Numerical Method(Math-2073/53) Lecture note: Chapter-II

"Mathematics is the most beautiful and most powerful creation of the human spirit."

Stefan Banac

Dejen Ketema
Department of Mathematics
dejen.ketema@amu.edu.et

March, 2019

Contents

Chapter 1

Solution of Nonlinear Equations

In this chapter we shall discuss one of the oldest approximation problems which consists of finding the roots of an equation. It is also one of the most commonly occurring problems in applied mathematics. The **root-finding problem** consists of the following: given a continuous function f, find the values of x that satisfy the equation

$$f(x) = 0. \tag{1.1}$$

The solutions of this equation are called the **zeros** of f or the **roots** of the equation. In general, Eqn. (1.1) is impossible to solve exactly. Therefore, one must rely on some numerical methods for an approximate solution. The methods we will discuss in this chapter are iterative and consist basically of two types: one in which the convergence is guaranteed and the other in which the convergence depends on the initial guess.

Example 1.1: Floating Sphere

Consider a sphere of solid material floating in water. Archimedes' principle states that the buoyancy force is equal to the weight of the replaced liquid. Let $V_s = (\frac{4}{3})\pi r^3$ *be the volume of the sphere, and let* V_w *be the volume of water it displaces when it is partially submerged. In static equilibrium, the weight of the sphere is balanced by the buoyancy force*

$$\rho_s g V_s = \rho_w g V_w$$

where ρ_s *is the density of the sphere material,* g *is the acceleration due to gravity,* ρ_w*, is the density of water. The volume* V_h *of water displaced when a sphere is submerged to a depth* h *is (see Figure 1.1)*

$$V_h = \frac{\pi}{3} h^2 (3r - h).$$

Applying Archimedes' principle produces the following equation in term of h

$$h^3 - 3rh^2 + 4\rho r^3 = 0.$$

Given values of r *and the specific gravity of the sphere material* $\rho = \frac{\rho_s}{\rho_w}$*, the solutions* h *of the above equation can be obtained by one of the iterative methods described in this chapter.*

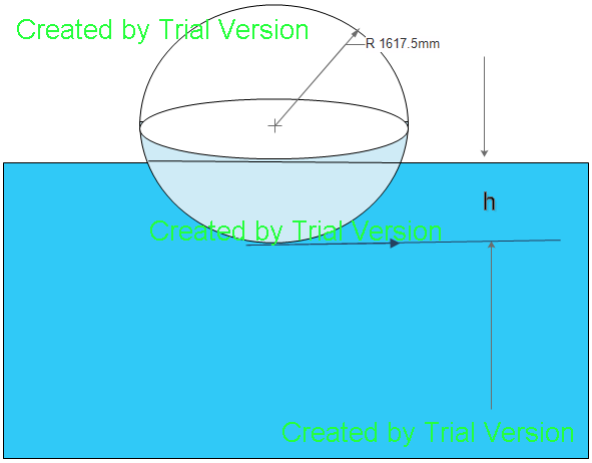

Figure 1.1: Sphere

Example 1.2: Area of a Segment

A segment of circle is the region enclosed by an arc and its chord (see Figure 1.2). If r is the radius of the circle and θ the angle subtended at the center of the circle, then it can be shown that the area A of the segment is

$$A = \frac{1}{2}r^2(\theta - \sin\theta)$$

where θ is in radian. Given A and r one can determine the value of θ by finding the zeros of $f(x) = \frac{1}{2}r^2(\theta - \sin\theta) - A$.

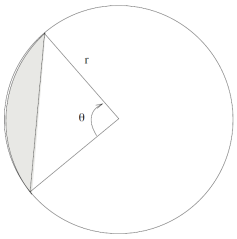

Figure 1.2: Area of a segment.

Definition 1.1: A Zero of function $f(x)$

We now consider one of the most basic problems of numerical approximation, namely the root-finding problem. This process involves finding a root, or solution, of an equation of the form $f(x) = 0$ for a given function f. A root of this equation is also called a zero of the function f.

The process of solving an equation numerically is different from the procedure used to find an analytical solution. An analytical solution is obtained by deriving an expression that has an exact numerical value. A numerical solution is obtained in a process that starts by finding an approximate solution and is followed by a numerical procedure in which a better (more accurate) solution is determined. Since numerical solutions are not exact, some criterion has to be applied in order to determine whether an estimated solution is accurate enough. Several measures can be used to estimate the accuracy of an approximate solution. The decision as to

which measure to use depends on the application and has to be made by the person solving the equation.

The methods used for solving equations numerically can be divided into two groups: **bracketing methods** and **open methods**.

Stopping Criteria

Many ways to decide when to stop:
Let c^* be the true (exact) solution such that $f(c^*) = 0$, and let m_k be a numerically approximated solution such that $f(m_k) = \epsilon$ (where ϵ is a small number). **Tolerance in the solution** A tolerance is the maximum amount by which the true solution can deviate from an approximate numerical solution.

When do we stop?

Let m_k is approximate solution of $f(x)$ at k^{th} iteration.
We can

1. keep going until successive iterates are close:
$$|m_k - m_{k-1}| < \epsilon$$
2. close in relative terms
$$\frac{|m_k - m_{k-1}|}{|m_k|} < \epsilon$$
3. the function value is small enough
$$|f(m_k)| < \epsilon$$

No choice is perfect. In general, where no additional information about f is known, the second criterion is the preferred one (since it comes the closest to testing the relative error).

Rate of Convergence

Suppose an algorithm generates a sequence of approximations, c_n, which approaches a limit, c_*, or
$$\lim_{n\to\infty} c_n = c_*$$
How quickly does $c_n \to c_*$?

Definition 1.2: Rate of Convergence

If a sequence $c_1, c_2, \cdots, c_n$ converges to a value c_* and if there exist real numbers $\lambda > 0$ and $\alpha \geq 1$ such that
$$\lim_{n\to\infty} \frac{|c_{n+1} - c_*|}{|c_n - c_*|^\alpha} = \lambda$$
*then we say that α is the rate of **convergence of the sequence**.*

1.1 Bisection Method

1.1.1 Idea

Theorem 1.1: The intermediate value theorem

The intermediate value theorem states that if a continuous function, f, with an interval, [a, b], as its domain, takes values f(a) and f(b) at each end of the interval, then it also takes any value between f(a) and f(b) at some point within the interval.

The theorem of existence of roots for continuous function (or Bolzano's theorem) states.

Theorem 1.2: Bolzano's theorem

If a continuous function has values of opposite sign inside an interval, then it has a root in that interval. Let $f : [a, b] \to \mathbb{R}$ be a continuous function such that $f(a) \cdot f(b) < 0$.Then there exist at least one point x in the open interval (a, b) such that $f(x) = 0$.

The **Bisection**, or **Binary-search, method** is a bracketing method for finding a numerical solution of an equation of the form $f(x) = 0$. Let $f(x)$ be a given function, continuous on an interval $[a, b]$, such that

$$f(a)f(b) < 0. \tag{1.2}$$

Using the Intermediate Value Theorem, it follows that there exists at least one zero of f in (a, b). To simplify our discussion, we assume that f has exactly one root c. Such a function is shown in Figure 1.3. The bisection method is based on halving the interval $[a, b]$ to determine

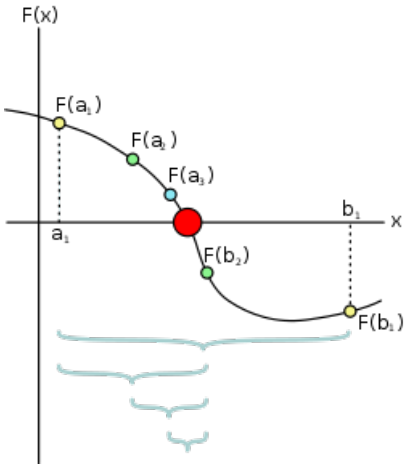

Figure 1.3: The bisection method and the first two approximations to its zero c.

a smaller and smaller interval within which c must lie. The procedure is carried out by first defining the midpoint of $a, b]$, $c = (a + b)/2$ and then computing the product $f(c)f(b)$. If the product is negative, then the root is in the interval $[c, b]$. If the product is positive, then the root is in the interval $[a, c]$. Thus, a new interval containing c is obtained. The process of halving the new interval continues until the root is located as accurately as desired, that is

$$|a_n - b_n| < \epsilon \tag{1.3}$$

where an and b_n are the endpoints of the nth interval $[a_n, b_n]$ and ϵ is a specified tolerance value. Some other stopping criteria that one can use, other than (1.3), are given by

$$\frac{|a_n - b_n|}{|a_n|} < \epsilon \tag{1.4}$$

or

$$|f(a_n)| < \epsilon. \tag{1.5}$$

1.1.2 Iteration tasks

The input for the method is a continuous function f, an interval $[a_0, b_0]$, and the function values $f(a_0)$ and $f(b_0)$. The function values are of opposite sign (there is at least one zero crossing within the interval).

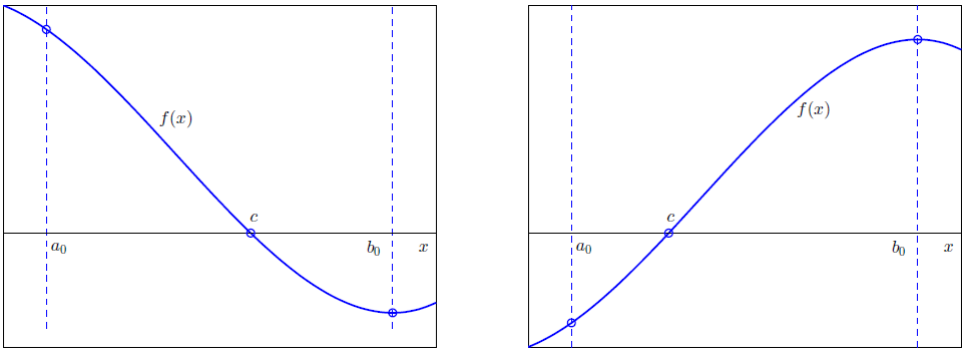

Figure 1.4: Bisection Method

Algorithm 1.1: The bisection method procedure is:

1. *Choose a starting interval* $[a_0, b_0]$ *such that* $f(a_0)f(b_0) < 0$.
2. *Compute* $f(m_0)$ *where* $m_0 = (a_0 + b_0)/2$ *is the midpoint.*
3. *Determine the next subinterval* $[a_1, b_1]$ *:*
   (a) *If* $f(a_0)f(m_0) < 0$, *then let* $[a_1, b_1]$ *be the next interval with* $a_1 = a_0$ *and* $b_1 = m_0$.
   (b) *If* $f(b_0)f(m_0) < 0$, *then let* $[a_1, b_1]$ *be the next interval with* $a_1 = m_0$ *and* $b_1 = b_0$.
4. *Repeat (2) and (3) until the interval* $[a_N, b_N]$ *reaches some predetermined length.*
5. *Constructs a sequence of intervals containing the root c:*

$$(a_0, b_0) \supset (a_1, b_1) \supset \cdots \supset (a_{N-1}, b_{N-1}) \supset (a_N, b_N) \ni c$$

6. *After k steps*

$$|b_k - a_k| = \frac{1}{2}|b_{k-1} - a_{k-1}| = \left(\frac{1}{2}\right)^k |b_0 - a_0|$$

7. *At step k, the midpoint* $m_k = \frac{a_k + b_k}{2}$ *is an estimate for the root c with*

$$m_k - d_k \le c \le m_k + d_k, \quad d_k = \left(\frac{1}{2}\right)^{k+1} |b_0 - a_0|$$

.

A solution of the equation $f(x) = 0$ in the interval $[a, b]$ is guaranteed by the Intermediate Value Theorem provided $f(x)$ is continuous on $[a, b]$ and $f(a)f(b) < 0$. In other words, the

function changes sign over the interval and therefore must equal 0 at some point in the interval $[a, b]$.

1.1.3 Absolute Error

The bisection method does not (in general) produce an exact solution of an equation $f(x) = 0$. However, we can give an estimate of the absolute error in the approxiation.

Theorem 1.3: L

t $f(x)$ be a continuous function on $[a, b]$ such that $f(a)f(b) < 0$. After N iterations of the biection method, let x_N be the midpoint in the N^{th} subinterval $[a_N, b_N]$

$$x_N = \frac{a_N + b_N}{2}$$

There exists an exact solution x_{true} of the equation $f(x) = 0$ in the subinterval $[a_N, b_N]$ and the absolute error is

$$| \, x_{\text{true}} - x_N \, | \leq \frac{b-a}{2^{N+1}}$$

Note that we can rearrange the error bound to see the minimum number of iterations required to guarantee absolute error less than a prescribed ϵ :

$$\begin{aligned} \frac{b-a}{2^{N+1}} &< \epsilon \\ \frac{b-a}{\epsilon} &< 2^{N+1} \\ \ln\left(\frac{b-a}{\epsilon}\right) &< (N+1)\ln(2) \\ \frac{\ln\left(\frac{b-a}{\epsilon}\right)}{\ln(2)} - 1 &< N \end{aligned}$$

The convergence of bisection method is slow. At each step we gain one binary digit in accuracy. Since $10^{-1} \approx 2^{-3.3}$, it takes on average 3.3 iterations to gain one decimal digit of accuracy.
Note: The rate of convergence is completely independent of the function f.
We are only using **the sign of** f at the endpoints of the interval(s) to make decisions on how to update. By making more effective use of the values of f we can attain significantly faster convergence.
Form the **sequence of midpoints** with $c_n = m_n$, then from the worst case scenario

$$|c_n - c_*| \leq \left(\frac{1}{2}\right)^{n+1} |b_0 - a_0| \quad \text{or} \quad \frac{|c_{n+1} - c_*|}{|c_n - c_*|} \approx \frac{1}{2}$$

It follows that for Bisection Method $\alpha = 1$, so the **rate of convergence is linear**

1.1.4 Implementation

Write a function called *bisection* which takes 4 input parameters f, a, b and N and returns the approximation of a solution of $f(x)$ given by N iterations of the bisection method. If $f(a_n)f(b_n) \geq 0$ at any point in the iteration (caused either by a bad initial interval or rounding error in computations), then print "*Bisection method fails.*" and return *None*.
Python: Bisection

```
def bisection(f,a,b,N):
    if f(a)*f(b) >= 0:
        print("Bisection method fails.")
        return None
    a_n = a
    b_n = b
    for n in range(1,N+1):
        m_n = (a_n + b_n)/2
        f_m_n = f(m_n)
        if f(a_n)*f_m_n < 0:
            a_n = a_n
            b_n = m_n
        elif f(b_n)*f_m_n < 0:
            a_n = m_n
            b_n = b_n
        elif f_m_n == 0:
            print("Found exact solution.")
            return m_n
        else:
            print("Bisection method fails.")
            return None
    return (a_n + b_n)/2
```

MATLAB Bisection.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function c=bisection(f,a,b,tol)
while 1
   c=(a+b)/2;
   if c-a<tol, break; end
   if f(a)*f(c)>0
     a=c;
   else
     b=c;
   end
end
```


Example 1.3: Golden Ratio

Let's use our function with input parameters $f(x) = x^2 - x - 1$ *and* $N = 25$ *iterations on* $[1, 2]$ *to approximate the* ***golden ratio***

$$\phi = \frac{1 + \sqrt{5}}{2}$$

The golden ratio is a root of the quadratic polynomial $x^2 - x - 1 = 0$.

```
f = lambda x: x**2 - x - 1
approx_phi = bisection(f,1,2,25)
print(approx_phi)
1.618033990263939
```

The absolute error is guaranteed to be less than $(2-1)/(2^{26})$ *which is*

```
error_bound = 2**(-26)
print(error_bound)
1.4901161193847656e-08
```

Let's verify the absolute error is then than this error bound:

```
abs( (1 + 5**0.5)/2 - approx_phi) < error_bound
True
```

Example 1.4

The bisection method applied to

$$f(x) = \left(\frac{x}{2}\right)^2 - \sin(x) = 0$$

with $(a_0, b_0) = (1.5, 2.0)$, *and* $(f(a_0), f(b_0)) = (-0.4350, 0.0907)$ *gives:*

Solution

k	a_k	b_k	m_k	$f(m_k)$
0	1.5000	2.0000	1.7500	-0.2184
1	1.7500	2.0000	1.8750	-0.0752
2	1.8750	2.0000	1.9375	0.0050
3	1.8750	1.9375	1.9062	-0.0358
4	1.9062	1.9375	1.9219	-0.0156
5	1.9219	1.9375	1.9297	-0.0054
6	1.9297	1.9375	1.9336	-0.0002
7	1.9336	1.9375	1.9355	0.0024
8	1.9336	1.9355	1.9346	0.0011
9	1.9336	1.9346	1.9341	0.0004

Example 1.5

Finding the root of a polynomial. Suppose that the bisection method is used to find a root of the polynomial

$$f(x) = x^3 - x - 2.$$

Solution

First, two numbers a and b have to be found such that $f(a)$ and $f(b)$ have opposite signs. For the above function, $a = 1$ and $b = 2$ satisfy this criterion, as

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

and

$$f(2) = (2)^3 - (2) - 2 = 4.$$

Because the function is continuous, there must be a root within the interval $[1, 2]$.

In the first iteration, the end points of the interval which brackets the root are $a_1 = 1$ and $b_1 = 2$, so the midpoint is

$$c_1 = \frac{2+1}{2} = 1.5$$

The function value at the midpoint is $f(c_1) = (1.5)^3 - (1.5) - 2 = -0.125$. Because $f(c_1)$ is negative, $a = 1$ is replaced with $a = 1.5$ for the next iteration to ensure that $f(a)$ and $f(b)$ have opposite signs. As this continues, the interval between a and b will become increasingly smaller, converging on the root of the function. See this happen in the table below.

Iteration	a_n	b_n	c_n	$f(c_n)$
1	*1*	*2*	*1.5*	*-0.125*
2	*1.5*	*2*	*1.75*	*1.6093750*
3	*1.5*	*1.75*	*1.625*	*0.6660156*
4	*1.5*	*1.625*	*1.5625*	*0.2521973*
5	*1.5*	*1.5625*	*1.5312500*	*0.0591125*
6	*1.5*	*1.5312500*	*1.5156250*	*-0.0340538*
7	*1.5156250*	*1.5312500*	*1.5234375*	*0.0122504*
8	*1.5156250*	*1.5234375*	*1.5195313*	*-0.0109712*
9	*1.5195313*	*1.5234375*	*1.5214844*	*0.0006222*
10	*1.5195313*	*1.5214844*	*1.5205078*	*-0.0051789*
11	*1.5205078*	*1.5214844*	*1.5209961*	*-0.0022794*
12	*1.5209961*	*1.5214844*	*1.5212402*	*-0.0008289*
13	*1.5212402*	*1.5214844*	*1.5213623*	*-0.0001034*
14	*1.5213623*	*1.5214844*	*1.5214233*	*0.0002594*
15	*1.5213623*	*1.5214233*	*1.5213928*	*0.0000780*

After 13 iterations, it becomes apparent that there is a convergence to about 1.521: a root for the polynomial.

Advantage and disadvantage

- The method is guaranteed to converge
- The error bound decreases by half with each iteration
- The bisection method converges very slowly
- The bisection method cannot detect multiple roots

Exercise 1.1

Find an approximation of $\sqrt{3}$ correct to within 10^{-4} by using the bisection method on $f(x) = x^2 - 3$ starting on $[1, 2]$.

Exercise 1.2

You are working for 'DOWN THE TOILET COMPANY' that makes floats for ABC commodes. The floating ball has a specific gravity of 0.6 and has a radius of 5.5 cm. You are asked to find the depth to which the ball is submerged when floating in water. The equation that gives the depth x to which the ball is submerged under water is given by

$$x^3 - 0.165x^2 + 3.993 \times 10^{-10}$$

Use the bisection method of finding roots of equations to find the depth x to which the ball is submerged under water. Conduct three iterations to estimate the root of the above equation. Find the absolute relative approximate error at the end of each iteration, and the number of significant digits at least correct at the end of each iteration.

1.2 False position (Regula Falsi) Method

The false position method retains the main features of the Bisection method, that the root is trapped in a sequence of intervals of decreasing size. This method uses the point where the secant lines intersect the x-axis. The secant line over the interval $[a, b]$ is the chord between $(a, f(a))$ and$(b, f(b))$. The two right angles in the figure are similar, which mean that

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

This implies that

Formula

$$c = \frac{af(b) - bf(a)}{f(b) - f(a)} = b - f(b)\frac{(b-a)}{f(b) - f(a)} \tag{1.6}$$

then we can compute $f(c)$ and repeat the process with the interval $[a, c]$, if $f(a) \times f(c) < 0$ or to the interval $[c, b]$, if and only if $f(c) \times f(b) < 0$.

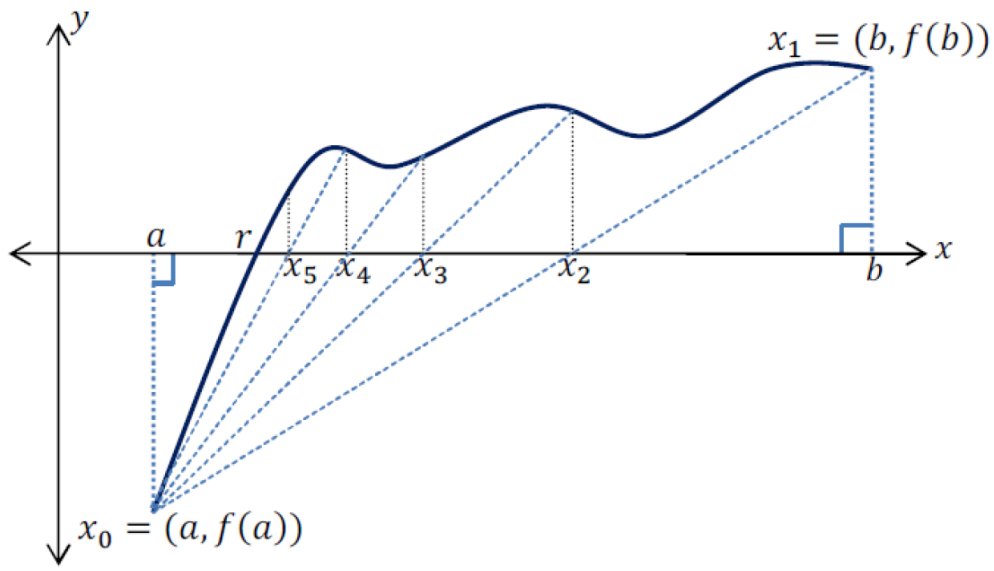

Figure 1.5: Root finding using Regula-Falsi method

Algorithm 1.2: Regula-Falsi Method: Given a continuous function $f(x)$

1. *Choose the first interval by finding points a and b such that a solution exists between them and $(a < b)$. This means that $f(a)$ and $f(b)$ have different signs such that $f(a)f(b) < 0$. The points can be determined by looking at a plot of $f(x)$ versus x.*
2. *Calculate the first estimate of the numerical solution c by using Eq. (1.6).*
3. *Determine whether the actual solution is between a and c or between c and b. This is done by checking the sign of the product $f(a) \times f(c)$:*
   (a) *If $f(a) \times f(c) < 0$, the solution is between a and c?.*
   (b) *If $f(a) \times f(c) > 0$, the solution is between c and b.*
4. *Select the subinterval that contains the solution (a to c, or c to b) as the new interval $[a, b]$, and go back to step 2.*

Steps 2 through 4 are repeated until a specified tolerance or error bound is attained.

Example 1.6

Using the False Position method, find a root of the function $f(x) = e^x - 3x^2$ to an accuracy of 5 digits. The root is known to lie between 0.5 and 1.0.

Solution

We apply the method of False Position with $a = 0.5$ and $b = 1.0$

$$y - f(x_0) = \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0).$$

The calculations based on the method of False Position are shown in the following Table

Iteration	a	b	$f(a)$	$f(b)$	x	$f(x)$
1	*0.5*	*1*	*0.89872*	*-0.28172*	*0.88067*	*0.08577*
2	*0.88067*	*1*	*0.08577*	*-0.28172*	*0.90852*	*0.00441*
3	*0.90852*	*1*	*0.00441*	*-0.28172*	*0.90993*	*0.00022*
4	*0.90993*	*1*	*0.00022*	*-0.28172*	*0.91000*	*0.00001*
5	*0.91000*	*1*	*0.00001*	*-0.28172*	*0.91001*	*0*

Although false position often performs better than bisection, there are other cases where it does not. As in the following example, there are certain cases where bisection yields superior results.

$$\text{True percent relative error}(|\epsilon_t|) = \frac{|\text{true value} - \text{approximate value}|}{\text{true value}} \times 100\%$$

$$\text{Approximate percent relative error } (|\epsilon_a|) = \left|\frac{c^{\text{new}} - c^{\text{old}}}{c^{\text{new}}}\right| \times 100\%.$$

where c^{new} is the root for the present iteration and c^{old} is the root from the previous iteration. When ϵ_a becomes less than a prespecified stopping criterion ϵ, the computation is terminated.

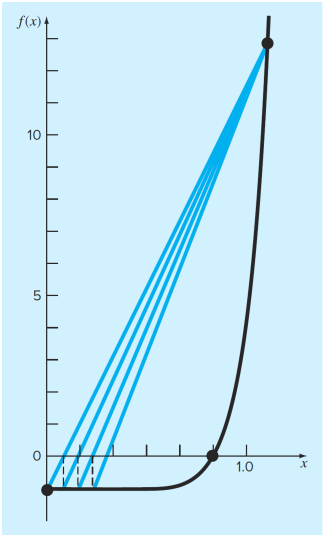

Figure 1.6: Plot of $f(x) = x^{10} - 1$, illustrating slow convergence of the false-position method.

Example 1.7: A Case Where Bisection Is Preferable to False Position

Use bisection and false position to locate the root of

$$f(x) = x^{10} - 1$$

between $x = 0$ *and* 1.3.

Solution

Using bisection, the results can be summarized as

Iteration	a	b	c	ϵ_a	ϵ_t
1	*0*	*1.3*	*0.65*	*100.0*	*35*
2	*0.65*	*1.3*	*0.975*	*33.3*	*2.5*
3	*0.975*	*1.3*	*1.1375*	*14.3*	*13.8*
4	*0.975*	*1.1375*	*1.05625*	*7.7*	*5.6*
5	*0.975*	*1.05625*	*1.015625*	*4.0*	*1.6*

Thus, after five iterations, the true error is reduced to less than 2%. For false position, a very different outcome is obtained:

Iteration	a	b	c	ϵ_a	ϵ_t
1	*0 1.3*	*0.09430*	*90.6*		
2	*0.09430*	*1.3*	*0.18176*	*48.1*	*81.8*
3	*0.18176*	*1.3*	*0.26287*	*30.9*	*73.7*
4	*0.26287*	*1.3*	*0.33811*	*22.3*	*66.2*
5	*0.33811*	*1.3*	*0.40788*	*17.1*	*59.2*

After five iterations, the true error has only been reduced to about 59%. Insight into these results can be gained by examining a plot of the function. As in Fig.1.6, the curve violates the premise on which false position was based that is, if $f(a)$ *is much closer to zero than* $f(b)$*, then the root should be much closer to* a *than to* b*.*

Exercise 1.3

Solve for a positive root of $x^3 - 4x + 1 = 0$ *by and Regula Falsi method (Hence the root is 2.7405.)*

1.3 Fixed point Iteration Method

Fixed-point iteration is a method for solving an equation of the form $f(x) = 0$. The method is carried out by rewriting the equation in the form:

$$x = g(x) \tag{1.7}$$

Obviously, when x is the solution of $f(x) = 0$, the left side and the right side of Eq. (1.7) are equal. This is illustrated graphically by plotting $y = x$ and $y = g(x)$, as shown in Fig.1.7. The point of intersection of the two plots, called the fixed point, is the solution. The numerical value of the solution is determined by an iterative process. It starts by taking a value of x near the **fixed point** as the first guess for the solution and substituting it in $g(x)$. The value of $g(x)$ that is obtained is the new (second) estimate for the solution. The second value is then

substituted x back in $g(x)$, which then gives the third estimate of the solution. The iteration formula is thus given by:

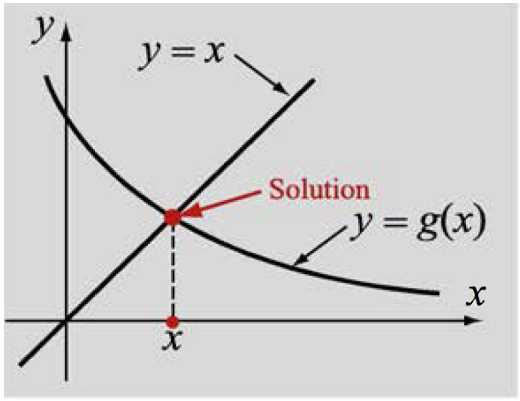

Figure 1.7: Fixed-point iteration method.

Definition 1.3

If we can write f(x)=0 in the form $x = g(x)$, then the point x would be a fixed point of the function g (that is, the input of g is also the output). Then an obvious sequence to consider is

$$x_{n+1} = g(x_n) \tag{1.8}$$

The function $g(x)$ is called the ***iteration function.***

- When the method works, the values of x that are obtained are successive iterations that progressively converge toward the solution. Two such cases are illustrated graphically in Fig.1.8. The solution process starts by choosing point x_1 on the x-axis and drawing a vertical line that intersects the curve $y = g(x)$ at point $g(x_1)$. Since $x_2 = g(x_1)$, a horizontal line is drawn from point $(x_1, g(x_1))$ toward the line $y = x$. The intersection point gives the location of x_2 . From x_2 a vertical line is drawn toward the curve $y = g(x)$. The intersection point is now $(x_2, g(x_2))$, and $g(x_2)$ is also the value of x_3. From point $(x_2, g(x_2))$ a horizontal line is drawn again toward $y = x$, and the intersection point gives the location of x_3? As the process continues the intersection points converge toward the fixed point, or the true solution x_{rs}.

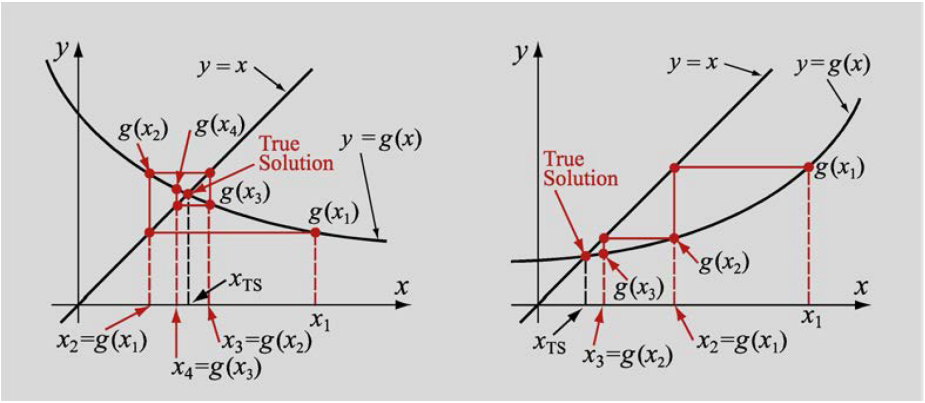

Figure 1.8: Convergence of the fixed-point iteration method.

- It is possible, however, that the iterations will not converge toward the fixed point, but rather diverge away. This is shown in Fig. 1.9. The figure shows that even though the starting point is close to the solution, the subsequent points are moving farther away from the solution.

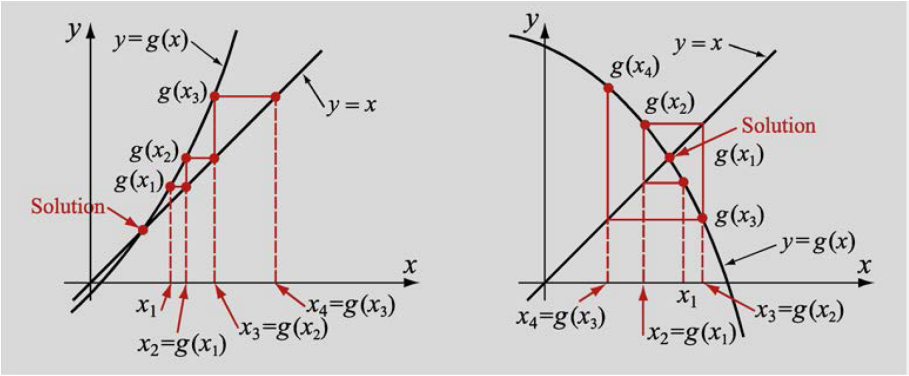

Figure 1.9: Divergence of the fixed-point iteration method.

- Sometimes, the form $f(x) = 0$ does not lend itself to deriving an iteration formula of the form $x = g(x)$. In such a case, one can always add and subtract x to $f(x)$ to obtain $x + f(x) - x = 0$. The last equation can be rewritten in the form that can be used in the fixed-point iteration method: $x = x + f(x) = g(x)$

1.3.1 Choosing the appropriate iteration function $g(x)$

For a given equation $f(x) = 0$, the iteration function is not unique since it is possible to change the equation into the form $x = g(x)$ in different ways. This means that several iteration functions g(x) can be written for the same equation. A $g(x)$ that should be used in Eq. (1.8) for the iteration process is one for which the iterations converge toward the solution. There might be more than one form that can be used, or it may be that none of the forms are appropriate so that the fixed-point iteration method cannot be used to solve the equation. In cases where there are multiple solutions, one iteration function may yield one root, while a different function yields other roots. Actually, it is possible to determine ahead of time if the iterations converge or diverge for a specific $g(x)$.

Theorem 1.4

The fixed-point iteration method converges if, in the neighborhood of the fixed point, the derivative of $g(x)$ has an absolute value that is smaller than 1 (also called Lipschitz continuous):

$$|g'(x)| < 1 \tag{1.9}$$

Algorithm 1.3

1. *Take an initial approximation* x_0
2. *Find the next (first) approximation* x_1 *by using* $x_1 = g(x_0)$
3. *Follow the above procedure to find the successive approximation*

$$x_{n+1} = g(x_n),\ n = 1, 2, 3, \cdot$$

4. *Stop evaluation where relative error less than the prescribed accuracy* ϵ.

Example 1.8

Consider the equation $f(x) = x^2 - 3x + 1 = 0$ *(whose true roots are* $a_1 = 0.381966$ *and* $a_2 = 2.618034$*. This can be rearranged as a fixed point problem in many different ways. Compare the following two algorithms.*

- $x_{n+1} = \frac{1}{3}(x_n^2 + 1) \equiv g_1(x_n)$
- $x_{n+1} = 3 - \frac{1}{x_n} \equiv g_2(x_n)$

1.3.2 Condition for the fixed point iteration scheme

Consider the equation $f(x) = 0$, which has the root α and can be written as the fixed point problem $g(x) = x$. If the following conditions hold

1. $g(x)$ and $g'(x)$ are continuous functions;
2. $|g'(\alpha)| < 1$ then the fixed point iteration scheme based on the function g will converge to α.
3. Alternatively, if $|g'(\alpha)| > 1$ then the iteration will not converge to α.
4. Note that when $|g'(\alpha)| = 1$ no conclusion can be reached.

Example 1.9

For the previous example, we have

$$g_1(x) = \frac{1}{3}(x_2 + 1) \Rightarrow g_1'(x) = \frac{2x}{3}$$

Evaluating the derivative at the two roots (or fixed points): $|g_1'(\alpha_1)| = 0.254\cdots < 1$ *and* $|g_1'(\alpha_2)| = 1.745\cdots > 1$ *so the first algorithm converges to* $\alpha_1 = 0.3819\cdots$ *but not to* $\alpha_2 = 2.618\cdots$ *. The second algorithm is given by*

$$g_2(x) = 3 - \frac{1}{x} \Rightarrow g_2'(x) = \frac{1}{x^2}$$

which gives

$$|g_2'(\alpha_1)| = 6.92\cdots > 1 \text{ and } |g_2'(\alpha_2)| = 0.13\cdots < 1$$

so the second algorithm converges to $\alpha_2 = 2.61\cdots$ *but not to* $\alpha_1 = 0.38\cdots$ *.*

n	x_n	x_{n+1}	$f(x_n)$	$x_n - x_{n-1}$
1.	*0.500000000*	*0.416666666*	*-0.076388888*	*-0.083333333*
2.	*0.416666666*	*0.391203703*	*-0.020570773*	*-0.025462963*
3.	*0.391203703*	*0.384346779*	*-0.005317891*	*-0.006856924*
4.	*0.384346779*	*0.382574148*	*-0.001359467*	*-0.001772630*
5.	*0.382574148*	*0.382120993*	*-0.000346526*	*-0.000453155*
6.	*0.382120993*	*0.382005484*	*-0.000088263*	*-0.000115508*
7.	*0.382005484*	*0.381976063*	*-0.000022477*	*-0.000029421*
8.	*0.381976063*	*0.381968571*	*-0.000005723*	*-0.000007492*
9.	*0.381968571*	*0.381966663*	*-0.000001457*	*-0.000001907*
10.	*0.381966663*	*0.381966177*	*-0.000000371*	*-0.000000485*

For the second root we use $g_2(x)$ *and the result is*

n	x_n	x_{n+1}	$f(x_n)$	$x_n - x_{n-1}$
1.	*2.750000000*	*2.636363636*	*0.041322314*	*-0.113636363*
2.	*2.636363636*	*2.620689655*	*0.005945303*	*-0.015673981*
3.	*2.620689655*	*2.618421052*	*0.000865651*	*-0.002268602*
4.	*2.618421052*	*2.618090452*	*0.000126259*	*-0.000330600*
5.	*2.618090452*	*2.618042226*	*0.000018420*	*-0.000048225*
6.	*2.618042226*	*2.618035190*	*0.000002687*	*-0.000007035*
7.	*2.618035190*	*2.618034164*	*0.000000392*	*-0.000001026*
8.	*2.618034164*	*2.618034014*	*0.000000057*	*-0.000000149*
9.	*2.618034014*	*2.618033992*	*0.000000008*	*-0.000000021*
10.	*2.618033992*	*2.618033989*	*0.000000001*	*-0.000000003*

Exercise 1.4

Consider the nonlinear equation

$$f(x) = x^3 - 2x^2 - 3 = 0$$

which has a root a between 2 and 3. (The true value is $\alpha = 2.485583998$.)

1. *Rewrite the equation $f(x) = 0$ as the fixed-point problem $g_1(x) = x$ where*

$$g_1(x) = 2 + \frac{3}{x^2}$$

   and, using the convergence criterion, show that the iteration algorithm associated with this problem converges to the root α.

2. *Create two other fixed-point iteration schemes, $g_2(x)$ and $g_3(x)$.*

3. *Perform ten iterations with six exact digits for each of the schemes $g_1(x), g_2(x)$ and $g_3(x)$ and compare the approximations. (Use the same starting point for all algorithms.)*

1.4 Newton-Raphson method

Newton's method is one of the most widely used of all iterative techniques for solving equations. Rather than using a secant line, the method uses a tangent line to the curve. Figure 1.10 gives a graphical interpretation of the method. To use the method we begin with an initial guess x_0, sufficiently close to the root x_*. The next approximation x_1 is given by the point at which the tangent line to f at $f(x_0, f(x_0))$ crosses the x-axis. It is clear that the value x_1 is much closer to x_* than the original guess x_0. If x_{n+1} denotes the value obtained by the succeeding iterations, that is the x-intercept of the tangent line to f at $(x_n, f(x_n))$, then a formula relating x_n and x_{n+1}, known as Newton's method, is given by

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \ n \geq 0 \tag{1.10}$$

If $f \in C^2[a, b]$, and we know $x_1 \in [a, b]$ be an approximation to x_* such that $f'(x_1) \neq 0$ and $|x_1 - x_*|$ is "small." Consider the first Taylor polynomial for $f(x)$ expanded about p_1 and evaluated at $x = x_*$:

$$0 = f(x_*) = f(x_1) + (x_1 - x_*)f'(x) + \frac{(x_1 - x_*)^2}{2!}f''(\xi(x_1)) \tag{1.11}$$

where $\xi(x_1) \in [x_1, x_*]$

Newton's method is derived by assuming that $|x_1 - x_*|$ is small, which means that $|x_1 - x_*|^2 \ll |x_1 - x_*|$, hence we make the approximation:

$$0 = f(x_1) + (x_1 - x_*)f'(x)$$

Now solve for x_*

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

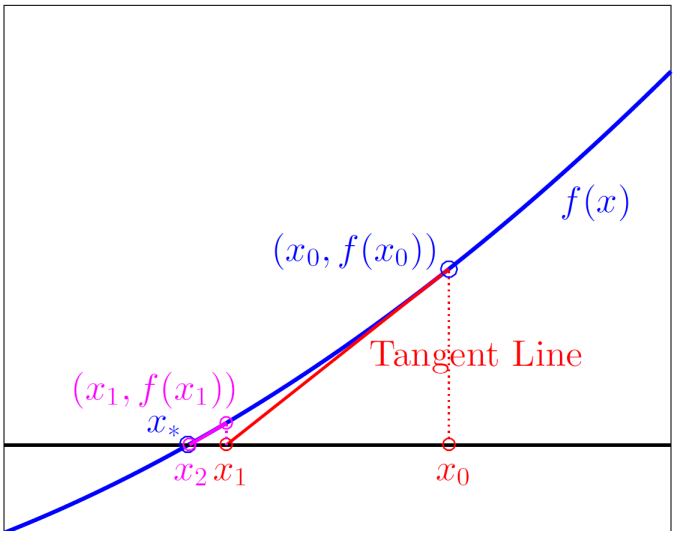

Figure 1.10: Newton's method.

Algorithm 1.4: Newton Method

Given a scalar differentiable function in one variable, $f(x)$:

1. *Start from an initial guess x_0.*
2. *For $n = 0, 1, 2, \cdots$, set*

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

until x_{n+1} satisfies termination criteria.

1.4.1 Finding a Starting Point for Newton's Method

Recall our initial argument that when $|x_1 - x_*|$ is small, then $|x_1 - x_*|^2 \ll |x_1 - x_*|$, and we can neglect the second order term in the Taylor expansion. In order for Newton's method to converge we need a and we can neglect the second order term in the Taylor expansion. In order for Newton's method to converge we need a **good starting point!**

Theorem 1.5

Let $f(x) \in C^2[a, b]$. If $x_* \in [a, b]$ such that $f(x_*) = 0$ and $f'(x_*) \neq 0$, then there exists a $\sigma > 0$ such that Newton's method generates a sequence $\{x_n\}_{n=1}\infty$ converging to x_* for any initial approximation $x_1 \in [x_* - \sigma, x_* + \sigma]$.

The theorem is interesting, but quite useless for practical purposes. In practice: Pick a starting value x_1, iterate a few steps. Either the iterates converge quickly to the root, or it will be clear that convergence is unlikely.

1.4.2 Implementation

Let's write a function called newton which takes 5 input parameters f, Df, x_0, epsilon and max_iter and returns an approximation of a solution of $f(x) = 0$ by Newton's method. The function may terminate in 3 ways:

1. If $abs(f(x_n)) < epsilon$, the algorithm has found an approximate solution and returns x_n.
2. If $f'(x_n) == 0$, the algorithm stops and returns **None**.
3. If the number of iterations exceed max_iter, the algorithm stops and returns **None**.

```
def newton(f,Df,x0,epsilon,max_iter):
    '''Approximate solution of f(x)=0 by Newton's method.

    Parameters
    ----------
    f : function
        Function for which we are searching for a solution f(x)=0.
    Df : function
        Derivative of f(x).
    x0 : number
        Initial guess for a solution f(x)=0.
    epsilon : number
        Stopping criteria is abs(f(x)) < epsilon.
    max_iter : integer
        Maximum number of iterations of Newton's method.

    Returns
    -------
    xn : number
        Implement Newton's method: compute the linear approximation
        of f(x) at xn and find x intercept by the formula
            x = xn - f(xn)/Df(xn)
        Continue until abs(f(xn)) < epsilon and return xn.
        If Df(xn) == 0, return None. If the number of iterations
        exceeds max_iter, then return None.
    xn = x0
    for n in range(0,max_iter):
        fxn = f(xn)
        if abs(fxn) < epsilon:
            print('Found solution after',n,'iterations.')
            return xn
        Dfxn = Df(xn)
        if Dfxn == 0:
            print('Zero derivative. No solution found.')
            return None
        xn = xn - fxn/Dfxn
    print('Exceeded maximum iterations. No solution found.')
    return None
```


Example 1.10: Supergolden Ratio

Let's test our function on $f(x) = x^3 - x^2 - 1$

```
f = lambda x: x**3 - x**2 - 1
Df = lambda x: 3*x**2 - 2*x
approx = newton(f,Df,1,1e-10,10)
print(approx)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Found solution after 6 iterations.
1.4655712318767877
```

Example 1.11: Divergent Example

Newton's method diverges in certain cases. For example, if the tangent line at the root is vertical as in . Note that bisection and secant methods would converge in this case.

```
f = lambda x: x**(1/3)
Df = lambda x: (1/3)*x**(-2/3)
approx = newton(f,Df,0.1,1e-2,100)
Exceeded maximum iterations. No solution found.
```

Example 1.12

Using Newton's method, solve $f(x) = x^6 - x - 1$.

Solution

Here

$$f(x) = x^6 - x - 1,\ f'(x) = 6x^5 - 1$$

and the iteration

$$x_{n+1} = x_n - \frac{x_n^6 - x_n - 1}{6x_n^5 - 1},\ \neq 0$$

The true root is $\alpha = 1.134724138$, *and* $x_6 = \alpha$ *to nine significant digits. Newton,s method may converge slowly at first. However, as the iterates come closer to the root, the speed of convergence increases.*

n	x_n	$f(x_n)$	$x_n - x_{n-1}$	$\alpha - x_{n-1}$
0	1.5	8.89E+1		
1	1.30049088	2.54E+1	-2.00E-1	-3.65E-1
2	1.18148042	5.38E−1	-1.19E-1	-1.66E-1
3	1.13945559	4.92E−2	-4.20E-2	-4.68E-2
4	1.13477763	5.50E−4	-4.68E-3	-4.73E-3
5	1.13472415	7.11E−8	-5.35E-5	-5.35E-5
6	1.13472414	1.55E−15	-6.91E-9	-6.91E-9
	1.134724138			

Table: Newton's Method for $x^6 - x - 1 = 0$

Exercise 1.5

1. *Let $p(x) = x^3 - x - 1$. The only real root of $p(x)$ is called the* ***plastic number*** *and is given by*
$$\frac{\sqrt[3]{108 + 12\sqrt{69}} + \sqrt[3]{108 - 12\sqrt{69}}}{6}$$
2. *Choose $x_0 = 1$ and implement 2 iterations of Newton's method to approximate the plastic number.*
3. *Use the exact value above to compute the absolute error after 2 iterations of Newton's method.*
4. *Starting with the subinterval $[1, 2]$, how many iterations of the bisection method is required to achieve the same accuracy?*

1.5 Secant Method

Newton's method is an extremely powerful technique, but it has a major weakness; the need to know the value of the derivative of f at each approximation. Frequently, $f'(x)$ is far more difficult and needs more arithmetic operations to calculate than $f(x)$.

The secant method is a variant of Newton's method, where $f'(x_n)$ is replaced by its finite difference approximation based on the evaluated function values at x_n and at the previous iterate x_{n-1}. Assuming convergence, observe that near the root

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

Substitution of this approximation into the formula for Newton's method yields the **Secant method**,

$$x_{n+1} = \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}, \ n = 0, 1, 2, 3, \cdots$$

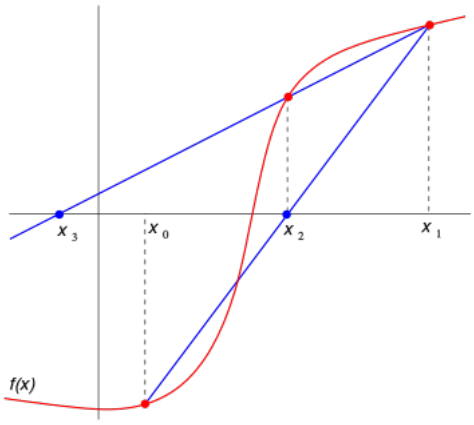

Figure 1.11: The first two iterations of the secant method. The red curve shows the function f and the blue lines are the secants. For this particular case, the secant method will not converge.

Algorithm 1.5

Given a scalar differentiable function in one variable, $f(x)$:

1. *Start from two initial guesses x_0 and x_1.*
2. *For $k = 1, 2, \cdots$, set*
$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}$$
until x_{n+1} satisfies termination criteria.

1.5.1 Derivation of the method

Because the bisection and the false position methods converge at a very slow speed, our next approach is an attempt to produce a method that is faster. One such method is the secant method. Similar to the false position method, it is based on approximating the function by a straight line connecting two points on the graph of the function f, but we do not require f to have opposite signs at the initial points. Figure 1.11 illustrates the method.

In this method, the first point, x_2, of the iteration is taken to be the point of intersection of the x-axis and the secant line connecting two starting points $(x_0, f(x_0))$ and $(x_1, f(x_1))$. The next point, x_3, is generated by the intersection of the new secant line, joining $(x1, f(x_1))$ and $(x_2, f(x_2))$ with the x-axis. The new point, x_3, together with x_2, is used to generate the next point, x_4, and so on. A formula for x_{n+1} is obtained by setting $x = x_{n+1}$ and $y = 0$ in the equation of the secant line from $(x_{n-1}, f(x_{n-1}))$ to $(x_n, f(x_n))$.

$$x_2 = x_1 - f(x_1)\frac{x_1 - x_0}{f(x_1) - f(x_0)}$$

$$x_3 = x_2 - f(x_2)\frac{x_2 - x_1}{f(x_2) - f(x_1)}$$

$$\vdots$$

$$x_n = x_{n-1} - f(x_{n-1})\frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})}$$

1.5.2 Convergence

The iterates x_n of the secant method converge to a root of f, if the initial values x_0 and x_1 are sufficiently close to the root. The order of convergence is α where

$$\alpha = \frac{1 + \sqrt{5}}{2} \approx 1.618$$

is the golden ratio. In particular, the convergence is superlinear, but not quite quadratic.

This result only holds under some technical conditions, namely that f be twice continuously differentiable and the root in question be simple (i.e., with multiplicity 1).

If the initial values are not close enough to the root, then there is no guarantee that the secant method converges. There is no general definition of "close enough", but the criterion has to do with how "wiggly" the function is on the interval [x_0, x_1]. For example, if f is differentiable

on that interval and there is a point where $f' = 0$ on the interval, then the algorithm may not converge.

Example 1.13

Find a root of the equation

$$x^6 - x - 1 = 0$$

Solution

We apply the method of secant method with $x_1 = 1$ *and* $x_2 = 1.5$.

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

The calculations based on the secant method are shown in the following Table

Iteration	a	b	$f(a)$	$f(b)$	x	$f(x)$
1.	*1.000000000*	*2.000000000*	*-1.000000000*	*61.000000000*	*1.016129032*	*-0.915367713*
2.	*2.000000000*	*1.016129032*	*61.000000000*	*-0.915367713*	*1.030674754*	*-0.831921414*
3.	*1.016129032*	*1.030674754*	*-0.915367713*	*-0.831921414*	*1.175688944*	*0.465227164*
4.	*1.030674754*	*1.175688944*	*-0.831921414*	*0.465227164*	*1.123679065*	*-0.110632879*
5.	*1.175688944*	*1.123679065*	*0.465227164*	*-0.110632879*	*1.133671081*	*-0.010805918*
6.	*1.123679065*	*1.133671081*	*-0.110632879*	*-0.010805918*	*1.134752681*	*0.000293664*
7.	*1.133671081*	*1.134752681*	*-0.010805918*	*0.000293664*	*1.134724065*	*-0.000000748*
8.	*1.134752681*	*1.134724065*	*0.000293664*	*-0.000000748*	*1.134724138*	*-0.000000000*
9.	*1.134724065*	*1.134724138*	*-0.000000748*	*-0.000000000*	*1.134724138*	*-0.000000000*
10.	*1.134724138*	*1.134724138*	*-0.000000000*	*-0.000000000*	*1.134724138*	*-0.000000000*

(Recall that the true root is $\alpha = 1.134724138$.*)*

Note: We can use a similar argument as in Newton's method to show that

$$\alpha - x_{n-1} \approx x_n - x_{n-1}$$

so that we have a measure of the absolute error at each step.

Advantages and disadvantages:

1. The error decreases slowly at first but then rapidly after a few iterations.
2. The secant method is slower than Newton's method but faster than the bisection method.
3. Each iteration of Newton's method requires two function evaluations, while the secant method requires only one
4. The secant method does not require differentiation.

1.6 Root Finding Methods Summary

1. The Bisection method
   (a) Very stable Algorithm - Good technique to
   nd starting point for Newton's method
   (b) Costs only one function evaluation, so rapid iterations
   (c) Linear convergence, so slow (3.3 iterations/digit)
2. The Secant method

(a) Hard to find starting points (Unknown basin of attraction)

(b) Costs only two function evaluations, so rapid iterations

(c) Superlinear convergence,$\alpha \approx 1.62$, which is pretty fast

3. The Newton's method

(a) Hard to find starting points (Unknown basin of attraction)

(b) Finding and evaluating derivative requires more machine work at each iteration

(c) Quadratic convergence is very fast- doubling the digits at each iteration.

Example 1.14: Summary

Find the roots of

$$x^3 + 4x^2 - 10 \quad x \in [1.5, 2]$$

n	*Bisection*	*Secant*	*Newton*
1	*1.25*	*1.33898305084745*	*1.45454545454545*
2	*1.375*	*1.36356284991687*	*1.36890040106951*
3	*1.3125*	*1.36525168742565*	*1.36523660020211*
4	*1.34375*	*1.36522999568865*	*1.36523001343536*
5	*1.359375*	*1.36523001341391*	*1.36523001341409*
6	*1.3671875*	*1.36523001341409*	
7	*1.36328125*		
8	*1.365234375*		
9	*1.3642578125*		
10	*1.36474609375*		
11	*1.364990234375*		
12	*1.3651123046875*