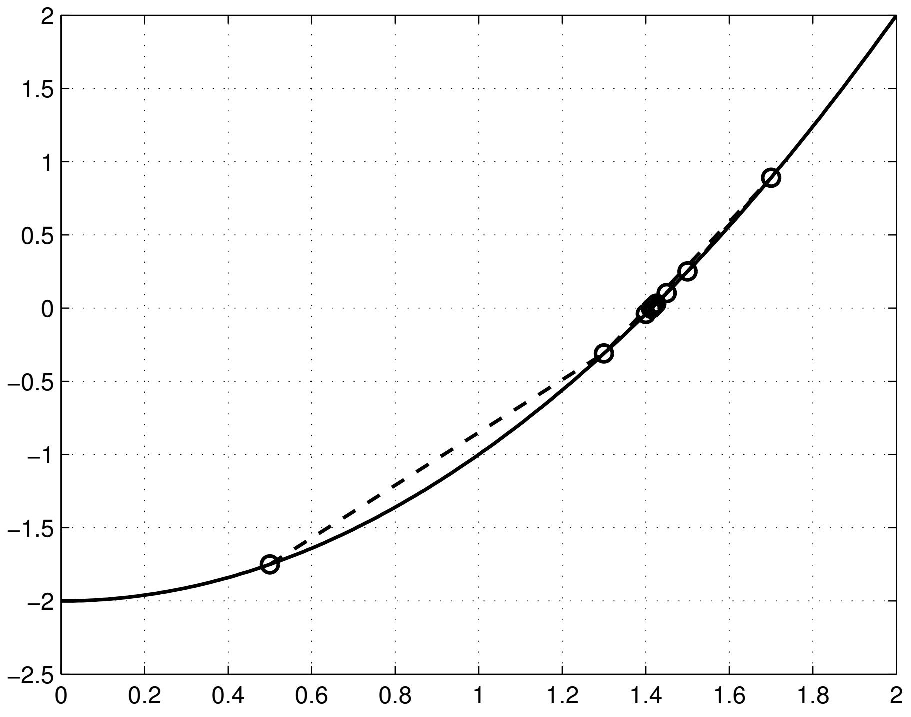


Figure 2: Example of the use of the bisection method. The root of the function $f(x) = x^2 - 2$ lies at approximately $x = 1.4142$. The bisection method needs 21 iterations to locate the root with the required accuracy. The midpoints (c) are plotted in the figure.

2.3 Convergence analysis

Suppose f is a continuous function that takes values of opposite sign at the ends of an interval $[a_0, b_0]$. Then there is a root r in $[a_0, b_0]$, and if we use the midpoint $c = (a_0 + b_0)/2$ as our estimate of r, we have

$$|r - c_0| \leq \frac{b_0 - a_0}{2}$$

Then after n steps, an approximate root will have been computed with error at most

$$\text{error} = |r - c_n| \leq \frac{b_0 - a_0}{2^{n+1}}$$

If an error tolerance ε has been prescribed in advance, it is possible to determine the number of steps required in the bisection method:

$$n > \frac{\log(b_0 - a_0) - \log(2\varepsilon)}{\log 2}$$

For the previous example, we obtain

$$n > \frac{\log(2.1 - (-1.1)) - \log(2 \times 10^{-6})}{\log 2} \approx 20.6$$

Thus we need at least 21 iterations, as was produced in the output of the bisection program.

In general, a sequence $\{x_n\}$ is said to have convergence of the order r to a limit x if it applies that

$$\lim_{n \to \infty} \frac{|x_{n+1} - x|}{|x_n - x|^r} = C$$

where C is some finite constant other than zero.

For the bisection method, we have $r = 1$ (linear convergence) and $C = 1/2$.

3 Newton's method

3.1 Interpretations of Newton's method

The Newton's method, also known as the *Newton-Raphson iteration*, is one of the most important procedures in numerical analysis. Here we apply it to a single equation of the form $f(x) = 0$, but in the more general form it can be applied to find roots of systems of equations.

It is assumed that the function f is continuous and differentiable. This implies that the graph of f has a definite slope at each point and hence a *unique tangent line*. At a given point x_0, the function f is approximated by its tangent:

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

We take the zero of the tangent line ($y = 0$) as the first approximation to the zero of $f(x)$ (see Fig. 3).

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Repeating the process produces the sequence of points known as the *iteration formula* of the Newton's method:

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

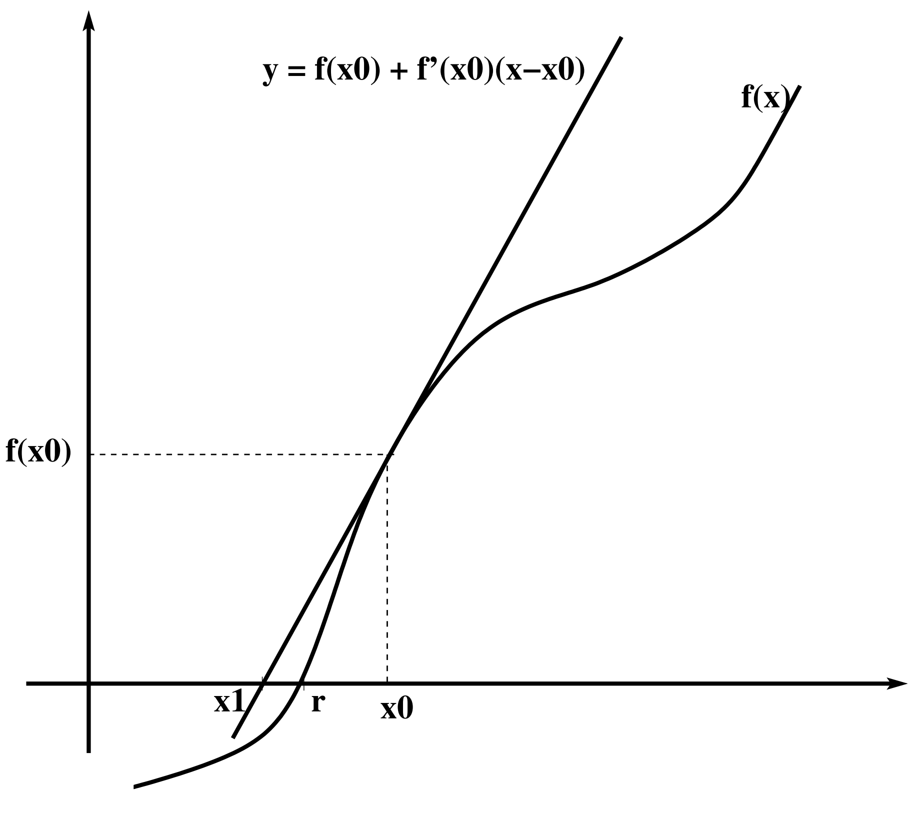


Figure 3: Illustration of Newton's method.

Example

As an illustration of Newton's method, we consider locating the root of $x^3 + x = 2x^2 + 3$. This is written in the form $f(x) = x^3 - 2x^2 + x - 3$ and we attempt to solve $f(x) = 0$. The derivative of f is $f'(x) = 3x^2 - 4x + 1$. For efficiency, these can be written in the *nested form*:

$$
\begin{aligned}
f(x) &= ((x-2)x+1)x-3 \\
f'(x) &= (3x-4)x+1
\end{aligned}
$$

Using C, the core of Newton's method can be written as follows:

```
for(i=1; i<=Nmax; i++) {
  fp = func2(x); /* value of derivative f' at x */

  /* Problem: no convergence ! */
  /* An error message should also be printed here */
  if(fabs(fp) <= delta)
    exit(1);

  d = fx/fp;
  x = x-d;
  fx = func(x);  /* value of f at point x */

  if(fabs(d) <= eps)
    break;
}
```

Output of the program:

```
i           x                     f(x)
0   4.000000000000000000  33.000000000000000000
1   3.000000000000000000   9.000000000000000000
2   2.437500000000000000   2.036865234375000000
3   2.213032716315109560   0.256363385061417537
4   2.175554938721488085   0.006463361488813065
5   2.174560100666445894   0.000004479068049961
6   2.174559410293312567   0.000000000002157175
7   2.174559410292979944  -0.000000000000000845
```

Notice how quickly the Newton's method converges until the maximum precision of the computer is reached. Figure 4 shows a plot of the function f along with the series of approximations x_n.

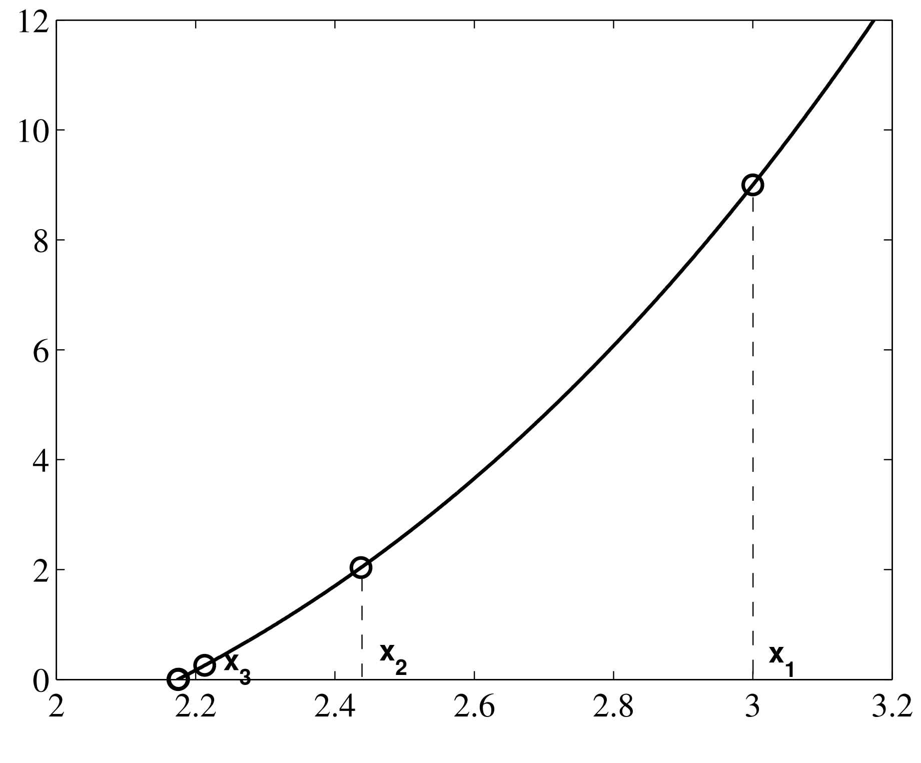


Figure 4: Example of the use of Newton's method. The function f is plotted along with the approximations of the root.

3.2 Convergence analysis

Let f have two continuous derivatives f' and f'', and let r be a simple zero of f ($f'(r) \neq 0$). The Newton's method, if started sufficiently close to r, converges *quadratically* to r, i.e.

$$|r - x_{n+1}| \leq c|r - x_n|^2$$

This can be shown as follows. Taylor expansion to 2nd order of the function around the root r is

$$f(r + \varepsilon_n) \approx f(r) + \varepsilon_n f'(r) + \frac{1}{2}\varepsilon_n^2 f''(r),$$

where ε_n is the error of the n^{th} iteration. Correspondingly for the derivative

$$f'(r + \varepsilon_n) \approx f'(r) + \varepsilon_n f''(r).$$

The iterated values for r are

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

Now, $x_n = r + \varepsilon_n$ and $x_{n+1} = r + \varepsilon_{n+1}$, so

$$r + \varepsilon_{n+1} = r + \varepsilon_n - \frac{f(r + \varepsilon_n)}{f'(r + \varepsilon_n)} \Leftrightarrow \varepsilon_{n+1} \approx \varepsilon_n - \frac{f(r) + \varepsilon_n f'(r) + \frac{1}{2}\varepsilon_n^2 f''(r)}{f'(r)}$$

$f(r) = 0$, so

$$\varepsilon_{n+1} = -\frac{1}{2}\frac{f''(r)}{f'(r)}\varepsilon_n^2$$

This leads us to the **Newton's method theorem**:

If f,f' and f'' are continuous in the neighborhood of a root r of f and if $f'(r) \neq 0$, then there is a positive δ with the following property: If the initial point in Newton's method satisfies $|r - x_0| \leq \delta$, then all subsequent points x_n satisfy the same inequality, converge to r, and do so quadratically; i.e.

$$|r - x_{n+1}| \leq c(\delta)|r - x_n|^2$$

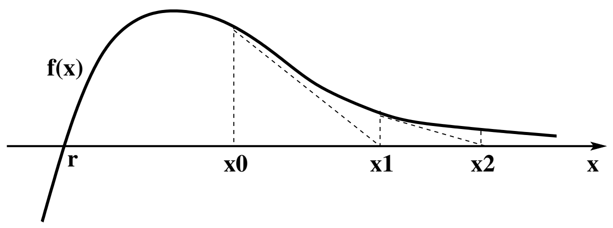
where

$$c(\delta) = \frac{1}{2}\frac{\max_{|x-r|\leq\delta}|f''(x)|}{\min_{|x-r|\leq\delta}|f'(x)|} \quad (\delta > 0)$$

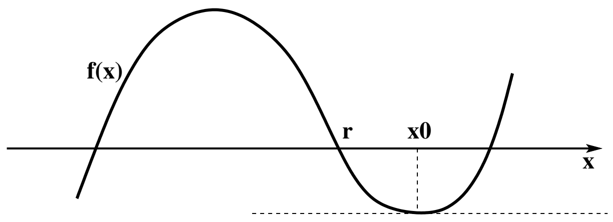
3.3 Starting point

Consideration must be given to a proper choice of a starting point. There are several cases where Newton's method can fail because of bad starting points.

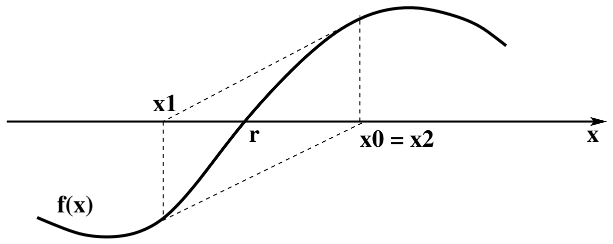
i. **Runaway**
If x_0 is not *sufficiently* close to r, the tangent line at x_0 can intersect the x axis at a point remote from the root r, and successive points in the Newton's iteration *recede* from r instead of converging to it.



ii. **Flat spot**
If the tangent line at x_0 is parallel to the x axis (in a local extremum point), the next approximation is $x_1 = \pm\infty$ and Newton's method fails. Here a combination of the bisection method and Newton's method would produce a successful outcome (see the course exercises).



iii. **Cycle**
It can happen that $x_2 = x_0$, which means that the iteration values cycle. This situation is often encountered if the function f is obtained by table interpolation. In practice, roundoff errors or limited precision of the computer may eventually cause this situation to become unbalanced so that the iteration continues either inward and converge or outward and diverge. With a better initial guess, the method would have succeeded.



Example. Newton's method and fractals.

The Newton's method can be used to create computer-generated displays with fractal patterns. We can take a polynomial in the complex variable z. For example, $p(z) = z^4 - 1$ is suitable. This polynomial has four zeros. Each of these zeros has a basin of attraction: a set of points z such that Newton's iteration, if started at z, will converge to that zero.

Intuitively, you might expect that the four convergence regions would somehow be surrounding the respective roots. But take a look at Fig. 5 which shows the results of a numerical exploration. The four basins of attraction have all been assigned different colors. We notice that the areas are far from simple, in fact, the boundaries are *fractal* (a fractal, or self-similar, structure is one that repeats itself on all scales of magnification).

So how does a fractal pattern emerge from something as simple as Newton's method? The answer lies in the "flat spot" example which shows how a local extremum (on the real line) causes Newton's method to shoot off to infinity. Consider a point that is slightly removed from a point of local extremum (on the complex plane). Then you might not get shot to infinity but - by luck - end up in the basin of attraction of the desired root. This means that in the neighborhood of an extremum, there must be a small "copy" of the basin of attraction (a kind of "bounce-away copy"). Similarly, there can be "two-bounce" and "three-bounce" copies and so on. This creates a fractal pattern.

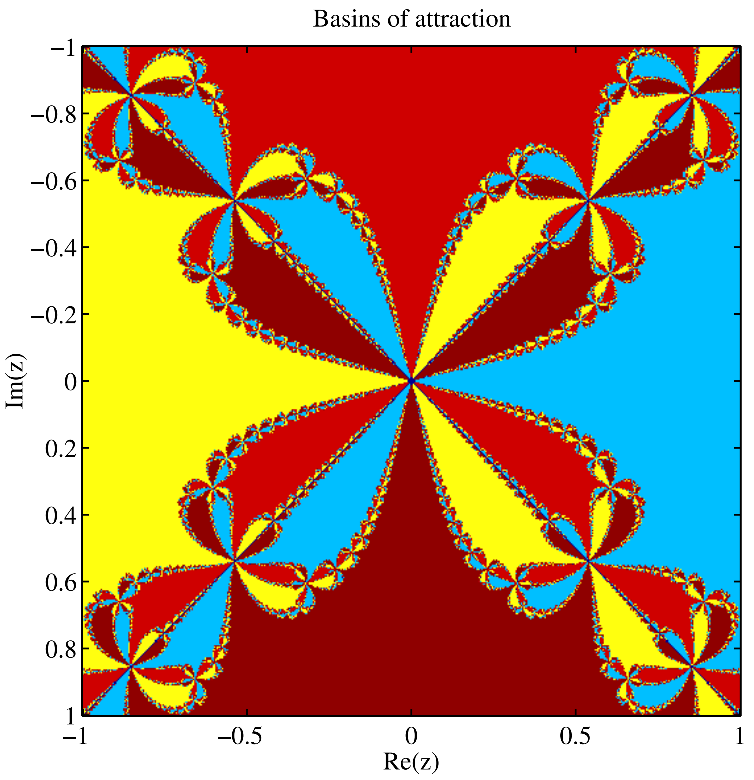


Figure 5: Basins of attraction. The complex polynomial $p(z) = z^4 - 1$ has four roots and each of the roots has an area (basin of attraction) from which Newton's method will converge to that particular root. The areas are not uniform but disjoint. Each color represents one basin of attraction.

3.4 Multiple roots

Consider a root r of function f. If in addition to $f(r) = 0$, it applies that $f'(r) = 0$, then r is called a multiple zero of f. Newton's method for a multiple zero converges only linearly!

The multiplicity of root r is m if it applies that

$$\begin{aligned} &f^{(k)}(r) = 0, \quad 0 \le k < m \\ &f^{(m)}(r) \neq 0 \end{aligned}$$

According to Taylor's theorem, we do a series expansion of f around the root. Noting that all the derivatives below the mth are zero, this leads to

$$f(x) = \frac{(x-r)^m}{m!} f^{(m)}(\xi_x)$$

From this we see that if m is *odd*, the function changes sign at $x = r$. If m is *even*, the function only touches the x axis (see Fig. 6).

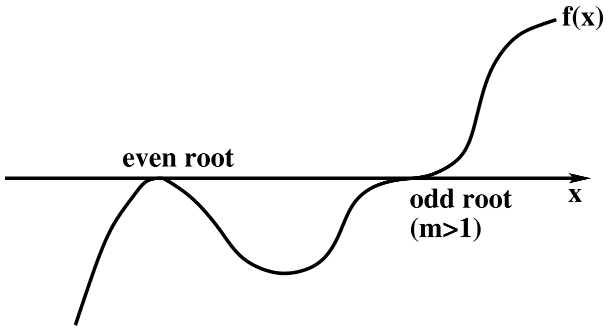


Figure 6: Multiple roots. An odd multiple root changes sign at $x = r$, while an even root only touches the x axis.

It can also happen that due to the finite precision of the floating-point numbers in a computer, two closely situated roots may be interpreted as a single multiple root.

3.5 Precision in finding multiple roots

Locating a multiple root is more difficult than finding a single root. In the vicinity of a multiple root, the function f is almost parallel to the x axis, which means that even a very small error in the function value can cause a large error in the location of the root.

Example. Consider the following functions (shown in Fig. 7):

$$f_1(x) = (x-1)^3$$
$$f_2(x) = x^3 - x^2 + x - 1$$

They both have roots at $x = 1$.

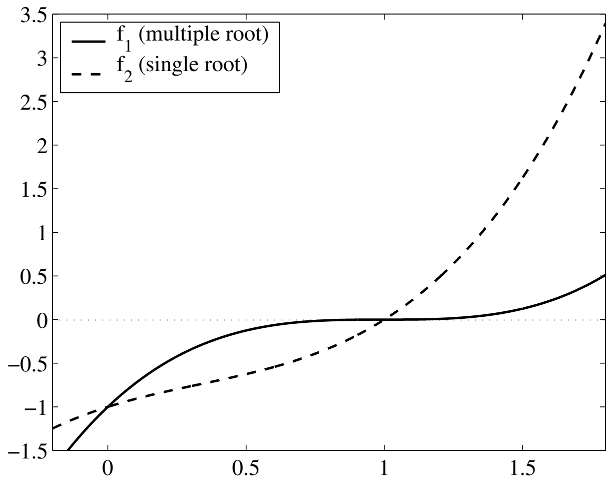


Figure 7: Two functions with roots at $x = 1$.

Figure 8 shows the absolute value of the error in the location of the root. We see that the convergence is much slower for the function f_1 which has the multiple root at $x = 1$.

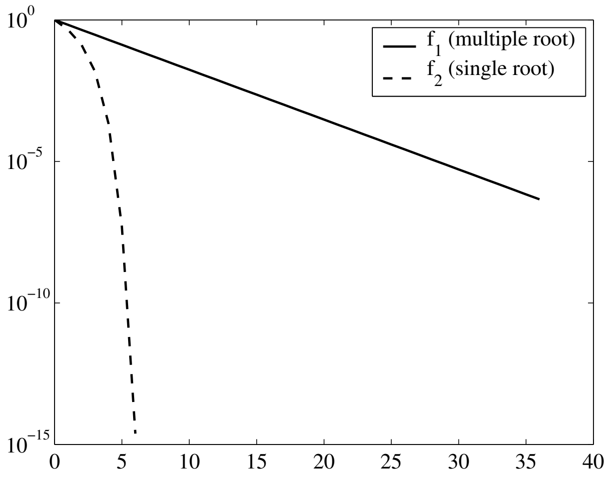


Figure 8: The absolute error in the location of the root using Newton's method.

3.6 Modified Newton's method: version 1

If we somehow knew in advance what the multiplicity of the zero is, the Newton's method could be accelerated by modifying the iteration formula to read

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

Example 1.

If we apply this to the case of the function $f_1 = (x-1)^3$, for which we easily see that $m = 3$, we obtain the correct answer $r = 1$ in just one iteration!

The reason is that in this case we get lucky. The derivative of $f_1(x)$ is given by

$$f_1'(x) = 3x^2 - 6x + 3 = 3(x-1)^2$$

Consequently the iteration formula in the modified Newton's method is written as

$$\begin{aligned} x_{n+1} &= x_n - m\frac{f(x_n)}{f'(x_n)} \\ &= x_n - 3\frac{(x_n-1)^3}{3(x_n-1)^2} = x_n - (x_n - 1) = 1 = r \end{aligned}$$

Thus we obtain the solution in one step regardless of the starting point x_0!! (This is a coincidence).

Example 2.

Consider the following function (shown in Fig. 9):

$$f(x) = xe^{-x} - e^{-1}$$

The zero of this function is $r = 1$. The multiplicity of the root is $m = 2$, which can be seen by differentiating the function twice:

$$\begin{aligned} f'(x) &= e^{-x}(1-x) \qquad & f'(1) &= e^{-1}(1-1) = 0 \\ f''(x) &= e^{-x}(x-2) \qquad & f''(1) &= e^{-1}(1-2) = -e^{-1} \neq 0 \end{aligned}$$

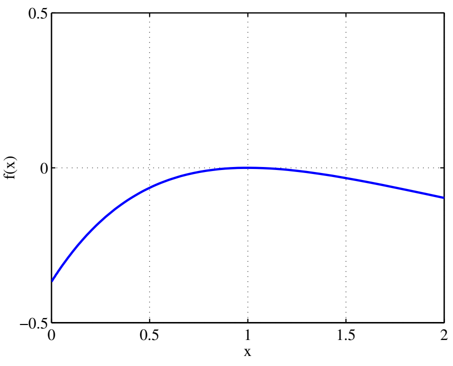


Figure 9: The function f of Example 2.

From Fig. 10 we see that using the regular Newton's method with tolerance 10^{-12}, the method does not converge in 100 iterations. The error fluctuates around 10^{-8}.

The modified scheme improves the speed of convergence initially (we obtain the accuracy of 10^{-8} faster), but the method suffers from the same oscillating values around the solution.

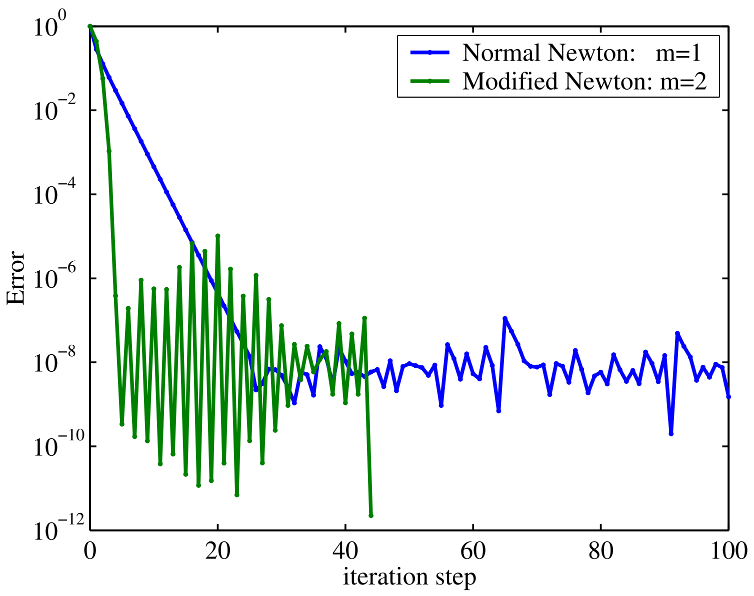


Figure 10: Comparison of the absolute error in locating the root of function f using the regular and modified Newton's methods.

3.7 Modified Newton's method: version 2

Let us denote

$$u(x) = \frac{f(x)}{f'(x)}$$

The idea is to replace $f(x)$ by $u(x)$ in the Newton's method. Thus the iteration formula is now

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

This approach can also be applied to the bisection method, which in the original form cannot be used at all for locating even roots. (Why?)

Example 3.

For the function

$$f(x) = xe^{-x} - e^{-1}$$

we get (see Fig. 11)

$$u(x) = \frac{xe^{-x} - e^{-1}}{e^{-x}(1-x)} \qquad u'(x) = \frac{1-e^{x-1}}{1-x} + \frac{x-e^{x-1}}{(1-x)^2}$$

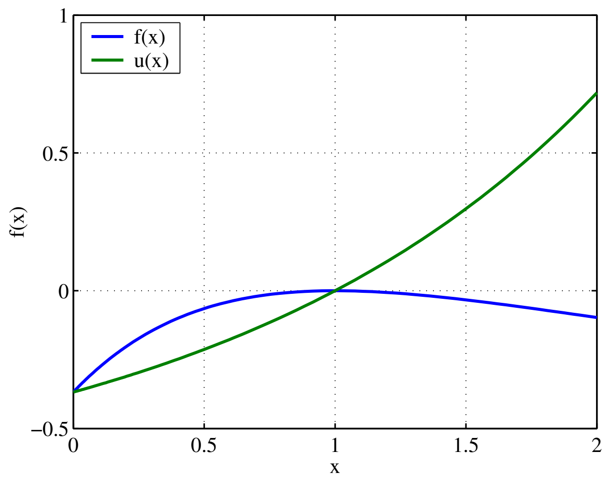


Figure 11: Functions f and u.

Figure 12 shows the error in the location of the root as a function of iteration step for three cases:

1. Normal Newton's method - single root
2. Modified Newton's method 1 - multiple root
3. Modified Newton's method 2 - multiple root

The second modified method reduces the oscillations, but the precision is still of the order 10^{-8}.

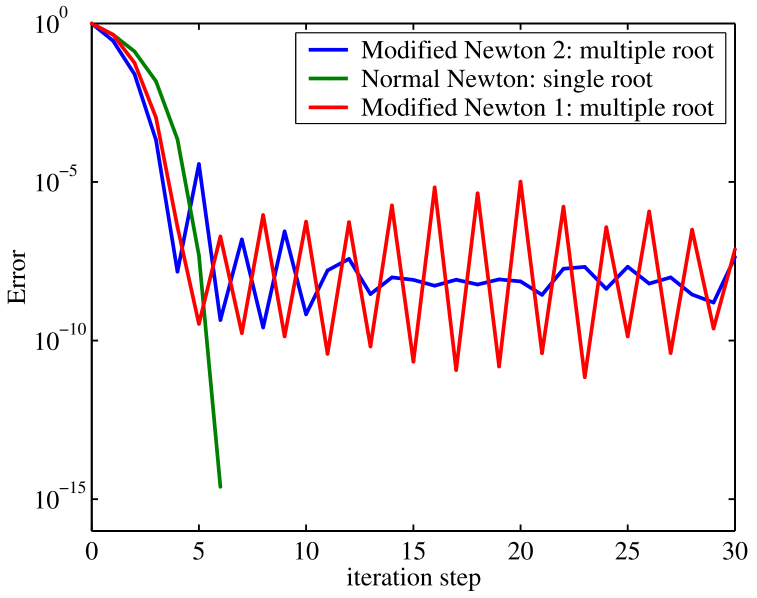


Figure 12: Comparison of the absolute error in locating the root of function f using the regular and two modified Newton's methods.

3.8 Modified bisection method

In the modified bisection method, we substitute the function

$$u(x) = \frac{f(x)}{f'(x)}$$

in the place of $f(x)$ and apply the algorithm as usual. Notice that problems arise if we select the initial values of a and b such that $c = (a+b)/2 = r$ due to a singularity of $u(x)$ at $x = r$.

Figure 13 shows a comparison of the absolute error for the modified bisection method and the modified Newton's method. The bisection method converges more slowly, but the accuracy is somewhat better.

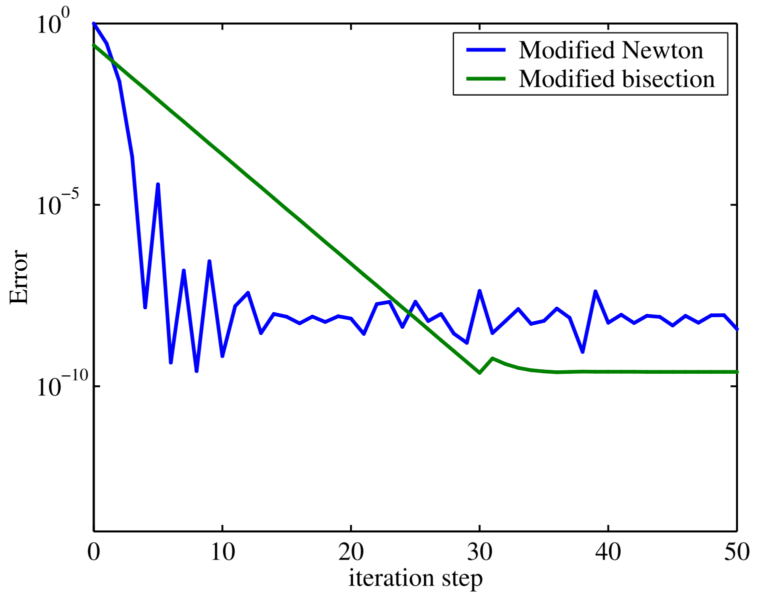


Figure 13: Comparison of the absolute error in locating the root of function f using the modified Newton's method and the modified bisection method.

3.9 Systems of nonlinear equations

Solving systems of nonlinear equations is considerably more difficult than solving a single equation. In a general case, a system of n nonlinear equations in n unknowns x_i can be displayed in the form

$$\begin{cases} f_1(x_1,x_2,\dots,x_n) & = 0 \\ f_2(x_1,x_2,\dots,x_n) & = 0 \\ & \vdots \\ f_n(x_1,x_2,\dots,x_n) & = 0 \end{cases}$$

In vector notation

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}$$

where

$$\mathbf{f} = (f_1, f_2, \dots, f_n)^T$$
$$\mathbf{x} = (x_1, x_2, \dots, x_n)^T$$

The extension of Newton's method for systems of nonlinear equations is

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - [\mathbf{f}'(\mathbf{x}^{(k)})]^{-1}\mathbf{f}(\mathbf{x}^{(k)})$$

where $\mathbf{f}'(\mathbf{x}^{(k)})$ is the *Jacobian matrix* which consist of partial derivatives $\frac{\partial f_i}{\partial x_j}$.
In practice, the computational form of Newton's method does not involve inverting the Jacobian matrix $\mathbf{J}$ but rather solves the Jacobian linear systems

$$[\mathrm{J}(\mathbf{x}^{(k)})]\mathbf{h}^{(k)} = -\mathbf{f}(\mathbf{x}^{(k)})$$

The next iteration of Newton's method is then

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{h}^{(k)}$$

This is called *Newton's method for nonlinear systems*. The Jacobian linear system can be solved by e.g., Gaussian elimination (to be discussed later during the course). Given the difficulties related to solving large systems of nonlinear equations, we will not go into this subject in more detail.

4 Secant method

4.1 Description of the method

We will now consider another procedure that converges almost as quickly as Newton's method but avoids the calculation of derivatives.

In the secant method, we replace $f'(x)$ by the difference quotient

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

In particular, if $x = x_n$ and $h = x_{n-1} - x_n$, we have

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

This expression is used in the place of the derivative in the Newton's formula, and thus the *iteration formula* for the secant method is

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

Figure 14 illustrates how the method works. Each successive approximation of the zero of f is given by the intersection of the secant line and the x axis.

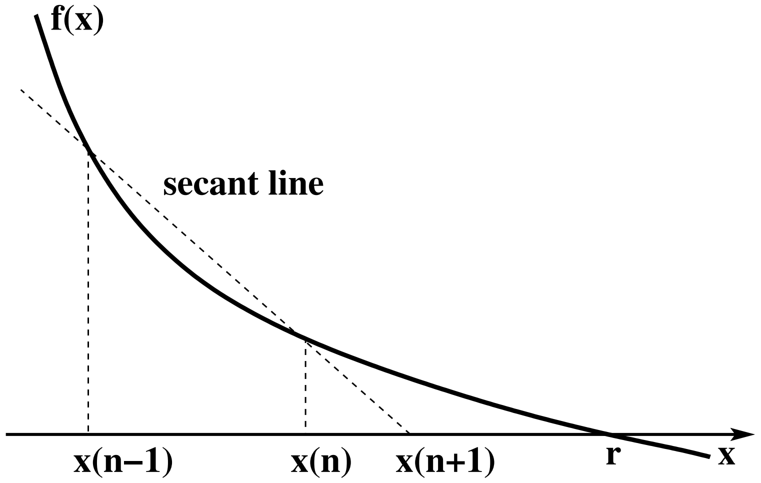


Figure 14: Illustration of the secant method.

4.2 Secant algorithm

In order to start, two points (x_0 and x_1) must be provided. The iteration formula can then be used to generate successive points $x_2, x_3, x_4, \ldots$.

Care must be taken in checking the quantity $f(x_n) - f(x_{n-1})$. If it is nearly zero, overflow may occur. It is also advisable to halt the iteration when $|f(x_n) - f(x_{n-1})| \leq \delta$ for some specified tolerance δ, e.g. $\frac{1}{2} \times 10^{-6}$.

The core of the secant algorithm is given in the following C implementation:

```
/* Initial points */
fa = func(a);
fb = func(b);

/* Set f(a) < f(b) */
if(fabs(fa) > fabs(fb)) {
  temp = fa; fa = fb; fb = temp;
  temp = a; a = b; b = temp;
}

/* Set max number of steps and tolerance */
Nmax = 100;
eps = 0.5*pow(10.0,-6.0);

/* Iteration */
for(i=2; i<=Nmax; i++) {

  /* Set f(a) < f(b) */
  if(fabs(fa) > fabs(fb)) {
    temp = fa; fa = fb; fb = temp;
    temp = a; a = b; b = temp;
  }

  /* Difference quotient */
  d = (b-a)/(fb-fa);

  /* Begin iteration */
  b = a;
  fb = fa;
  d = d*fa;

  /* Check for convergence */
  if(fabs(d) <= eps) break;

  /* New point */
  a = a-d;
  /* New function value */
  fa = func(a);
}
```

Example. When the secant algorithm is applied to the following functions

$$f_1(x) = (x-1)^3 \qquad \text{multiple roots}$$
$$f_2(x) = x^3 - x^2 + x - 1 \qquad \text{single root}$$

we get the results shown in Fig. 15. Again, convergence is much faster in the single-root case.

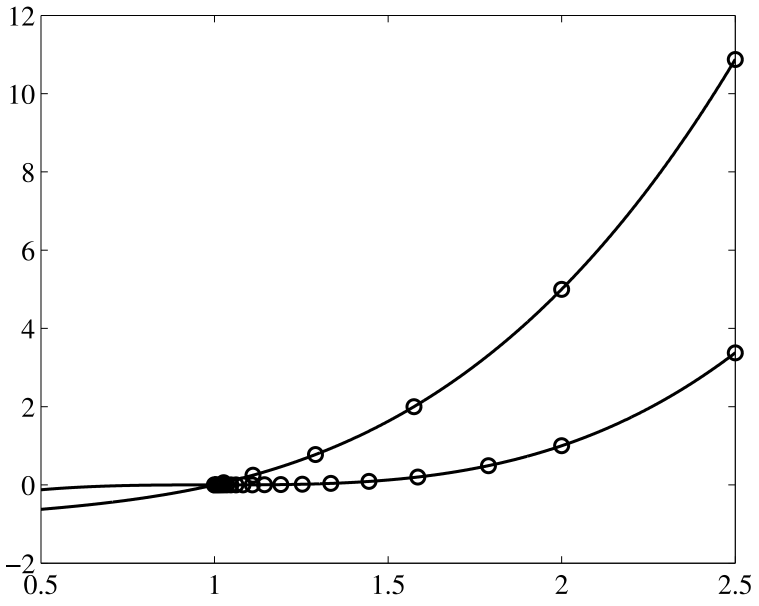


Figure 15: Convergence of the secant method in a single-root and a multiple-root case.

4.3 Convergence

The order of convergence for the secant method can be expressed in terms of of the inequality

$$|e_{n+1}| \leq C|e_n|^\alpha$$

where $\alpha = \frac{1}{2}(1+\sqrt{(}5) \approx 1.62$. Since $\alpha > 1$, we say that the convergence is *superlinear*.

4.4 Comparison of methods

Applied to locating a single root of a continuous and differentiable function f, Newton's method gives fastest convergence (quadratic). The convergence of the secant method is superlinear and the bisection method converges linearly. Figure 16 shows the measured absolute error from an example case with a single root.

Using all of these methods, suitable initial values must be provided. In difficult problems, a combination of two methods (e.g. bisection and newton or bisection and secant) can be a working approach.

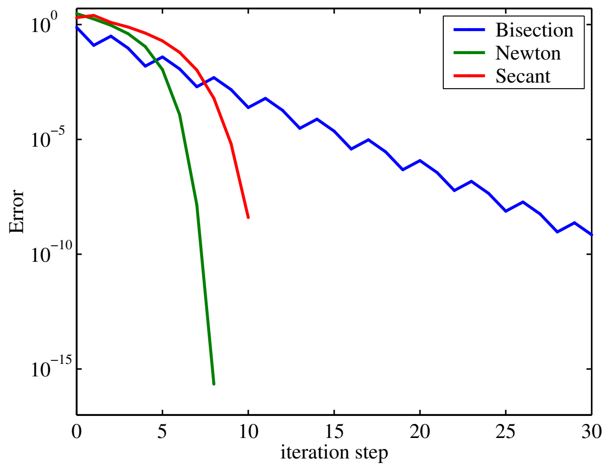


Figure 16: Comparison of convergence speed obtained using Newton's method, the secant method and the bisection method.

The stepped shape of the error in the bisection method is explained by the sequence of x_n vs $f(x_n)$. The value of x_n shifts from one side of the root to the other (see Fig. 17).

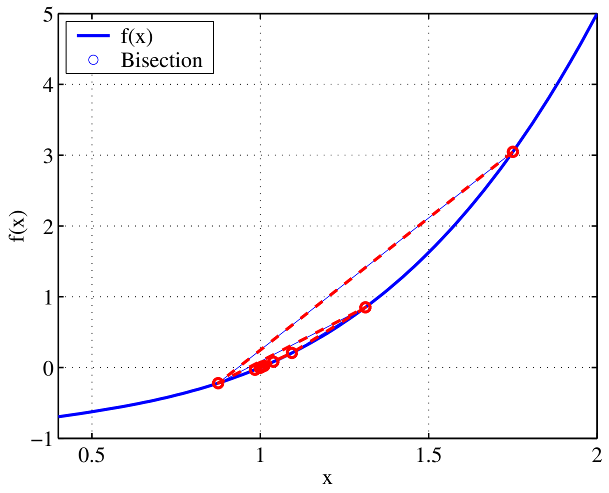


Figure 17: In the bisection method, the value of x_n shifts from one side of the root to the other, which results in steps in the absolute error of the estimate.