Chapter 1

Numerical Methods for the Root Finding Problem

Oct. 11, 2011 HG

1.1 A Case Study on the Root-Finding Problem: Kepler's Law of Planetary Motion

The root-finding problem is one of the most important computational problems. It arises in a wide variety of practical applications in **physics**, **chemistry**, **biosciences**, **engineering**, etc. As a matter of fact, determination of any unknown appearing implicitly in scientific or engineering formulas gives rise to a root-finding problem. We consider one such simple application here.

One of the classical laws of planetary motion due to Kepler says that *a planet revolves around the sun in an elliptic orbit as shown in the figure below*:

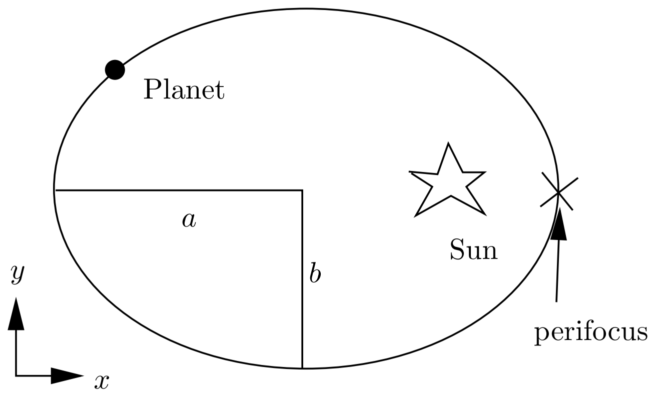
Suppose one needs to find the position (x, y) of the planet at time t. This can be determined by the following formula:

$$\begin{aligned} x &= a\cos(E - e) \\ y &= a\sqrt{1 - e^2}\sin E \end{aligned}$$

where

$$\begin{aligned} e &= \text{The eccentricity of the Ellipse} \\ E &= \text{Eccentric anomaly} \end{aligned}$$

Figure 1.1: A planet in elliptic orbit around the sun.



To determine the position (x, y), one must know how to compute E, which can be computed from Kepler's equation of motion:

$$M = E - e\sin(E), \quad 0 < e < 1,$$

where M is the mean anomaly. This equation, thus, relates the eccentric anomaly, E to the mean anomaly, M. Thus to find E we can solve the **nonlinear equation**:

$$f(E) = M - E + e\sin(E) = 0$$

The solution of such an equation is the subject of this chapter.

A solution of this equation with numerical values of M and e using several different methods described in this Chapter will be considered later.

1.2 Introduction

As the title suggests, the **Root-Finding Problem** is the problem of finding a root of the equation $f(x) = 0$, where $f(x)$ is a function of a single variable x. Specifically, the problem is stated as follows:

The Root-Finding Problem

Given a function $f(x)$. Find a number $x = \xi$ such that $f(\xi) = 0$.

Definition 1.1. The number $x = \xi$ such that $f(\xi) = 0$ is called a **root** of the equation $f(x) = 0$ or a **zero** of the function $f(x)$.

As seen from our case study above, the root-finding problem is a classical problem and dates back to the 18th century. The function $f(x)$ can be **algebraic** or **trigonometric** or a combination of both.

1.2.1 Analytical versus Numerical Methods

Except for some very special functions, it is not possible to find an analytical expression for the root, from where the solution can be exactly determined. This is true for even commonly arising polynomial functions. A polynomial $P_n(x)$ of degree n has the form:

$$P_n(x) = a_0 + a_1(x) + a_2x^2 + \cdots + a_nx^n \quad (a_n \neq 0)$$

The **Fundamental Theorem of Algebra** states that a polynomial $P_n(x)$ of degree $n (n \geq 1)$ has at least one zero.

A student learns very early in school, how to solve a quadratic equation: $P_2(x) = ax^2 + bx + c$ using the analytical formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

(see later for a numerically effective version of this formula.)

Unfortunately, *such analytical formulas do not exist for polynomials of degree 5 or greater.* This fact was proved by **Abel** and **Galois** in the 19th century.

Thus, most computational methods for the root-finding problem have to be iterative in nature. The **idea** behind an iterative method is the following:

Starting with an initial approximation x_0, construct a sequence of iterates $\{x_k\}$ using an iteration formula with a hope that this sequence converges to a root of $f(x) = 0$.

Two important aspects of an iterative method are: **convergence** and **stopping criterion**.

- We will discuss the convergence issue of each method whenever we discuss such a method in this book.

- Determining an universally accepted stopping criterion is complicated for many reasons.

Here is a **rough guideline**, which can be used to terminate an iteration in a computer program.

Stopping Criteria for an Iterative Root-Finding Method

Accept $x = c_k$ as a root of $f(x) = 0$ if any one of the following criteria is satisfied:

1. $|f(c_k)| \leq \epsilon$ (*The functional value is less than or equal to the tolerance*).

2. $\frac{|c_{k-1} - c_k|}{|c_k|} \leq \epsilon$ (*The relative change is less than or equal to the tolerance*).

3. *The number of iterations k is greater than or equal to a predetermined number, say N.*

1.3 Bisection-Method

As the title suggests, the method is based on repeated bisections of an interval containing the root. The basic idea is very simple.

Basic Idea:

Suppose $f(x) = 0$ is known to have a real root $x = \xi$ in an interval $[a, b]$.

- Then **bisect** the interval $[a, b]$, and let $c = \frac{a+b}{2}$ be the middle point of $[a, b]$. If c is the root, then we are done. Otherwise, one of the intervals $[a, c]$ or $[c, b]$ will contain the root.
- Find the one that contains the root and bisect that interval again.
- Continue the process of bisections until the root is trapped in an interval as small as warranted by the desired accuracy.

To implement the above idea, we must know in each iteration: which of the two intervals contain the root of $f(x) = 0$.

The **Intermediate Value Theorem** of calculus can help us identify the interval in each iteration. For a proof of this theorem, see any calculus book (e.g. Stewart []).

Intermediate Value Theorem (IVT)

Suppose

(i) $f(x)$ is continuous on a closed interval $[a, b]$, and

(ii) M is any number between $f(a)$ and $f(b)$.

Then

there is at least one number c in $[a, b]$ such that $f(c) = M$.

Consequently, if

(i) $f(x)$ is continuous on $[a, b]$,

and

(ii) $f(a)$ and $f(b)$ are of opposite signs.

Then there is a root $x = c$ of $f(x) = 0$ or in $[a, b]$.

Algorithm 1.2 (The Bisection Method for Rooting-Finding).

Inputs: $f(x)$ - The given function.
a_0, b_0 - The two numbers such that $f(a_0)f(b_0) < 0$.

Output: An approximation of the root of $f(x) = 0$ in $[a_0, b_0]$.

For $k = 0, 1, 2, \ldots$, do until satisfied:

- Compute $c_k = \frac{a_k + b_k}{2}$.
- Test, using one of the criteria stated in the next section, if c_k is the desired root. If so, stop.
- If c_k is not the desired root, test if $f(c_k)f(a_k) < 0$. If so, set $b_{k+1} = c_k$ and $a_{k+1} = a_k$. Otherwise, set $a_{k+1} = c_k$, $b_{k+1} = b_k$.

End.

Example 1.3

Find the positive root of $f(x) = x^3 - 6x^2 + 11x - 6 = 0$ using Bisection method.

Solution:

Finding the interval $[a, b]$ bracketing the root:

Since the bisection method finds a root in a given interval $[a, b]$, we must try to find that interval first. This can be done using IVT.

Choose $a_0 = 2.5$ and $b_0 = 4$.

We now show that both hypotheses of IVT are satisfied for $f(x)$ in $[2.5, 4]$.

(i) $f(x) = x^3 - 6x^2 + 11x - 6$ is continuous on $[2.5, 4]$.

(ii) $f(2.5)f(4) < 0$.

Thus, by IVT, there is a root of $f(x) = 0$ in $[2.5, 4]$.

Input Data:

$$\begin{cases} \text{(i) } f(x) = x^3 - 6x^2 + 11x - 6 \\ \text{(ii) } a_0 = 2.5, \quad b_0 = 4 \end{cases}$$

Solution.

Iteration 1. ($\boldsymbol{k = 0}$):

$$c_0 = \frac{a_0 + b_0}{2} = \frac{4 + 2.5}{2} = \frac{6.5}{2} = 3.25.$$

Since $f(c_0)f(a_0) = f(3.25)f(2.5) < 0$, set $b_1 = c_0$, $a_1 = a_0$.

Iteration 2. ($\boldsymbol{k = 1}$):

$$c_1 = \frac{3.25 + 2.5}{2} = 2.8750.$$

Since $f(c_1)f(a_1) > 0$, set $a_2 = 2.875$, $b_2 = b_1$.

Iteration 3. ($\boldsymbol{k = 2}$):

$$c_2 = \frac{a_2 + b_2}{2} = \frac{2.875 + 3.250}{2} = 3.0625.$$

Since $f(c_2)f(a_2) = f(3.0625)f(2.875) < 0$, set $b_3 = c_2$, $a_3 = a_2$.

Iteration 4. ($\boldsymbol{k = 3}$):

And $c_3 = \frac{a_3+b_3}{2} = \frac{2.875+3.0625}{2} = 2.9688$.

It is clear that the iterations are converging towards the root $x = 3$.

Note:

1. From the statement of the bisection algorithm, it is clear that *the algorithm always converges.*

2. The example above shows that the convergence, however, can be very slow.

3. *Computing* c_k: It might happen that at a certain iteration k, computation of $c_k = \frac{a_t+b_k}{2}$ will give overflow. It is better to compute c_k as:

$$c_k = a_k + \frac{b_k - a_k}{2}.$$

Stopping Criteria

Since this is an iterative method, we must determine some stopping criteria that will allow the iteration to stop. Here are some commonly used stopping criteria.

Let ϵ be the tolerance; that is, we would like to obtain the root with an error of at most of ϵ. Then

Note: Criterion 1 can be misleading since it is possible to have $|f(c_k)|$ very small, even if c_k is not close to the root. (**Do an example to convince yourself that it is true**).

Number of Iterations Needed in the Bisection Method to Achieve a Certain Accuracy

Let's now find out what is the *minimum number of iterations* N needed with the bisection method to achieve a certain desired accuracy.

Criterion 3 can be used to answer this.

The interval length after N iterations is $\frac{b_0-a_0}{2^N}$. So, to obtain an accuracy of ϵ, we must have $\frac{b_0-a_0}{2^N} \leq \epsilon$.

That is,

$$\begin{aligned} & 2^{-N}(b_0 - a_0) \leq \epsilon, \\ \text{or} \quad & 2^N \geq \frac{b_0 - a_0}{\epsilon}, \\ \text{or} \quad & N \ln 2 \geq \ln(b_0 - a_0) - \ln \epsilon, \ \text{(Taking the natural log on both sides.)} \\ \text{or} \quad & N \geq \frac{\ln(b_0 - a_0) - \ln \epsilon}{2}. \end{aligned}$$

Theorem 1.4. The number of iterations N needed in the bisection method to obtain an accuracy of ϵ is given by

$$N \geq \frac{\log_{10}(b_0 - a_0) - \log_{10}(\epsilon)}{\log_{10} 2}. \tag{1.1}$$

Remark: Note the number N depends only on the initial interval $[a_0, b_0]$ bracketing the root.

Example 1.5

Find the minimum number of iterations needed by the bisection algorithm to approximate the root $x = 3$ of $x^3 - 6x^2 + 11x - 6 = 0$ with error tolerance 10^{-3}.

Input Data:

$$\begin{cases} \text{End points of the interval: } a = 2.5, \quad b = 4 \\ \text{Error tolerance: } t = 10^{-3} \end{cases}$$

Formula to be used: Inequality (1.1).

Substituting the values of a_0 and b_0 in the above formula, we get

$$N \geq \frac{\log_{10}(1.5) - \log_{10}(10^{-3})}{\log_{10} 2} = \frac{\log_{10}(1.5) + 3}{\log_{10}(2)} = 10.5507.$$

Thus, a minimum of 11 iterations will be needed to obtain the desired accuracy using the bisection method.

Remarks: (i) Since the number of iterations N needed to achieve a certain accuracy depends upon the initial length of the interval containing the root, *it is desirable to choose the initial interval* $[a_0,\ b_0]$ *as small as possible.*

1.3.1 A Stopping Criterion for the Bisection Method

Besides the stopping criteria mentioned in the Introduction, a suitable stopping criterion, specific for the bisection method, will be:

Stop the bisection iteration if for any k:

$$\frac{b-a}{2^k} \leq \epsilon$$

(*The length of the interval after k iterations is less than or equal to the tolerance ϵ.*)

1.4 Fixed-Point Iteration

Definition 1.6. A number ξ is a **fixed point** of a function $g(x)$ if $g(\xi) = \xi$.

Suppose that the equation $f(x) = 0$ is written in the form $x = g(x)$; that is,

$$f(x) = x - g(x) = 0.$$

Then any fixed point ξ of $g(x)$ is a root of $f(x) = 0$ because

$$f(\xi) = \xi - g(\xi) = \xi - \xi = 0.$$

Thus, a root of $f(x) = 0$ can be found by finding a fixed point of $x = g(x)$, which corresponds to $f(x) = 0$.

Finding a root of $f(x) = 0$ by finding a fixed point of $x = g(x)$ immediately suggests an iterative procedure of the following type.

Start with an initial guess x_0 of the root and form a sequence $\{x_k\}$ defined by

$$x_{k+1} = g(x_k), \quad k = 0, 1, 2, \ldots$$

If the sequence $\{x_k\}$ converges, then $\lim_{k\to\infty} x_k = \xi$ will be a root of $f(x) = 0$.

The question therefore rises:

Given $f(x) = 0$ in $[a, b]$.

How do we write $f(x) = 0$ in the form $x = g(x)$ such that the sequence $\{x_k\}$ is defined by

$$x_{k+1} = g(x_k);$$

will converge to the root $x = \xi$ for any choice of the initial approximation x_0?

The simplest way to write $f(x) = 0$ in the form $x = g(x)$ is to add x on both sides, that is,

$$x = f(x) + x = g(x).$$

But it does not very often work.

To convince yourself, consider **Example 1.3** again. Here,

$$f(x) = x^3 - 6x^2 + 11x - 6 = 0.$$

Define

$$g(x) = x + f(x) = x^3 - 6x^2 + 12x - 6.$$

We know that there is a root of $f(x)$ in $[2.5,\ 4]$; namely $x = 3$.

Let's start the iteration $x_{k+1} = g(x_k)$ with $x_0 = 3.5$.

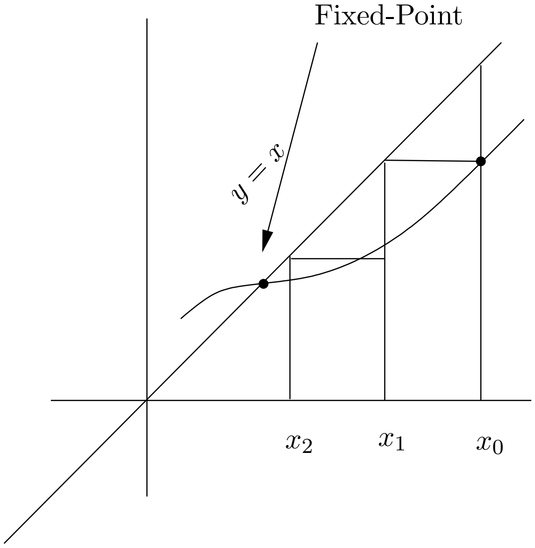
Then we have:

$$\begin{aligned} x_1 &= g(x_0) = g(3.5) = 5.3750, \\ x_2 &= g(x_1) = g(5.3750) = 40.4434, \\ x_3 &= g(x_2) = g(40.4434) = 5.6817 \times 10^4, \\ \text{and } x_4 &= g(x_3) = g(5.6817 \times 10^4) = 1.8340 \times 10^{14}. \end{aligned}$$

The sequence $\{x_k\}$ is clearly diverging.

The convergence and divergence of the fixed-point iteration are illustrated by the following graphs.

Figure 1.2: Convergence of the Fixed-Point Iteration



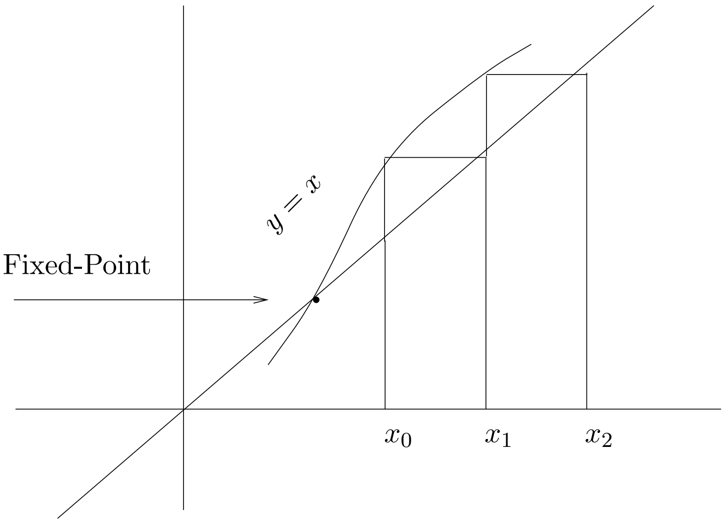
A Sufficient Condition for Convergence

The following theorem gives a sufficient condition on $g(x)$ which ensures the convergence of the sequence $\{x_k\}$ with any initial approximation x_0 in $[a, b]$.

Theorem 1.7 (Fixed-Point Iteration Theorem). Let $f(x) = 0$ be written in the form $x = g(x)$. Assume that $g(x)$ satisfies the following properties:

(i) For all x in $[a,\ b]$, $g(x) \in [a,\ b]$; that is $g(x)$ takes every value between a and b.

Figure 1.3: Divergence of the Fixed-Point Iteration



(ii) $g'(x)$ exists on $(a,\ b)$ with the property that there exists a positive constant $0 < r < 1$ such that

$$|g'(x)| \leq r,$$

for all x in (a, b).

Then

(i) there is a unique fixed point $x = \xi$ of $g(x)$ in $[a, b]$,

(ii) for any x_0 in $[a,\ b]$, the sequence $\{x_k\}$ defined by

$$x_{k+1} = g(x_k),\ k = 0, 1, \cdots$$

converges to the fixed point $x = \xi$; that is, to the root ξ of $f(x) = 0$.

The proof of the fixed-point theorem will require two important theorems from calculus: **Initial Value Theorem** (IVT) and the **Mean Value Theorem** (MVT). The IVT has been stated earlier. We will now state the MVT.

The Mean Value Theorem (MVT)

Let $f(x)$ be a function such that

(i) it is continuous on $[a,\ b]$

(ii) it is differentiable on (a, b)

Then there is a number c in (a, b) such that

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

Proof of the Fixed-Point Theorem:

The proof comes in three parts: **Existence, Uniqueness, and Convergence**.

Since a fixed point of $g(x)$ is a root of $f(x) = 0$, this amounts to proving that there is a fixed point in $[a,\ b]$ and it is unique.

Proof of Existence

First, consider the case where either $g(a) = a$ or $g(b) = b$.

- If $g(a) = a$, then $x = a$ is a fixed point of $g(x)$. Thus, $x = a$ is a root of $f(x) = 0$.
- If $g(b) = b$, then $x = b$ is a fixed point of $g(x)$. Thus $x = b$ is a root of $f(x) = 0$.

Next, consider the general case where the above assumptions are not true. That is, $g(a) \neq a$ and $g(b) \neq b$.

In such a case, $g(a) > a$ and $g(b) < b$. This is because, by Assumption 1, both $g(a)$ and $g(b)$ are in $[a, b]$, and since $g(a) \neq a$ and $g(b) \neq b$, $g(a)$ must be greater than a and $g(b)$ must be greater than b.

Now, define the function $h(x) = g(x) - x$. *We will now show that $h(x)$ satisfies both the hypotheses of the Intermediate Value Theorem (IVT).*

In this context, we note

(i) since $g(x)$ is continuous by our assumptions, $h(x)$ is also continuous on $[a, b]$. **(Hypothesis (i) is satisfied)**

(ii) $h(a) = g(a) - a > 0$ and $h(b) = g(b) - b < 0$. **(Hypothesis (ii) is satisfied)**

Since both hypotheses of the IVT are satisfied, by this theorem, there exists a number c in $[a, b]$ such that

$$h(c) = 0$$

this means $g(c) = c$.

That is, $x = c$ is a fixed point of $g(x)$. This proves the *Existence part of the Theorem.*

Proof of Uniqueness (by contradiction):

We will prove the uniqueness by contradiction. Our line of proof will be as follows:

We will first assume that there are two fixed points of $g(x)$ in $[a, b]$ and then show that this assumption leads to a contradiction.

Suppose ξ_1 and ξ_2 are two fixed points in $[a,\ b]$, and $\xi_1 \neq \xi_2$.

The proof is based on the *Mean Value Theorem applied to* $g(x)$ in $[\xi_1, \xi_2]$.

In this context, note that $g(x)$ *is differentiable* and hence, *continuous* on $[\xi_1,\ \xi_2]$.

So, by MVT, we can find a number c in (ξ_1, ξ_2) such that

$$\frac{g(\xi_2) - g(\xi_1)}{(\xi_2 - \xi_1)} = g'(c).$$

Since $g(\xi_1) = \xi_1$, and $g(\xi_2) = \xi_2$, (because they are fixed points of $g(x)$), we get

$$\frac{\xi_2 - \xi_1}{\xi_2 - \xi_1} = g'(c).$$

That is, $g'(c) = 1$, *which is a contradiction to our Assumption 2.* Thus, ξ_1 *can not be different from* ξ_2, this proves the uniqueness.

Proof of Convergence:

Let ξ be the root of $f(x) = 0$. Then the **absolute error** at step $k + 1$ is given by

$$e_{k+1} = |\xi - x_{k+1}|. \tag{1.2}$$

To prove that the sequence converges, we need to show that $\lim_{k\to\infty} e_{k+1} = 0$.

To show this, we apply the Mean Value Theorem to $g(x)$ in $[x_k, \xi]$. Since $g(x)$ also satisfies both the hypotheses of the MVT in $[x_k, \xi]$, we get

$$\frac{g(\xi) - g(x_k)}{\xi - x_k} = g'(c) \tag{1.3}$$

where $x_k < c < \xi$.

Now, $x = \xi$ is a fixed point of $g(x)$, so we have (by definition):

(i) $g(\xi) = \xi$.

Also, from the iteration $x = g(x)$, we have

(ii) $x_{k+1} = g(x_k)$.

So, from ( ), we get

$$\frac{\xi - x_{k+1}}{\xi - x_k} = g'(c)$$

Taking absolute values on both sides we have

$$\frac{|\xi - x_{k+1}|}{|\xi - x_k|} = |g'(c)| \tag{1.4}$$

or

$$\frac{e_{k+1}}{e_k} = |g'(c)|. \tag{1.5}$$

Since $|g'(c)| \leq r$, we have $e_{k+1} \leq r e_k$.

Since the above relation ( ) is true for every k, we have $e_k \leq r e_{k-1}$. Thus, $e_{k+1} \leq r^2 e_{k-1}$.

Continuing in this way, we finally have $e_{k+1} \leq r^{k+1} e_0$, where e_0 is the initial error. (That is, $e_0 = |x_0 - \xi|$).

Since $r < 1$, we have $r^{k+1} \to 0$ as $k \to \infty$.

Thus,

$$\begin{aligned}\lim_{k\to\infty} e_{k+1} &= \lim_{k\to\infty} |x_{k+1} - \xi| \\ &\leq \lim_{k\to\infty} r^{k+1} e_0 = 0\end{aligned}$$

This proves that the sequence $\{x_k\}$ converges to $x = \xi$ and that $x = \xi$ is the only fixed point of $g(x)$.

In practice, it is not easy to verify Assumption 1. The following corollary of the fixed-point theorem is more useful in practice because the conditions here are more easily verifiable. We leave the proof of the corollary as an [**Exercise**].

Corollary 1.8. Let the iteration function $g(x)$ be such that

(i) it is continuously differentiable in some open interval containing the fixed point $x = \xi$,

(ii) $|g'(\xi)| < 1$.

Then there exists a number $\epsilon > 0$ such that the iteration: $x_{k+1} = g(x_k)$ converges whenever x_0 is chosen in $|x_0 - \xi| \leq \epsilon$.

Example 1.9

Find a positive zero of $x^3 - 6x^2 + 11x - 6$ in $[2.5, 4]$ using fixed-point iterations.

Input Data:

$$\begin{cases} \text{(i) } f(x) = x^3 - 6x^2 + 11x - 6, \\ \text{(ii) } a = 2.5, b = 4. \end{cases}$$

Solution.

Step 1. *Writing $f(x) = 0$ in the form $x = g(x)$ so that both hypotheses of Theorem 1.7 are satisfied.*

We have seen in the last section that the obvious form: $g(x) = x + f(x) = x^3 - 6x^2 + 12x - 6$ does not work.

Let's now write $f(x) = 0$ in the form: $x = \frac{1}{11}(-x^3 + 6x^2 + 6) = g(x)$.

Then (i) for all $x \in [2.5, 4], g(x) \in [2.5, 4]$ (Note that $g(2.5) = 2.5341$ and $g(4) = 3.4545$).

(ii) $g'(x) = \frac{1}{11}(-3x^2 + 12x)$ steadily decreases in $[2.5, 4]$ and remains less than 1 for all x in $(2.5, 4)$. (Note that $g'(2.5) = 1.0227$ and $g'(4) = 0$.)

Thus, both hypotheses of Theorem 1.6 are satisfied. So, *according to this theorem, for any x_0 in $[2.5, 4]$, the sequence $\{x_k\}$ should converge to the fixed point.*

We now do a few iterations to verify this assertion. Starting with $x_0 = 3.5$ and using the iteration:

$$x_{k+1} = g(x_k) = -\tfrac{1}{11}(3x_k^2 + 12x_k)$$

$$(k = 0, 1, 2, \ldots, 11)$$

We obtain:

$$
\begin{aligned}
x_0 &= 3.5 \\
x_1 = g(x_0) &= 3.3295 \\
x_2 = g(x_1) &= 3.2367 \\
x_3 = g(x_2) &= 3.1772 \\
x_4 = g(x_x) &= 3.1359 \\
x_5 = g(x_4) &= 3.1059 \\
x_6 = g(x_5) &= 3.0835 \\
x_7 = g(x_6) &= 3.0664 \\
x_8 = g(x_7) &= 3.0531 \\
x_9 = g(x_8) &= 3.0427 \\
x_{10} = g(x_9) &= 3.0344 \\
x_{11} = g(x_{10}) &= 3.0278
\end{aligned}
$$

The *sequence is clearly converging to the root* $x = 3$.

Example 1.10

Find a root of $x - \cos x = 0$ in $(0, \frac{\pi}{2})$.

Input Data: (i) $f(x) = x - \cos x$

(ii) $a = 0, b = \frac{\pi}{2}$

Solution.

Write $f(x) = 0$ in the form $x = g(x)$ so that $x_{k+1} = g(x_k)$ converges for any choice of x_0 in $[0, \frac{\pi}{2}]_n$:

From $f(x) = x - \cos x = 0$, we obtain $x = \cos x$.

Thus, if we take $g(x) = \cos x$, then

(i) For all x in $[0, \frac{\pi}{2}]$, $g(x) \in [0, \frac{\pi}{2}]$. (In particular, note that $g(0) = \cos(0) = 1$ and $g(\frac{\pi}{2}) = \cos(\frac{\pi}{2}) = 0$).

(ii) $g'(x) = -\sin x$. Thus, $|g'(x)| < 1$ in $[0, \frac{\pi}{2}]$.

Thus, both properties of $g(x)$ of Theorem 1.6 are satisfied.

According to the Fixed-Point Iteration Theorem (**Theorem 1.7**), the iterations must converge with any choice of x_0 in $[0, \frac{\pi}{2}]$. This is verified from the following computations.

$$\begin{aligned}
x_0 &= 0\\
x_1 &= \cos x_o = 1\\
x_2 &= \cos x_1 = 0.5403\\
x_3 &= \cos x_2 = 0.8576\\
&\vdots\\
x_{17} &= 0.73956\\
x_{18} &= \cos x_{17} = 0.73955
\end{aligned}$$

1.5 The Newton Method

The Newton- method, described below, shows that there is a special choice of $g(x)$ that will allow us to readily use the results of **Corollary 1.8**.

Assume that $f''(x)$ exists and is continuous on $[a,\ b]$ and ξ is a **simple root** of $f(x)$, that is, $f(\xi) = 0$ and $f'(\xi) \neq 0$.

Choose

$$g(x) = x - \frac{f(x)}{f'(x)}.$$

Then $g'(x) = 1 - \frac{(f'(x))^2 - f(x)f''(x)}{(f'(x))^2} = \frac{f(x)f''(x)}{(f'(x))^2}$.

Thus, $g'(\xi) = \frac{f(\xi)f''(\xi)}{(f'(\xi))^2} = 0$, since $f(\xi) = 0$, and $f'(\xi) \neq 0$.

Since $g'(x)$ is continuous, this means that there exists a small neighborhood around the root $x = \xi$ such that for all points x in that neighborhood, $|g'(x)| < 1$.

Thus, if $g(x)$ is chosen as above and the *starting approximation* x_0 *is chosen sufficiently close to the root* $x = \xi$, then the fixed-point iteration is guaranteed to converge.

This leads us to the following well-known classical method known as the **Newton** method.

Note: In many text books and literature, Newton's Method is referred to as **Newton's Method**. See the article about this in *SIAM Review* (1996).

Algorithm 1.11 (The Newton Method).

Inputs: $f(x)$ - The given function
x_0 - The initial approximation
ϵ - The error tolerance
N - The maximum number of iterations

```
Output: An approximation to the root $x = \xi$ or a message of failure.

Assumption: $x = \xi$ is a simple root of $f(x)$.

For $k = 0, 1, \cdots$, do until convergence or failure.

    • Compute $f(x_k), f'(x_k)$.
    • Compute $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$.
    • Test for convergence or failure:
        $\left.\begin{array}{ll} \text{If} & |f(x_k)| < \epsilon, \\ \text{or} & \frac{|x_{k+1} - x_k|}{|x_k|} < \epsilon, \\ \text{or} & k > N, \ \text{stop.} \end{array}\right]$ stopping criteria.

End
```

Definition 1.12. The iterations $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$ are called **Newton's iterations**.

Example 1.13

Find a positive real root of $\cos x - x^3 = 0$.

Obtaining an initial approximation: Since $\cos x \leq 1$ for all x and for $x > 1$, $x^3 > 1$, the positive zero must lie between 0 and 1. So, let's take $x_0 = 0.5$.

Input Data:

$$\begin{cases} \text{(i) The function } f(x) = \cos x - x^3 \\ \text{(ii) Initial approximation: } x_0 = 0.5 \end{cases}$$

Formula to be used: $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$

Solution.

Compute $f'(x) = -\sin x - 3x^2$.

Iteration 1. k=0: $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 0.5 - \frac{\cos(0.5) - (0.5)^3}{-\sin(0.5) - 3(0.5)^2} = 1.1121$.

Iteration 2. k=1: $x_2 = x_1 - \frac{f(x_1}{f'(x_1)} = 0.9097$.

Iteration 3. k=2: $x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = 0.8672$.

Iteration 4. k=3: $x_4 = x_3 - \frac{f(x_3)}{f'(x_3)} = 0.8654$.

Iteration 5. k=4: $x_5 = x_4 - \frac{f(x_4)}{f'(x_4)} = 0.8654.$

Notes:

1. If none of the above criteria has been satisfied within a predetermined, say N, iterations, then the method has failed after the prescribed number of iterations. In that case, one could try the method again with a different x_0.
2. A judicious choice of x_0 can sometimes be obtained by drawing the graph of $f(x)$, if possible. However, there does not seem to exist a clear-cut guideline of how to choose a right starting point x_0 that guarantees the convergence of the Newton Method to a desired root.
3. **Starting point issues.**
   (a) If x_0 is far from the solution, Newton's method may diverge [**Exercise**].
   (b) For some functions, some starting point may enter an infinite circle in the sense that the sequence of iterations will oscillate without converging to a root [**Exercise**].

1.5.1 Some Familiar Computations Using Newton's Method

Computing the Square Root of a Positive Number A:

Compute $\sqrt{A}$, where $A > 0$.

Computing $\sqrt{A}$ is equivalent to solving $x^2 - A = 0$. The number $\sqrt{A}$, thus, may be computed by applying the Newton Method to $f(x) = x^2 - A$.

Since $f'(x) = 2x$, we have the following algorithm for computing the square root of a positive number A.

Newton Iterations for Computing $\sqrt{A}$

Input: A - A positive number

Output: An approximation to $\sqrt{A}$

Step 1. Choose an initial approximation x_0 to $\sqrt{A}$.

Step 2. Compute the successive approximations $\{x_k\}$ as follows:

```
For $k = 0, 1, 2, \ldots,$ do until convergence
    $x_{k+1} = x_k - \frac{x_k^2 - A}{2x_k} = \frac{x_k^2 + A}{2x_k}$
End
```

Example 1.14 Approximate the Positive Square Root of 2 Using Newton's Method with $x_0 = 1.5$ (Do three iterations).

Input Data: $A = 2$, $x_0 = 1.5$

Formula to be Used: $x_{k+1} = \frac{x_k^2+A}{2x_k}$, $k = 0, 1$, and 2.

Solution.

Iteration 1. $x_1 = \frac{x_0^2+A}{2x_0} = \frac{(1.5)^2+2}{3} = 1.4167$,

Iteration 2. $x_2 = \frac{x_1^2+A}{2x_1} = 1.4142$,

Iteration 3. $x_3 = \frac{x_2^2+A}{2x_2} = 1.4142$.

Computing the n^{th} Root

It is easy to see that the above Newton- Method to compute $\sqrt{A}$ can be easily generalized to compute $\sqrt[n]{A}$. In this case, $f(x) = x^n - A$, $f'(x) = nx^{n-1}$.

Thus, Newton's iterations in this case are:

Newton's Iteration for the nth Root

$$x_{k+1} = x_k - \frac{x_k^n - A}{nx_k^{n-1}} = \frac{(n-1)x_k^n + A}{nx_k^{n-1}}.$$

Finding the Reciprocal of a Nonzero Number A

The problem here is to compute $\frac{1}{A}$, where $A \neq 0$. Since $f'(x) = -\frac{1}{x^2}$, the Newton iteration in this case is:

Thus, computing $\frac{1}{A}$ is equivalent to finding the root of the equation: $f(x) = \frac{1}{x} - A$.

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{\frac{1}{x_k} - A}{-\frac{1}{x_k^2}} = x_k(2 - Ax_k).$$

```
**The Newton Iteration for Computing $\frac{1}{A}$**

Input: $A$ - A nonzero number

Output: An approximation to $\frac{1}{A}$

Step 1. Choose an initial approximation $x_0$ to $\frac{1}{A}$.

Step 2. For $k = 0, 1, 2, \ldots$ do until convergence
            $x_{k+1} = x_k(2 - Ax_k)$
        End
```

Example 1.15

Approximate $\frac{1}{3}$ by Newton method with $x_0 = 2.5$ *(Do three iterations.)*

Input Data:

$A = 3, x_0 = 0.25$

Formula to be Used:

$x_{k+1} = x_k(2 - Ax_k), k = 0, 1, 2$

Solution.

Iteration 1. $x_1 = x_0(2 - Ax_0) = 0.25(2 - 3 \times 0.025) = 0.3125$

Iteration 2. $x_2 = x_1(2 - Ax_1) = 0.3125(2 - 3 \times 0.3125) = 0.3320$

Iteration 3. $x_3 = x_2(2 - Ax_2) = 0.3320(2 - 3 \times 0.3320) = 0.3333$

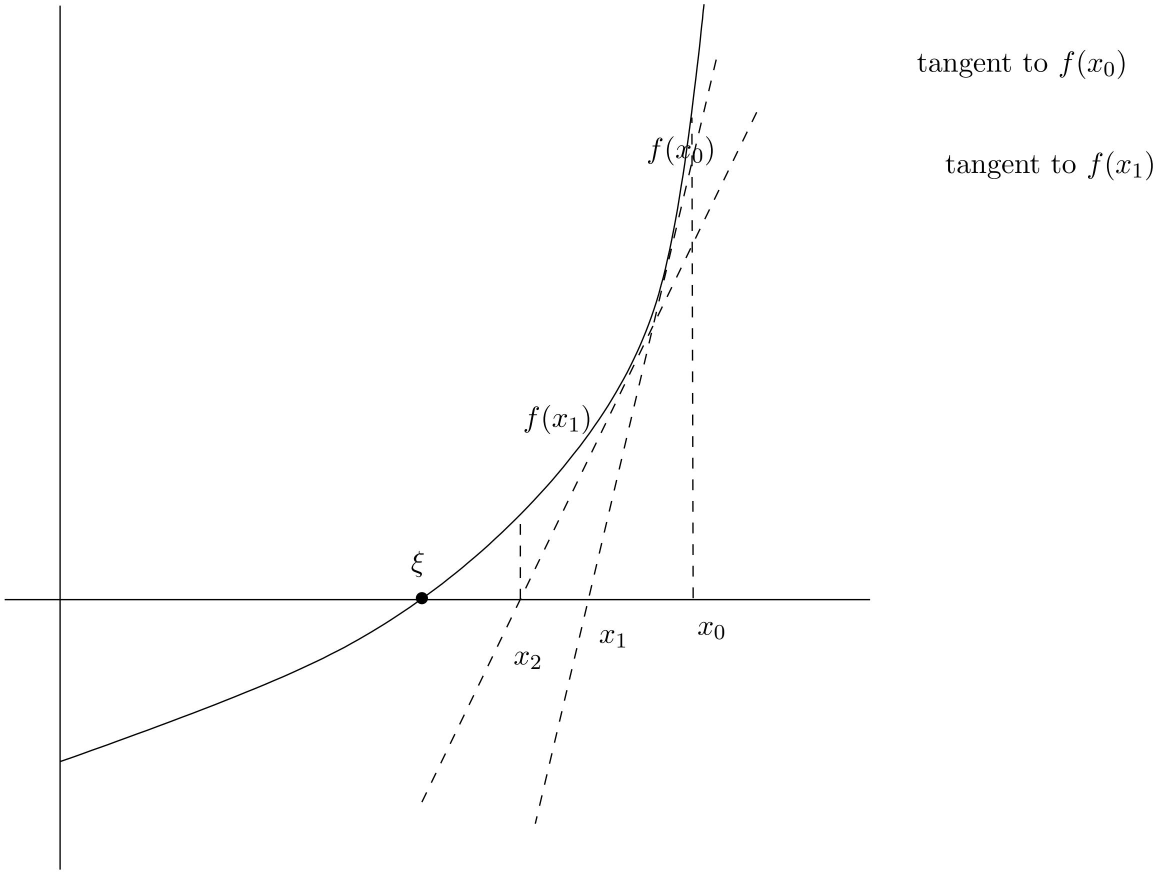
Note *that 0.3333 is the 4-digit approximation of* $\frac{1}{3}$ *(correct up to 4 decimal figures).*

1.5.2 A Geometric Interpretation of the Newton- Method

- The first approximation x_1 can be viewed as the x-intercept of the tangent line to the graph of $f(x)$ at $(x_0, f(x_0))$.

- The second approximation x_2 is the x-intercept of the tangent line to the graph of $f(x)$ at $(x_1, f(x_1))$ and so on.

Figure 1.4: Graphical Representation of the Newton Method.



1.5.3 The Secant Method

A major disadvantage of the Newton Method is the requirement of finding the value of the derivative of $f(x)$ at each approximation. There are functions for which this job is either extremely difficult (if not impossible) or time consuming. A way out is to approximate the derivative by knowing the values of the function at that and the previous approximation. Knowing $f(x_k)$ and $f(x_{k-1})$, we can approximate $f'(x_k)$ as:

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

Then, the Newton iterations in this case become:

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

This consideration leads to the **Secant method**.

Algorithm 1.16 (The Secant Method).

Inputs: $f(x)$ - The given function
x_0, x_1 - The two initial approximations of the root
ϵ - The error tolerance
N - The maximum number of iterations

Output: An approximation of the exact solution ξ or a message of failure.

For $k = 1, 2, \cdots$, until the stopping criteria is met,

- Compute $f(x_k)$ and $f(x_{k-1})$.
- Compute the next approximation: $x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}$.
- Test for convergence or maximum number of iterations: If $|x_{k+1} - x_k| < \epsilon$ or if $k > N$, Stop.

End

Note: The secant method needs two approximations x_0 and x_1 to start with, whereas the Newton method just needs one, namely, x_0.

Example 1.17

Approximate the Positive Square Root of 2, choosing $x_0 = 1.5$ and $x_1 = 1$ *(Do four iterations).*

Input Data: $\begin{cases} \text{The function } f(x) = x^2 - 2 \\ \text{Initial approximations: } x_0 = 1.5, x_1 = 1 \end{cases}$

Formula to be Used: $x_{k+1} = x_c - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}$

Solution.

Iteration 1. $(k = 1)$. Compute x_2 from x_0 and x_1:

$$
\begin{aligned}
x_2 &= x_1 - \frac{f(x_1)(x_1 - x_0)}{f(x_1) - f(x_0)} \\
&= 1 - \frac{(-1)(-.5)}{-1 - .25} = 1 + \frac{.5}{1.25} = 1.4,
\end{aligned}
$$

Iteration 2. ($k = 2$). Compute x_3 from x_1 and x_2:

$$\begin{aligned} x_3 &= x_2 - \frac{f(x_2)(x_2 - x_1)}{f(x_2) - f(x_1)} \\ &= 1.4 - \frac{(-0.04)(0.4)}{(-0.04) - (-1)} \\ &= 1.4 + 0.0167 \\ &= 1.4167, \end{aligned}$$

Iteration 3. ($k = 3$). Compute x_4 from x_2 and x_3:

$$\begin{aligned} x_4 &= x_3 - \frac{f(x_3)(x_3 - x_2)}{f(x_3) - f(x_2)} \\ &= 1.4142, \end{aligned}$$

Iteration 4. ($k = 4$). Compute x_5 from x_3 and x_4:

and

$$x_5 = x_4 - \frac{f(x_4)(x_4 - x_3)}{f(x_4) - f(x_3)} = 1.4142.$$

Note: By comparing the results of this example with those obtained by the Newton method **Example 1.14**, we see that

it took 4 iterations by the secant method to obtain a 4-digit accuracy of $\sqrt{2} = 1.4142$*; whereas for the Newton method, this accuracy was obtained just after 2 iterations.*

In general, *the Newton method converges faster than the secant method.* The exact rate of convergence of these two and the other methods will be discussed in the next section.

1.5.4 A Geometric Interpretation of the Secant Method

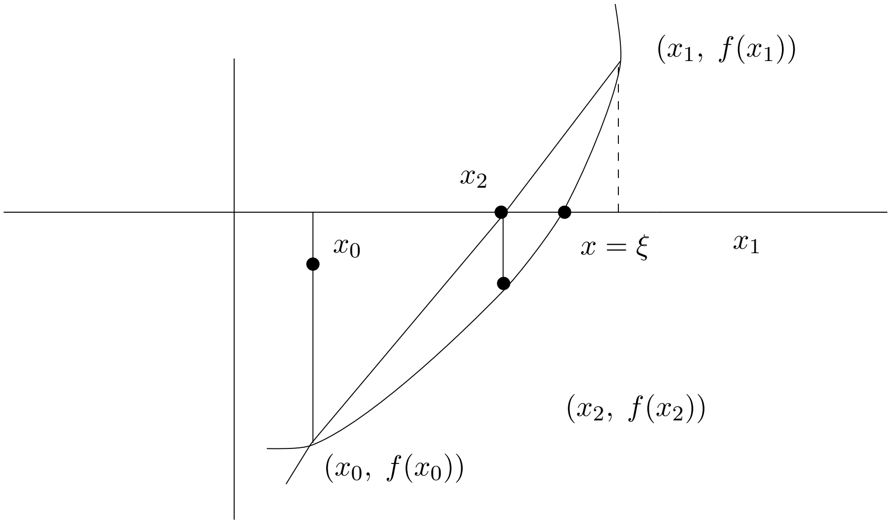
- x_2 is the x-intercept of the secant line passing through $(x_0, f(x_0))$ and $(x_1, f(x_1))$.
- x_3 is the x-intercept of the secant-line passing through $(x_1, f(x_1))$ and $(x_2, f(x_2))$.
- And so on.

Note: That is why the method is called the secant method.

1.6 The Method of False Position (The Regular Falsi Method)

In the Bisection method, every interval under consideration is guaranteed to have the desired root $x = \xi$. That is why the Bisection method is often called a **bracket method**, because every

Figure 1.5: Geometric Interpretation of the Secant Method



interval brackets the root. However, the *Newton method and the secant method are not bracket methods in that sense*, because there is no guarantee that the two successive approximations will bracket the root. *Here is an example.*

Example 1.18

(The Secant Method does not bracket the root.)

Compute a zero of $\tan(\pi x) - 6 = 0$, choosing $x_0 = 0$, and $x_1 = 0.48$, using the **Secant Method**.

(Note that $\xi = 0.44743154$ is a root of $f(x) = 0$ lying in the interval [0, 0.48]).

Iteration 1. $(k = 1)$. Compute x_2:

$x_2 = x_1 - f(x_1)\frac{(x_1 - x_0)}{f(x_1) - f(x_0)} = 0.181194.$

Iteration 2. $(k = 2)$. Compute x_3:

$x_1 = 0.48$, $x_2 = 0.181194$. (*Two starting initial approximations for Iteration 2.*)

$x_3 = x_2 - f(x_2)\frac{(x_2 - x_1)}{f(x_2) - f(x_1)} = 0.286187$

Iteration 3. $(k = 3)$. Compute x_4: $x_2 = 0.181194$, $x_3 = 0.286187$. (*Two starting initial approximations for Iteration 3.*)

$x_4 = x_3 - f(x_3)\frac{(x_3 - x_2)}{f(x_3) - f(x_2)} = 1.091987.$

Clearly, **the iterations are diverging, despite of the fact of the initial interval** $[0, 0.48]$

brackets the root.

However, *the secant method can easily be modified to make it a bracket method.* The idea is as follows:

Modification of the Secant Method to Make it a Bracket Method: Method of False Positives:

- Choose the initial approximations x_0 and x_1 such that $f(x_0)\ f(x_1) < 0$, ensuring that the root is in $[x_0,\ x_1]$.
- Compute x_2 as in the Secant method; that is, take x_2 as the x-intercept of the secant line passing through x_0 and x_1.
- Compute x_3 out of the three approximation x_0, x_1, and x_2 as follows:

  First, compute $f(x_1)f(x_2)$; if it is negative, then $[x_1,\ x_2]$ brackets the root and x_3 is the x-intercept of the secant line joining $(x_1,\ f(x_1))$ and $(x_2,\ f(x_2))$ (as in the Secant method). Otherwise, compute x_3 as the x-intercept of the line joining $(x_0,\ f(x_0))$ and $(x_2,\ f(x_2))$.
- Continue the process.

This yields a method called, the **Method of False Position** or the **Method of Regula Falsi**.

Algorithm 1.19 (The Method of False Position).

```
Input:  $f(x)$ - The given function
        $x_0$, $x_1$ - The two initial approximations $f(x_0)f(x_1) < 0$.
        $\epsilon$ - The tolerance
        $N$ - The maximum number of iterations
Output: An approximation to the root $x = \xi$.

Step 0. Set $i = 1$.

Step 1. Compute $x_{i+1} = x_i - f(x_i)\frac{(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}$.
        If $|\frac{x_{i+1} - x_i}{x_i}| < \epsilon$ or $i \geq N$, stop.

Step 2. If $f(x_i)f(x_{i+1}) < 0$, then set $x_i \equiv x_{i+1}$, and $x_{i-1} \equiv x_i$.
        Otherwise, set $x_i \equiv x_{i+1}$, $x_{i-1} \equiv x_{i-1}$. Return to Step 0.

End
```

Example 1.20

Consider again the previous example: (**Example 1.18**). *Find the positive root* of $\tan(\pi x) - 6 = 0$ in $[0, 0.48]$, using the **Method of False Position**.

Input Data: $\begin{cases} f(x) = \tan(\pi x) - 6, \\ \text{Initial approximation: } x_0 = 0, x_1 = 0.48 \end{cases}$

Formula to be Used: $x_{i+1} = x_i - f(x_i)\frac{(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}$

Solution.

Iteration 1. (Initial approximations for Iteration 1: $x_0 = 0, x_1 = 0.48$).

Step 1. $x_2 = x_1 - f(x_1)\frac{x_1 - x_0}{f(x_1) - f(x_0)} = 0.181192$

Step 2. $f(x_2)f(x_1) < 0$

Set $x_0 = 0.48, \quad x_1 = 0.181192$.

Iteration 2. (Initial approximations for Iteration 2: $x_0 = 0.48, x_1 = 0.181192$).

Step 1. $x_2 = x_1 - f(x_1)\frac{x_1 - x_0}{f(x_1) - f(x_0)} = 0.286186$

Step 2. $f(x_2)f(x_1) > 0$

Set $x_0 = 0.48$, $x_1 = 0.286186$.

Iteration 3. (Initial approximations for Iteration 3: $x_0 = 0.48, x_1 = 0.286186$).

Step 1. $x_2 = 0.348981$

Step 2. $f(x_2)f(x_1) > 0$

Set $x_0 = 0.48$, $x_1 = 0.348981$.

Iteration 4. (Initial approximations for Iteration 4: $x_0 = 0.48, x_1 = 0.348981$).

Step 1. $x_2 = 0.387053$

Step 2. $f(x_2)f(x_1) > 0$

Iteration 5. (Initial approximations for Iteration 5: $x_0 = 0.48, x_1 = 0.387053$).

Step 1. $x_2 = 0.410305$

1.7 Convergence Analysis of the Iterative Methods

We have seen that the Bisection method always converges, and each of the methods: the Newton Method, the Secant method, and the method of False Position, converges under certain conditions. The question now arises:

When a method converges, how fast does it converge?

In other words, what is the rate of convergence? To this end, we first define:

Definition 1.21 (Rate of Convergence of an Iterative Method). Suppose that the sequence $\{x_k\}$ converges to ξ. Then the sequence $\{x_k\}$ is said to converge to ξ with the *order of convergence* α if there exists a **positive constant** p such that

$$\lim_{k\to\infty} \frac{|x_{k+1}-\xi|}{|x_k-\xi|^\alpha} = \lim_{k\to\infty} \frac{e_{k+1}}{e_k^\alpha} = p.$$

- If $\alpha = 1$, the convergence is **linear**
- If $\alpha = 2$, the convergence is **quadratic**
- If $1 < \alpha < 2$, the convergence is **superlinear**

Using the above definition, we will now show:

- The rate of convergence of the fixed-point iteration is usually linear,
- The rate of convergence of the Newton method is quadratic,
- The rate of convergence of the Secant method is superlinear [**Exercise**].

Rate of Convergence of the Fixed-Point Iteration Method

Recall that in the proof of the **Fixed-Point Iteration Theorem**, we established the relation

$$e_{k+1} = |g'(c)|e_k,$$

where c is a number between x_k and ξ.

So, $\lim_{k\to\infty} \frac{e_{k+1}}{e_k} = \lim_{k\to\infty} |g'(c)|$.

Since $\{x_k\} \to \xi$, and c is in (x_k, ξ), it follows that $\lim_{k\to\infty} |g'(c)| = |g'(\xi)|$.

This gives $\lim_{k\to\infty} \frac{e_{k+1}}{e_k} = |g'(\xi)|$. Thus $\alpha = 1$.

We, therefore, have

The fixed point iteration scheme has linear rate of convergence, when it converges.

Rate of Convergence of the Newton Method

To study the rate of convergence of the Newton Method, we need Taylor's Theorem.

Taylor's Theorem of order n

Suppose

(i) $f(x)$ possesses continuous derivatives of order up to $n+1$ in $[a,\ b]$,

(ii) p is a point in $[a, b]$.

Then for every x in this interval, there exists a number c (depending upon x) between p and x such that

$$f(x) = f(p) + f'(p)(x-p) + \frac{f'(p)}{2!}(x-p)^2 + \cdots + \frac{f^n(p)}{n!}(n-p)^n + R_n(x),$$

where $R_n(x)$, called the **remainder after n terms**, is given by:

$$R_n(x) = \frac{f^{(n+1)}(c)}{(n+1)!}(x-p)^{n+1}.$$

Let's choose a small interval around the root $x = \xi$. Then, for any x in this interval, we have, by *Taylor's theorem of order 1*, the following expansion of the function $g(x)$:

$$g(x) = g(\xi) + (x-\xi)g'(\xi) + \frac{(x-\xi)^2}{2!}\, g''(\eta_k)$$

where η_k lies between x and ξ.

Now, for the Newton Method, we have seen that $g'(\xi) = 0$.

So, setting $g'(\xi) = 0$ in the above expression, we have

$$g(x) = g(\xi) + \frac{(x-\xi)^2}{2!}\, g''(\eta_k),$$

$$\text{or} \qquad g(x_k) = g(\xi) + \frac{(x_k-\xi)^2}{2!}\, g''(\eta_k),$$

$$\text{or} \qquad g(x_k) - g(\xi) = \frac{(x_k-\xi)^2}{2!}\, g''(\eta_k).$$

Since $g(x_k) = x_{k+1}$, and $g(\xi) = \xi$, from the last equation, we have

$$x_{k+1} - \xi = \frac{(x_k-\xi)^2}{2!}\, g''(\eta_k).$$

Or, $e_{k+1} = \frac{e_k^2}{2!}\|g'(\eta_k)\|$. Taking the limit on both sides, $\displaystyle\lim_{k\to 0} \frac{e_{k+1}}{e_k^2} = \frac{1}{2}\|g^{''}(\eta_k)\|$.

That is, $|x_{k+1} - \xi| = \frac{|x_k-\xi|^2}{2}\, |g''(\eta_k)|$.

Since η_k lies between x and ξ for every k, it follows that

$$\lim_{k\to\infty} \eta_k = \xi.$$

So, we have $\lim_{k\to\infty} \frac{e_{k+1}}{e_k^2} = \lim \frac{|g''(\eta_k)|}{2} = \frac{|g''(\xi)|}{2}$.

Convergence of Newton's Method for a Single Root

- Newton's Method has quadratic rate of convergence.
- The quadratic convergence roughly means that the accuracy gets doubled at each iteration.

Convergence of the Newton Method for a Multiple Root

The above proof depends on the assumption $f'(\xi) = 0$; that is, $f(x)$ has a simple root at $x = \xi$.

In case $f(x) = 0$ has a multiple root, the convergence is no longer quadratic.

It can be shown [**Exercise**] that the Newton method still converges, but *the rate of convergence is linear rather than quadratic.* For example, given the multiplicity is 2, the error decreases by a factor of approximately $\frac{1}{2}$ with each iteration.

The Rate of Convergence for the Secant Method

It can be shown that *if the Secant method converges, then the convergence factor α is approximately 1.6 for sufficiently large k* [**Exercise**].

This is said as "*The rate of convergence of the Secant method is superlinear*".

1.8 A Modified Newton Method for Multiple Roots

Recall that the most important underlying assumption in the proof of quadratic convergence of the Newton method was that $f'(x)$ *is not zero at the approximation $x = x_k$, and in particular* $f'(\xi) \neq 0$. This means that ξ is a simple root of $f(x) = 0$.

The question, therefore, is: *How can the Newton method be modified for a multiple root so that the quadratic convergence can be retained?*

Note that if $f(x)$ has a root ξ of multiplicity m, then

$$f(\xi) = f'(\xi) = f''(\xi) = \cdots = f^{(m-1)}(\xi) = 0,$$

where $f^{(m-1)}(\xi)$ denotes the $(m-1)^{\text{th}}$ derivative of $f(x)$ at $x = \xi$.

In this case $f(x)$ can be written as

$$f(x) = (x - \xi)^m h(x),$$

where $h(x)$ is a polynomial of degree at most $m - 2$.

In case $f(x)$ has a root of multiplicity m, it was suggested by Ralston and Robinowitz (1978) that, the following modified iteration will ensure quadratic convergence in the Newton method, under certain conditions [**Exercise**].

The Newton Iterations for Multiple Roots of Multiplicity m: Method I

$$x_{i+1} = x_i - \frac{mf(x_i)}{f'(x_i)}$$

Since in practice, the multiplicity m is not known a priori, another *useful modification is to apply the Newton iteration to the function:*

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

in place of $f(x)$.

Thus, in this case, the modified Newton iteration becomes $x_{i+1} = x_i - \frac{u(x_i)}{u'(x_i)}$.

Note that, since $f(x) = (x-\xi)^m\, h(x)$, $u'(x) \neq 0$ at $x = \xi$.

Since $u'(x_i) = \frac{f'(x)f'(x) - f(x)f''(x)}{[f'(x)]^2}$ we have

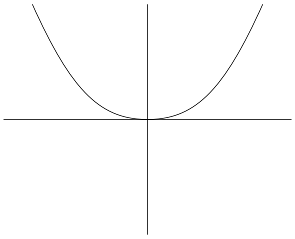
The Newton Iterations for Multiple Roots: Method II

$$x_{i+1} = x_i - \frac{f(x_i)f'(x_i)}{[f'(x_i)]^2 - [f(x_i)]f''(x_i)}$$

Example 1.22

Approximate the double root $x = 0$ of $e^x - x - 1 = 0$, choosing $x_0 = 0.5$. (Do two iterations.)

Figure 1.6: The Graph of $f(x) = e^x - x - 1$.



Method I

Input Data:

$$\begin{cases} \text{(i) } f(x) = e^x - x - 1 \\ \text{(ii) Multiplicity of the roots: } m = 2 \\ \text{(iii) Initial approximation: } x_0 = 0.5 \end{cases}$$

Formula to be Used: $x_{i+1} = x_i - \frac{2f(x_i)}{f'(x_i)}$

Solution.

Iteration 1. $(i = 0)$. Compute x_1:

$$x_1 = 0.5 - \frac{2(e^{0.5} - 0.5 - 1)}{(e^{0.5} - 1)} = .0415 = 4.15 \times 10^{-2}$$

Iteration 2. $(i = 1)$. Compute x_2:

$$x_2 = 0.0415 - \frac{2(e^{0.0415} - 0.0415 - 1)}{e^{0.0415} - 1} = .00028703 = 2.8703 \times 10^{-4}.$$

Method II

Input Data: $\begin{cases} \text{(i) } f(x) = e^x - x - 1, \\ \text{(ii) Initial approximation: } x_0 = 0.5 \end{cases}$

Formula to be Used: $x_{i+1} = x_i - \frac{f(x_i)f'(x_i)}{(f'(x_i)^2 - f(x_i)f''(x_i))}$

Solution.

Compute the first and second derivatives of $f(x)$: $f'(x) = e^x - 1$, $f^{''}(x) = e^x$.

Iteration 1. $(i = 0)$. Compute x_1:

$$\begin{aligned} x_1 &= \tfrac{f(x_0)f'(x_0)}{(f'(x_0))^2 f(x_0)f^{''}(x_0)} \\ &= \tfrac{(e^{0.5}-0.5-1)\times(e^{0.5}-1)}{(e^{0.5})^2\times(e^{0.5}-0.5-1)\times e^{0.5}} \\ &= 0.0493 = 4.93 \times 10^{-2} \end{aligned}$$

Iteration 2. $(i = 1)$. Compute x_2:

$$\begin{aligned} x_2 &= \tfrac{f(x_1)f'(x_1)}{(f'(x_1))^2 f(x_1)f^{''}(x_1)} \\ &= \tfrac{(e^{0.0493}-0.0493-1)\times(e^{0.0493}-1)}{(e^{0.0493})^2\times(e^{0.0493}-0.0493-1)\times e^{0.0493}} \\ &= 4.1180 \times 10^{-4} \end{aligned}$$

1.9 The Newton Method for Polynomials

A major requirement of all the methods that we have discussed so far is the evaluation of $f(x)$ at successive approximations x_k. Furthermore, Newton's method requires evaluation of the derivatives of $f(x)$ at each iteration.

If $f(x)$ happens to be a polynomial $P_n(x)$ (which is the case in many practical applications), then there is an extremely simple classical technique, known as **Horner's Method** to compute $P_n(x)$ at $x = z$.

- *The value $P_n(z)$ can be obtained recursively in n-steps from the coefficients of the polynomial $P_n(x)$.*

- *Furthermore, as a by-product, one can get the value of the derivative of $P_n(x)$ at $x = z$; that is $P'_n(z)$.* Then, the scheme can be incorporated in the Newton Method to compute a zero of $P_n(x)$.

(*Note that the Newton Method requires evaluation of $P_n(x)$ and its derivative at successive iterations*).

1.9.1 Horner's Method

Let $P_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$ and let $x = z$ be given.

We need to compute $P_n(z)$ and $P'_n(z)$.

Let's write $P_n(x)$ as:

$$P_n(x) = (x - z)\ Q_{n-1}\ (x) + b_o,$$

where $Q_{n-1}(x) = b_nx^{n-1} + b_{n-1}\ x^{n-2} + \cdots + b_2x + b_1$.

Thus, $P_n(z) = b_0$.

Let's see how to compute b_0 recursively by knowing the coefficients of $P_n(x)$

From above, we have

$$a_0 + a_1x + a_2x^2 + \cdots + a_nx^n = (x - z)(b_nx^{n-1} + b_{n-1}\ x^{n-2} + \cdots + b_1) + b_o$$

Comparing the coefficients of like powers of x from both sides, we obtain

$$\begin{aligned} b_n &= a_n \\ b_{n-1} - b_nz &= a_{n-1} \\ \Rightarrow b_{n-1} &= a_{n-1} + b_nz \\ &\vdots \end{aligned}$$

and so on.

In general, $b_k - b_{k+1}z = a_k$

$\Rightarrow b_k = a_k + b_{k+1}z.\ k = n - 1, n - 2, \ldots, 1, 0.$

Thus, knowing the coefficients $a_n, a_{n-1}, \ldots, a_1, a_0$ of $P_n(x)$, the coefficients $b_n, b_{n-1}, \ldots, b_1$ of $Q_{n-1}(x)$ can be computed recursively starting with $b_n = a_n$, as shown above. That is,

$$b_k = a_k + b_{k+1}z, \quad k = n - 1, \quad k = 2, \ldots, 1, 0.$$

Again, note that $P_n(z) = b_0$. So, $P_n(z) = b_0 = a_0 + b_1\,z$.

That is, if we know the coefficients of a polynomial, we can compute the value of the polynomial at a given point out of these coefficients recursively as shown above.

Algorithm 1.23 (Evaluating Polynomial: $P_n(x) = a_0 + a_1 x_p + a_2 x^2 + \ldots + a_n x^n$ **at** $x = z$**).**

Input: (i) $a_0, a_1, ..., a_n$ - The coefficients of the polynomial $P_n(x)$.

(ii) z - The number at which $P_n(x)$ has to be evaluated.

Output: $b_0 = P_n(z)$.

```
Step 1. Set $b_n = a_n$.
Step 2. For $k = n-1, n-2, \ldots, 1, 0$ do.
            Compute $b_k = a_k + b_{k+1} z$
Step 3. Set $P_n(z) = b_0$
End
```

It is interesting to note that as a by-product of above, we also obtain $P_n'(z)$, as the following computations show.

Computing $P_n'(z)$

$P_n(x) = (x-z)Q_{n-1}(x) + b_0$

Thus, $P_n'(x) = Q_{n-1}(x) + (x-z)Q_{n-1}'(x)$.

So, $P_n'(z) = Q_{n-1}(z)$.

Write $Q_{n-1}(x) = (x-z)R_{n-2}(x) + c_1$

Substituting $x = z$, we get $\boxed{Q_{n-1}(z) = c_1.}$

To compute c_1 from the coefficients of $Q_{n-1}(x)$ we proceed in the same way as before.

$$\text{Let } R_{n-2}(x) = c_n x^{n-2} + c_{n-1} x^{n-3} + \cdots + c_3 x + c_2$$

Then from above we have

$$b_n x^{n-1} + b_{n-1} x^{n-2} + \cdots + b_2 x + b_1 = (x-z)(c_n x^{n-2} + c_{n-1} x^{n-3} + \cdots + c_3 x + c_2) + c_1.$$

Equating the coefficients of like powers on both sides, we obtain

$$\begin{aligned} b_n &= c_n \\ \Rightarrow c_n &= b_n \\ b_{n-1} &= c_{n-1} - zc_n \\ \Rightarrow c_{n-1} &= b_{n-1} + zc_n \\ &\vdots \\ b_1 &= c_1 - zc_2 \\ \Rightarrow c_1 &= b_1 + zc_2 \end{aligned}$$

Since b's have already been computed (by ***Algorithm 1.23****), we can obtain c's from b's.*

Then, we have the following scheme to compute $P'_n(z)$:

Computing $P'_n(z)$

Inputs: (i) $b_1, ..., b_n$ - The coefficients of the polynomial $Q_{n-1}(x)$ obtained from the previous algorithm

(ii) z - The number at which $P'_3(x)$ has to be evaluated.

Output: $c_1 = Q_{n-1}(z)$.

```
Step 1. Set $c_n = b_n$,
Step 2. For $k = n-1, n-2, ...2, 1$ do
            Compute $c_k = b_k + c_{k+1}\ z$,
        End
Step 3. Set $P'_n(z) = c_1$.
```

Example 1.24

Given $P_3(x) = x^3 - 7x^2 + 6x + 5$. Compute $P_3(2)$ and $P'_3(2)$ using Horner's scheme.

Input Data:

(i) The coefficients of the polynomial:	$a_0 = 5, a_1 = 6, a_2 = -7, a_3 = 1$
(ii) The point at which the polynomial and its derivative need to be evaluated:	$z = 2$
(iii) The degree of the polynomial:	n=3

Formula to be used:

$$\begin{cases} \text{(i) Computing } b_k\text{'s from } a_k\text{'s: } b_k = a_k + b_{k+1}z, k = n-1, n-2, \ldots, 0; P_3(z) = b_0. \\ \text{(ii) Computing } c_k\text{'s from } b_k\text{'s: } c_k = b_k + c_{k+1}z, k = n-1, n-2, \ldots, 1; P_3'(z) = c_1 \end{cases}$$

Solution.

- Compute $P_3(2)$.

  Generate the sequence $\{b_k\}$ from $\{a_k\}$:

$$\begin{aligned} b_3 &= a_3 = 1 \\ b_2 &= a_2 + b_3 z = -7 + 2 = -5 \\ b_1 &= a_1 + b_2 z = 6 - 10 = -4 \\ b_0 &= a_0 + b_1 z = 5 - 8 = -3. \end{aligned}$$

$\boxed{\text{So, } P_3(2) = b_0 = -3}$

- Compute $P_3'(2)$.

  Generate the sequence $\{c_k\}$ from $\{b_k\}$.

$$\begin{aligned} c_3 &= b_3 = 1 \\ c_2 &= b_2 + c_3 z = -5 + 2 = -3 \\ c_1 &= b_1 + c_2 z = -4 - 6 = -10. \end{aligned}$$

$\boxed{\text{So, } P_3'(2) = -10.}$

1.9.2 The Newton Method with Horner's Scheme

We now describe the Newton Method for a polynomial using Horner's scheme. Recall that the Newton iteration for finding a root of $f(x) = 0$ is:

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

In case $f(x)$ is a polynomial $P_n(x)$ of degree n: $f(x) = P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$, the above iteration becomes:

$$x_{k+1} = x_k - \frac{P_n(x_k)}{P_n'(x_k)}$$

If the sequence $\{b_k\}$ and $\{c_k\}$ are generated using Horner's Scheme, at each iteration we then have

$$\boxed{x_{k+1} = x_k - \frac{b_0}{c_1}}$$

(Note that at the iteration $k, P_n(x_k) = b_0$ and $P_n'(x_k) = c_1$).

Algorithm 1.25 (The Newton Method with Horner's Scheme).

Input: $a_0, a_1, \cdots, a_n$ - The coefficients of $P_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$.
x_0 - The initial approximation
ϵ - The tolerance
N - The maximum number of iterations to be performed.

Output: An approximation of the root $x = \xi$ or a failure message.

```
For $k = 0, 1, 2, \cdots$, do

Step 1. $z = x_k$, $b_n \equiv a_n$, $c_n \equiv b_n$

Step 2. Computing $b_1$ and $c_1$
        For $j = n-1, n-2, \cdots, 1$ do
            $b_j \equiv a_j + z\ b_{j+1}$
            $c_j \equiv b_j + z\ c_{j+1}$
        End.
Step 3. $b_0 = a_0 + z\ b_1$ ] Computing $P_n(x_k)$ and $P_n'(x_k)$

Step 4. $x_{k+1} = x_k - b_0/c_1$

Step 5. If $|x_{k+1} - x_k| < \epsilon$ or $k > N$, stop.

End
```

Note:

1. The outer-loop corresponds to the Newton Method

2. The inner-loop and the statement $b_0 = a_0 + zb_1$ correspond to the evaluation of $P_n(x)$ and its derivative $P_n'(x)$ at successive approximations.

Example 1.26

Find a root of $P_3(x) = x^3 - 7x^2 + 6x + 5 = 0$, using the Newton method and Horner's scheme, choosing $x_0 = 2$ (Note that $P_3(x) = 0$ has a root between 1.5 and 2).

Input Data:

$$\begin{cases} \text{(i) The coefficients of } P_3(x)\text{: } a_0 = 5, a_1 = 6, a_2 = -7, \text{ and } a_3 = 1. \\ \text{(ii) The degree of the polynomial: } n = 3. \\ \text{(iii) Initial approximation: } x_0 = 2. \end{cases}$$

Solution.

Iteration 1. $(k = 0)$. Compute x_1 from x_0:

$$x_1 = x_0 - \frac{b_0}{c_1} = 2 - \frac{3}{10} = \frac{17}{10} = 1.7$$

Note that $b_0 = P_3(x_0)$ and $c_1 = P_3'(x_0)$ have already been computed in Example 1.23.

Iteration 2. $(k = 1)$. Compute x_2 from x_1:

Step 1. $z = x_1 = 1.7,\ b_3 = 1,\ c_3 = b_3 = 1$

Step 2.

$$\begin{cases} b_2 &= a_2 + z\, b_3 = -7 + 1.7 = -5.3 \\ c_2 &= b_2 + z\, c_3 = -5.3 + 1.7 = -3.6 \\ b_1 &= a_1 + z\, b_2 = 6 + 1.7(-5.3) = -3.0100 \\ c_1 &= b_1 + z\, c_2 = -3.01 + 1.7(-3.6) = -9.130 \text{ (Value of } P_3'(x_1)). \end{cases}$$

Step 3. $b_0 = a_0 + z\, b_1 = 5 + 1.7(-3.0100) = -0.1170$ (Value of $P_3(x_1)$).

Step 4. $x_2 = x_1 - \frac{b_0}{c_1} = 1.7 - \frac{0.1170}{9.130} = 1.6872$

(The **exact root**, correct up to four decimal places *is 1.6872*).

1.10 The Müller Method: Finding a Pair of Complex Roots.

The Müller Method is an extension of the Secant Method. It is conveniently used to approximate a **complex root** of a real equation, using real starting approximations or just a real root.

Note that none of the methods: the Newton, the Secant, the Fixed Point iteration Methods, etc. can produce an approximation of a complex root starting from real approximations.

The Müller Method is capable of doing so.

Given two initial approximations, x_0 and x_1 the successive approximations are computed as follows:

- **Approximating x_2 from x_0 and x_1:** The approximation x_2 is computed as the x-intercept of the secant line passing through $(x_0, f(x_0))$ and $(x_1, f(x_1))$.

- **Approximating x_3 from x_0, x_1, and x_2:** Having $(x_0, f(x_0))$, $(x_1, f(x_1))$, and $(x_2, f(x_2))$ at hand, the next approximation x_3 is computed as the x-intercept of the parabola passing through these three points. Since a parabola can intersect the x-axis at two points, the one closest to x_2 is taken as x_3.

- Thus, computing x_3 from the starting approximations x_0, x_1, and x_2 by Muller's Method consists of the following two steps:

  Step 1. Find the parabola P passing through $(x_0,\ f(x_0))$, $(x_1,\ f(x_1))$, and $(x_2,\ f(x_2))$.

  Step 2. Solve the quadratic equation representing the parabola P and take x_3 as the root closest to x_2.

  (*Note that even when x_0, x_1, and x_2 are real, x_3 can be complex.*)

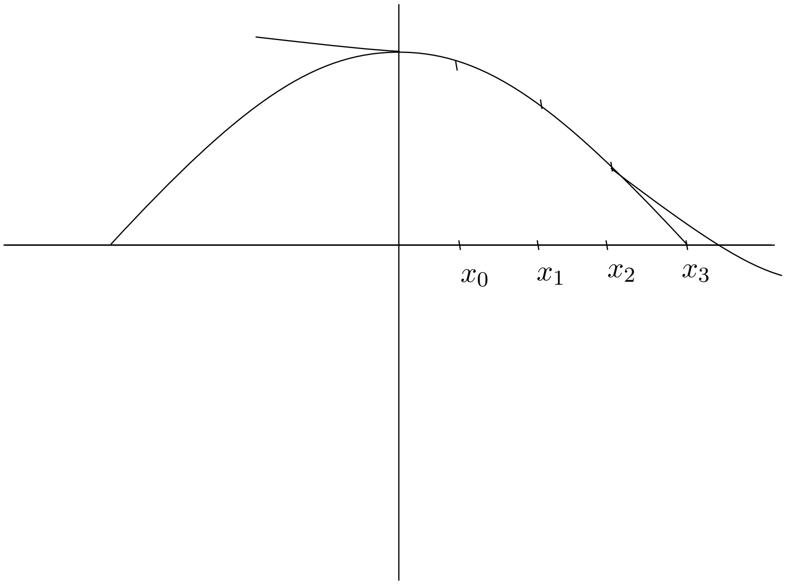
- **(Computing x_4 and successive approximations.)**

The process, of course, can be continued to obtain other successive approximations.

x_4 is obtained by solving the quadratic equation representing the parabola P passing through $(x_1,\ f(x_1))$, $(x_2,\ f(x_2))$, and $(x_3,\ f(x_3))$, and taking x_4 as the one that is closest to x_3; and so on.

Remarks: Note that it is advisable to relabel the approximation at the beginning of each iteration so that we need to store only 3 approximations per iteration.

Figure 1.7: Geometric Interpretation of the Müller Method



Based on the above idea, we now give the details of how to obtain x_3 starting from $x_0,\ x_1$, and x_2.

Let the equation of the parabola P be

$$P(x) = a(x - x_2)^2 + b(x - x_2) + c.$$

That is, $P(x_2) = c$, and a and b are computed as follows:

Since $P(x)$ passes through $(x_0,\ f(x_0))$, $(x_1,\ f(x_1))$, and $(x_2,\ (f(x_2))$, we have

$$\begin{aligned} P(x_0) &= f(x_0) = a(x_0 - x_2)^2 + b(x_0 - x_2) + c \\ P(x_1) &= f(x_1) = a(x_1 - x_2)^2 + b(x_1 - x_2) + c \\ P(x_2) &= f(x_2) = c. \end{aligned}$$

Knowing $c = f(x_2)$, we can now obtain a and b by solving the first two equations:

$$\left.\begin{aligned} a(x_0 - x_2)^2 + b(x_0 - x_2) &= f(x_0) - c \\ a(x_1 - x_2)^2 + b(x_1 - x_2) &= f(x_1) - c. \end{aligned}\right]$$

Or

$$\begin{pmatrix} (x_0 - x_2)^2 & (x_0 - x_2) \\ (x_1 - x_2)^2 & (x_1 - x_2) \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} f(x_0) - c \\ f(x_1) - c \end{pmatrix}.$$

Knowing a and b from above, we can now obtain x_3 by solving $P(x_3) = 0$.

$$P(x_3) = a(x_3 - x_2)^2 + b(x_3 - x_2) + c = 0.$$

To avoid **catastrophic cancellation** (**See Chapter 3**), instead of using the traditional formula, we use the following mathematical equivalent formula for solving the quadratic equation $ax^2 + bx + c = 0$.

Solving the Quadratic Equation

$$\begin{aligned} ax^2 + bx + c &= 0 \\ x &= \frac{-2c}{b \pm \sqrt{b^2 - 4ac}} \end{aligned}$$

The sign in the denominator is chosen so that it is the largest in magnitude, guaranteeing that x_3 is closest to x_2.

To solve $P(x_3) = 0$ using the above formula, we get

$$x_3 - x_2 = \frac{-2c}{b \pm \sqrt{b^2 - 4ac}} \text{ or } x_3 = x_2 + \frac{-2c}{b \pm \sqrt{b^2 - 4ac}}.$$

Note: One can also give on explicit expressions for a and b as follows:

$$b = \frac{(x_0 - x_2)^2\,[f(x_1) - f(x_2)] - (x_1 - x_2)^2\,[f(x_0) - f(x_2)]}{(x_0 - x_2)(x_1 - x_2)(x_0 - x_1)}$$

$$a = \frac{(x_1 - x_2)\,[f(x_0) - f(x_2)] - (x_0 - x_2)\,[f(x_1) - f(x_2)]}{(x_0 - x_2)(x_1 - x_2)(x_0 - x_1)}.$$

However, for computational efficiency and accuracy, it is easier to solve the following 2 × 2 linear system (displayed in Step2-2 of Algorithm 1.27 below) to obtain a and b, once $c = f(x_2)$ is computed.

Algorithm 1.27 (The Müller Method for Approximations of a Pair of Complex Roots).

Input: $f(x)$ - The given function
x_0, x_1, x_2 - Initial approximations
ϵ - Tolerance
N - The maximum number of iterations

Output: An approximation of the root $x = \xi$

Step 1. Evaluate the functional values at the initial approximations: $f(x_0)$, $f(x_1)$, and $f(x_2)$.

Step 2. Compute the next approximation x_3 as follows:
2.1 Set $c = f(x_2)$.
2.2 Solve the 2 × 2 linear system for a and b:

$$\begin{pmatrix} (x_0 - x_2)^2 & (x_0 - x_2) \\ (x_1 - x_2)^2 & (x_1 - x_2) \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} f(x_0) - c \\ f(x_1) - c \end{pmatrix}.$$

2.3 Compute $x_3 = x_2 + \dfrac{-2c}{b \pm \sqrt{b^2 - 4ac}}$ choosing the sign + or − so that the denominator is the largest in magnitude.

Step 3. Test if $|x_3 - x_2| < \epsilon$ or if the number of iterations exceeds N. If so, stop. Otherwise go to Step 4.

Step 4. Relabel x_3 as x_2, x_2 as x_1, and x_1 as x_0: $\begin{cases} x_2 \equiv x_3 \\ x_1 \equiv x_2 \\ x_0 \equiv x_1 \end{cases}$

Return to **Step 1**.

Example 1.28

(**Approximating a Pair of Complex Roots by Muller's Method**) Approximate a pair of complex roots of $x^3 - 2x^2 - 5 = 0$ starting with the initial approximations as $x_0 = -1, x_1 = 0$, and $x_2 = 1$. (Note the roots of $f(x) = \{2.6906, -0.3453 \pm 1.31876\}$.)

Input Data: $\begin{cases} \text{The function: } f(x) = x^3 - 2x^2 - 5, \\ \text{The initial approximations: } x_0 = -1, x_1 = 0, x_2 = 1 \end{cases}$

Formula to be used: $x_3 = x_2 + \frac{-2c}{b \pm \sqrt{b^2 - 4ac}}$,

Choosing + or − to obtain the largest (in magnitude) denominator.

Solution.

Iteration 1.

Step 1. Compute the functional values. $f(x_0) = -8$, $f(x_1) = -5$, $f(x_2) = -6$.

Step 2. Compute the next approximation x_3 as follows:

2.1 Set c: $c = f(x_2) = -6$

2.2 Compute a and b: a and b are obtained by solving the 2×2 linear system:

$$\left.\begin{array}{r} 4a - 2b = -2 \\ a - b = 1 \end{array}\right] \Rightarrow \quad a = -2, \ b = -3$$

2.3 Compute x_3:

$$x_3 = x_2 - \frac{2c}{b - \sqrt{b^2 - 4ac}} = x_2 + \frac{12}{-3 - \sqrt{9 - 48}} = 0.25 + 1.5612i.$$

Iteration 2. Relabel x_3 as x_2, x_2 as x_1, and x_1 as x_0:

$$\begin{cases} x_2 \equiv x_3 = 0.25 + 1.5612i \\ x_1 \equiv x_2 = 1 \\ x_0 \equiv x_1 = 0 \end{cases}$$

Step 1. Compute the new functional values: $f(x_0) = -5$, $f(x_1) = -6$, and $f(x_2) = -2.0625 - 5.0741i$

Step 2. Compute the next approximation x_3:

2.1 Set $c = f(x_2)$.

2.2 Obtain a and b by solving the 2×2 linear system:

$$\begin{pmatrix} x_2^2 & -x_2 \\ (x_1 - x_2)^2 & (x_1 - x_2) \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} f(x_0) - c \\ f(x_1) - c \end{pmatrix},$$

which gives $a = -0.75 + 1.5612i$, $b = -5.4997 - 3.1224i$.

2.3 $x_3 = x_2 - \dfrac{2c}{b - \sqrt{b^2 - 4ac}} = x_2 - (0.8388 + 0.3702i) = -0.5888 + 1.1910i$

Iteration 3. Relabel x_3 as x_2, x_2 as x_1, and x_1 as x_0:

$$\begin{cases} x_2 \equiv x_3 = -0.5888 + 1.1910i \\ x_1 \equiv x_2 = 0.25 + 1.5612i \\ x_0 \equiv x_1 = 1 \end{cases}$$

Step 1. Compute the functional values: $f(x_0) = -6$, $f(x_1) = -2.0627 - 5.073i$, $f(x_2) = -0.5549 + 2.3542i$

Step 2. Compute the new approximation x_3:

2.1 Set $c = f(x_2)$.

2.2 Obtain a and b by solving the 2×2 system:

$$\begin{pmatrix} (x_0 - x_2)^2 & (x_0 - x_2) \\ (x_1 - x_2)^2 & (x_1 - x_2) \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} f(x_0) - c \\ f(x_1) - c \end{pmatrix},$$

which gives $a = -1.3888 + 2.7522i$ and $b = -2.6339 - 8.5606i$.

3 $x_3 = x_2 - \dfrac{2c}{b - \sqrt{b^2 - 4ac}} = -0.3664 + 1.3508i$

Iteration 4. $x_3 = -0.3451 + 1.3180i$ (Details computations are similar and not shown)

Iteration 5. $x_3 = -0.3453 + 1.3187i$

Iteration 6. $x_3 = 0.3453 + 1.3187i$

Test for Convergence: $\left|\dfrac{x_3 - x_2}{x_2}\right| < 5.1393 \times 10^{-4}$, we stop here and accept the latest value of x_3 as an approximation to the root.

Thus, an approximate pair of complex roots is:

$$\alpha_1 = -0.3455 + 1.3187i, \alpha_2 = -0.3455 - 1.3187i.$$

Verify: How close is this root to the actual root? $(x-\alpha_1)(x-\alpha_2)(x-2.6906) = x^3-1.996x^2-5$

Example 1.29

(Approximating a Real Root by Muller's Method)

Approximate a real root of $f(x) = x^3 - 7x^2 + 6x + 5$ using $x_0 = 0, x_1 = 1$, and $x_2 = 2$ as initial approximations. (All three roots are real: 5.8210, 1.6872, -0.5090. We will try to approximate the root 1.6872).

Input Data: $\begin{cases} f(x) = x^3 - 7x^2 + 6x + 5 \\ \text{Initial approximations: } x_0 = 0, x_1 = 1, \text{ and } x_2 = 2. \end{cases}$

Solution.

Iteration 1. Compute x_3 from x_0, x_1, and x_2:

Step 1. The functional values at the initial approximations: $f(x_0) = 5, f(x_1) = 5, f(x_2) = -3$.

Step 2. Compute the next approximation x_3:

2.1 Compute $c = f(x_2) = -3$.

2.2 Compute a and b by solving the 2×2 linear system:

$$\begin{pmatrix} (x_0 - x_2)^2 & (x_0 - x_2) \\ (x_1 - x_2)^2 & (x_1 - x_2) \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 5 - c \\ 5 - c \end{pmatrix}$$

or

$$\begin{pmatrix} 4 & -2 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 8 \\ 8 \end{pmatrix} \Rightarrow \begin{cases} a = -4 \\ b = -12 \end{cases}$$

2.3 Compute $x_3 = x_2 - \dfrac{2c}{b - \sqrt{b^2 - 4ac}}$

$x_3 = x_2 - 0.2753 = 1.7247$ (choosing the negative sign in the determinant).

Iteration 2. Relabel x_3 as x_2, x_2 as x_1, and x_1 as x_0: $\begin{cases} x_2 \equiv x_3 = 1.7247, \\ x_1 \equiv x_2 = 2, \\ x_0 \equiv x_1 = 1 \end{cases}$

Step 1. Compute the functional values at the relabeled approximations: $f(x_0) = 5$, $f(x_1) = -3$, $f(x_2) = -0.3441$

Step 2. Compute the new approximation x_3:

2.1 Set $c = f(x_2) = -0.3441$.

2.2 Then a and b are obtained by solving the 2×2 linear system:

$$\begin{pmatrix} 0.5253 & -0.7247 \\ 0.0758 & 0.2753 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 5 - c \\ 3 - c \end{pmatrix}, \Rightarrow \begin{cases} a = -2.2752 \\ b = -9.0227 \end{cases}$$

2.3 $x_3 = x_2 - \dfrac{2c}{b - \sqrt{b^2 - 4ac}} = 1.7247 - 0.0385 = 1.6862$

Comparison of Different Methods for the Root-Finding Problem

Method	Work Required	Convergence	Remarks
Bisection	Two function evaluations per iteration	Always converges (but slow)	(i) The initial interval containing the root must be found first. (ii) A bracketing method - every iteration brackets the root.
Fixed-Point	One function evaluation per iteration	Rate of convergence is linear	Finding the right iteration function $x = g(x)$ is a challenging task
Newton	Two function evaluations per iteration−function value and value of the derivative at the iterate	Quadratic convergence at a simple root. Convergence is linear for a multiple root.	(i) Initial approximation x_0 must be chosen carefully-if it is far from the root, the method will diverge. (ii) For some function, the derivative is difficult to compute, if not impossible
Secant	Two function evaluations per iteration−however, only one of the two is a new evaluation	Superlinear	Needs two initial approximations
Müller	Three function evaluations and solution of a 2 × 2 linear system per iteration	Convergence is almost quadratic near the root	Does not require derivative and can compute complex roots with starting with real approximations.

Concluding Remarks

- If the derivation of $f(x)$ is computable, then the Newton method is an excellent root-finding mehtod. The initial approximation x_0 can be computed by drawing a graph of $f(x)$ or by running a far iteration of the bisection method and taking the approximate value obtained by the bisection method as the initial approximation for Newton-Ralphson iteration.

- If the derivative is not computable, use the secant method, instead.

1.11 Sensitivity of the Roots of a Polynomial

The roots of a polynomial can be very sensitive to small perturbations of the coefficients of the polynomial. For example, take

$$p(x) = x^2 - 2x + 1.$$

The roots are $x_1 = 1$, $x_2 = 1$. Now perturb only the coefficient -2 to -1.9999, leaving the other two coefficients unchanged. The roots of the peturbed polynomial are: $1 \pm .01i$. Thus, *both the roots become complex.*

One might think that this happened because of the roots of $p(x)$ are multiple roots. It is true that the multiple roots are prone to perturbations, but *the roots of a polynomial with well-separated roots can be very sensitive too!*

The well-known example of this is the celebrated **Wilkinson polynomial**:

$$p(x) = (x-1)(x-2)...(x-20).$$

The roots of this polynomial are 1, 2, ..., 20 and thus very well-separated. But, a small perturbation of the coefficient x^{19} which is $-210 + 2^{-23}$ will change the roots completely. Some of them will even become complex. This is illustrated in the following graph. *Note that some roots are more sensitive than the others.*

1.12 Deflation Technique

Deflation is a technique to compute the other roots of $f(x) = 0$, once an approximate real root or a pair of complex roots are computed. Then if $x = \alpha$ is an approximate root of $P(x) = 0$, where $P(x)$ is a polynomial of degree n, then we can write

$$P(x) = (x - \alpha)Q_{n-1}(x)$$

where $Q_{n-1}(x)$ is a polynomial of degree $n - 1$.

Similarly, if $x = \alpha \pm i\beta$ is an approximate pair of complex conjugate roots, then

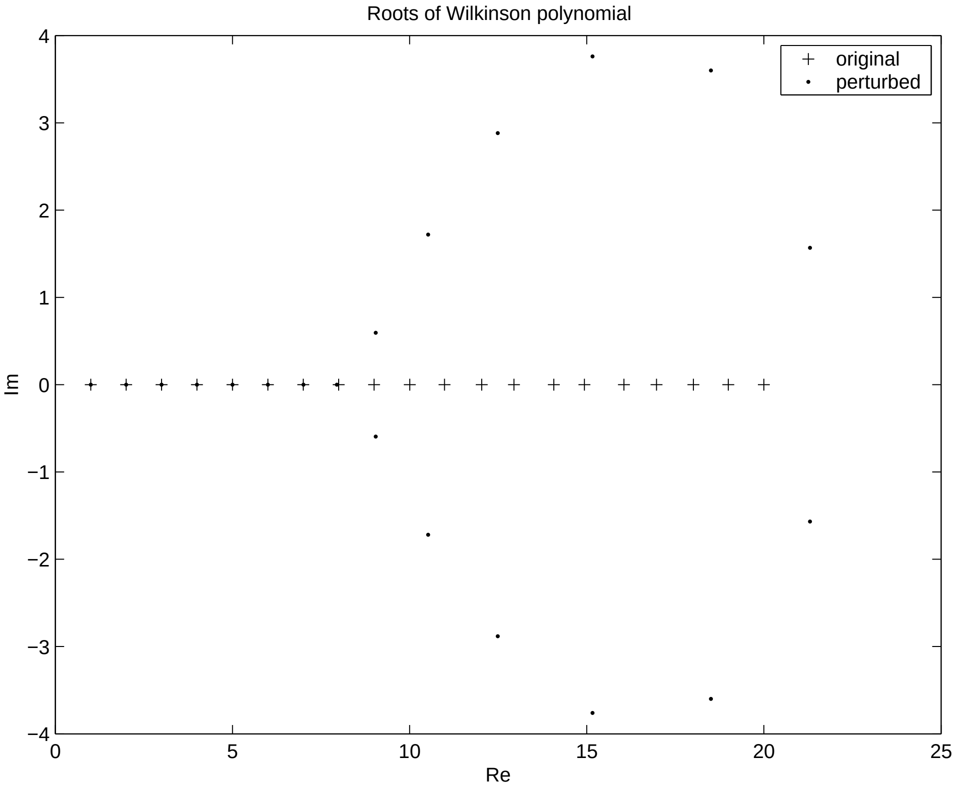
$$P(x) = (x - \alpha - i\beta)(x - \alpha + i\beta)Q_{n-2}(x)$$

where $Q_{n-2}(x)$ is a polynomial of degree $n - 2$.

Moreover, the zeros of $Q_{n-1}(x)$ in the first case and those of $Q_{n-2}(x)$ is the second case are also the zeros of $P(x)$.

The coefficients of $Q_{n-1}(x)$ or $Q_{n-2}(x)$ can be generated by using **synthetic division** as in Horner's method, and then any of the root-finding procedures obtained before can be applied to

Figure 1.8: Roots of the Wilkinson Polynomial (Original and Perturbed)



compute a root or a pair of roots of $Q_{n-2}(x) = 0$ or $Q_{n-1}(x)$. The procedure can be continued until all the roots are found.

Example 1.30

Find the roots of $f(x) = x^4 + 2x^3 + 3x^2 + 2x + 2 = 0$.

It is easy to see that $x = \pm i$ is a pair of complex conjugate roots of $f(x) = 0$. Then we can write

$$f(x) = (x + i)(x - i)(b_3x^2 + b_2x + b_1)$$

or

$$x^4 + 2x^3 + 3x^2 + 2x + 2 = (x^2 + 1)(b_3x^2 + b_2x + b_1).$$

Equating coefficients of x^4 , x^3, and x^2 on both sides, we obtain

$$b_3 = 1, \quad b_2 = 2, \quad b_1 + b_3 = 3,$$

giving $b_3 = 1$, $b_2 = 2$, $b_1 = 2$.

So, $(x^4 + 2x^3 + 3x^2 + 2x + 2) = (x^2 + 1)(x^2 + 2x + 2)$.

Thus the roots of the polynomial equation: $x^4 + 2x^3 + 2x^2 + 2x + 2$ are given by the roots of (i) $x^2 + 1 = 0$ and (ii) $x^2 + 2x + 2$.

The zeros of the deflated polynomial $Q_2(x) = x^2 + 2x + 2$ can now be found by using any of the methods described earlier.

1.13 MATLAB Functions for Root-Finding

MATLAB built-in functions **fzero**, **poly**, **polyval**, and **roots** can be used in the context of zero-finding of a function.

In a MATLAB setting, type **help** follwed by these function names, to know the details of the uses of these functions. Here are some basic uses of these functions.

A. Function **fzero**: Find the zero of $f(x)$ near x_0. *No derivative is required.*

Usage:

$y =$ **fzero** (function, x_0).

function – Given function (specified either as a MATLAB *function file* or as an *anonymous function*).

$\mathbf{x_0}$ – A point near the **zero** of $f(x)$, if it is a scalar.

x_0 – If given as a 2-vector, $\begin{bmatrix} a \\ b \end{bmatrix}$ then it is assumed that $f(x)$ has a zero in the interval $[a, b]$.

• The function **fzero** can also be used with **options** to display the optimal requirements such as values of x and $f(x)$ at each iteration (possibly with some specified tolerance). For example, the following **usages**:

$$\begin{cases} \text{options} & = \textbf{optimset} \text{ ('Display', 'iter')} \\ z & = \textbf{fzero} \text{ (function, } x_0\text{, options)} \end{cases}$$

or

$z =$ **fzero** (function, x, optimset ('Display', 'iter'). will display the value of x and the corresponding value of $f(x)$ at each iteration performed (see below **Exercise 1.31**). Other values of **options** are also possible.

Example 1.31 (i) Find the zero of $f(x) = e^x - 1$ near $x = 0.1$.

(ii) Find the zero of $f(x) = e^x - 1$ in $[-1, 2]$ and display the values of x and $f(x)$ at each iteration.

Solution (i)

$$z = \textbf{ fzero } (@(z)\,\exp(x) - 1, 0.1)$$
$$z = 0.$$

Solution (ii)

$$z = \text{ fzero } (@(x)\,\exp(x) - 1, [-1, 2]^T)$$
$$z = 2.1579e^{-17}$$

- $z =$ **fzero** $(@(x)\,\exp(x) - 1, [-1, 2]$ opimset ('display', 'iter')).

Funct Count	x	$f(x)$	Procedure
2	-1	0.632121	initial
3	-0.729908	-0.518047	interpolation
4	0.404072	0.48811	interpolation
5	-0.151887	-0.140914	interpolation
6	-0.0292917	-0.0288669	interpolation
7	0.000564678	0.000546857	interpolation
8	$-8.30977e^{-06}$	$-8.30974e^{-06}$	interpolation
9	$-2.34595e^{-09}$	$-2.345959e^{-09}$	interpolation
10	$2.15787e^{-17}$	0	interpolation

Zero found in the interval $[-1, 2]$. $z = 2.1579e^{-17}$.

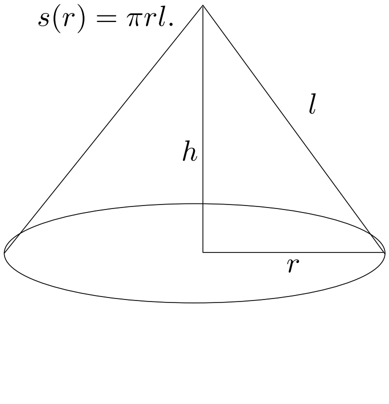
Remarks: As shown from the above output that the function **fzero** uses hybrid methods for root-finding. Initially it uses a method such as the secant method which does not require derivatie computation and then switches to **inverse interpolation**.

Example 1.32

It is well-known that the lateral surface area of a cone is:

$$s(r) = \pi r l.$$

$r =$	radius
$l =$	length
$h =$	height
Since	$l = \sqrt{h^2 + r^2}$



We see that $s(r)$ is a function of f if h is known:

$$s(r) = \pi r \sqrt{h^2 + r^2}.$$

Find the radius of the cone whose lateral surface area is 750m^2 and the height is 9m, using MATLAB function **fzero**.

Solution

We need to find a zero r of the function:

$$f(r) = 750 - \pi r\sqrt{r^2+9}$$

Choose $r_0 = 5$.

$$r = \textbf{ fzero } (@(r)750 - \pi * r * sqrt + (r^2+9), 5)$$
$$r = 15.3060.$$

So, the radius of the cone should be 15.3060m.

B. Functions: **poly**, **polyval**, and **roots**.

These functions relate to finding the roots of a **polynomial** equation.

poly - converts **roots** to a polynomial **equation**.

- poly (A), when A is an $n \times n$ matrix, gives a row vector with $(n+1)$ elements which are the coefficients of the characteristic polynomial

$$P(\lambda) = det(\lambda I - A)$$

- poly (v), when v is a vector, gives a vector whose elements are the coefficients of the polynomial whose roots are the elements of v.

roots - Find the polynomial roots.

• roots (c) computes the roots of the polynomial whose coefficients are the elements of the vector c:

$$P(x) = c_1x^n + c_2x^{n-1} + \cdots + c_nx + c_{x+1}$$

polyval - Evaluates the value of a polynomial at a given point.

• $y =$ polyval (p, z) returns the value of a polynomial $P(x)$ at $x = z$.

where

$p =$ the vector containing the coefficients $(c_1, c_2, \ldots, c_{n+1})$ of the polynomial $P(x)$ as above.

$z =$ a number, a vector, or a matrix.

If it is a vector or matrix, $P(x)$ is evaluated at all points in z.

Example 1.33

Using the functions **poly**, **roots**, and **polyval**,

(i) Find the coefficients of the polynomial equation $P_3(x)$ whose roots are: $1, 1, 2$, and 3.

(ii) Compute now, numerically, the roots and check the accuracy of each computed root.

(iii) Perturb the coefficient of x^2 of $P_4(x)$ by 10^{-4} and recompute the roots.

Solution (i).

$$\begin{aligned} v &= [1, 1, 2, 3]^T \\ c &= \textbf{poly} v \end{aligned}$$

The vector c contains the coefficients of the 4-th degree polynomial $P_4(x)$ with the roots: 1,1,2, and 3.

$$\begin{aligned} &c = [1, -7, 17, -17, 6]^T \\ &P_4(x) = x^4 - 7x^3 + 17x^2 - 17x + 6 \end{aligned}$$

Solution (ii).

$z =$ **roots** (c) gives the computed zeros of the polynomial $P_4(x)$.

$$z = \begin{pmatrix} 3.0000 \\ 2.0000 \\ 1.0000 \\ 1.0000 \end{pmatrix}$$

Solution (iii).

Perturbed polynomial $\tilde{P}_4(x)$ is given by

$$\tilde{P}_4(x) = x^4 - (7 + 10^{-4})x^3 + 17x^2 - 17x + 6$$

The vector cc containing the coefficients of $\tilde{P}_4(x)$ is given by:

$$\begin{aligned} cc &= [\qquad]^T \\ zz &= \textbf{ roots } (cc). \end{aligned}$$

The vector zz contains the zeros of the perturbed polynomial $\tilde{P}_4(x)$:

$$vv = \textbf{ polyval } (zz).$$

The vector vv contains the values of $\tilde{P}_4(x)$ evaluated at the computed roots of $\tilde{P}_4(x)$:

$$vv = [\qquad]^T$$

• Function **eig**.

• **eig**(A) is used to compute the eigenvalues of a matrix A. It can be used to find all the roots of a polynomial, as the following discussions show.

It is easy to verify that the roots of the polynomial

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1}x^{n-1} + x^n$$

are the eigenvalues of the **companion matrix**.

$$C_n = \begin{pmatrix} 0 & 0 & \cdots & 0 & -a_0 \\ 1 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & \cdots & 0 & -a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -a_{n-1} \end{pmatrix}$$

Thus,

eig(c_n) will compute all the roots of $P_n(x)$. In fact, the MATLAB built-in function **roots**(c) is based on this idea.

Example 1.34

Find the roots of $P_3(x) = x^3 - 3x + 1$.

Solution.

The matrix $c_3 = \begin{pmatrix} 0 & 0-1 & \\ 1 & 0 & 0 \\ 0 & 1 & 3 \end{pmatrix}$.

eig(C_3) gives the roots of $P_3(x)$:

$$\mathbf{eig}(C_3) = (-1.8794, 1.5321, 0.3473).$$

Thus the roots of $P_3(x)$ are: $\begin{cases} -1.8784 \\ 1.5321 \\ 0.3473 \end{cases}$

Verify: We will now show that these roots are the same as obtained by the function **roots**:

$$c = [1, 0, 3, 1]^T$$
$$\mathbf{roots}\ (c).$$

$$\textbf{roots}(c) = \begin{cases} -1.8784 \\ 1.5321 \\ 0.3473 \end{cases} \quad \text{same as obtained by } \textbf{eig}(C_3).$$

Remarks: It is not recommended that the eigenvalues of an arbitrary matrix A are computed by transforming A first into companion form C and then finding its zeros. *Because, the transforming matrix can be highly ill-conditioned* (Please see **Chapter 8**).

EXERCISES

0. (Conceptual) Answer **True** or **False**. Give reasons for your answer.

   (i) Bisection method is guaranteed to find a root in the interval in which the root lies.

   (ii) Bisection method converges quite fast.

   (iii) It is possible to theoretically find the minimum number of iterations needed by bisection method prior to starting the iteration.

   (iv) Bisection method brackets the root in each iteration.

   (v) Newton's method always converges.

   (vi) Quadratic convergence means that about two digits of accuracy are obtained at each iteration.

   (vii) Fixed point iteration always converges, but the rate of convergence is quite slow.

   (viii) In the presence of a multiple root, Newton's method does not converge at all.

   (ix) The rate of convergence of Newton's method, when it converges, is better than the secant method.

   (x) Both the secant and false-positive methods are bracket methods.

   (xi) Horner's scheme efficiently computes the value of a polynomial and its derivative at a given point.

   (xii) Muller's method can compute a pair of complex conjugate roots using only real approximations.

   (xiii) Only the multiple roots of a polynomial equation are sensitive to small perturbations of the data.

   (xiv) The fixed-point iteration theorem (Theorem ??) gives both necessary and sufficient conditions of the convergence of the fixed-point iteration to a root.

   (xv) A computational advantage of the secant method is that the derivative of $f(x)$ is not required.

   (xvi) Newton's method is a special case of the fixed-point iteration scheme.

   (xvii) Both Newton and second methods require two initial approximations to start the iteration.

   (xviii) The convergence rate of the standard Newton's method for a multiple root is only linear.

   (xix) A small functional value at $x = \alpha$ guarantees that it is a root $f(x) = 0$.

1. Use the bisection method to approximate $\sqrt{3}$ with an error tolerance $\epsilon = 10^{-3}$ in the intervals: $[1, 2]$, $[1, 3]$, and $[0, 2]$.

2. (Computational) For each of the following functions, do the following:

   (a) Test if there is a zero of $f(x)$ in the indicated interval.

   (b) Find the minimum number of iterations, N, needed to achieve an accuracy of $\epsilon = 10^{-3}$ by using the bisection method.

   (c) Using the bisection method, perform N number of iterations and present the results both in tabular and graphical forms.

   (i) $f(x) = x \sin x - 1$ in $[0, 2]$

   (ii) $f(x) = x^3 - 1$ in $[0, 2]$

   (iii) $f(x) = x^2 - 4 \sin x$ in $[1, 3]$

   (iv) $f(x) = x^3 - 7x + 6$ in $[0, 1.5]$

   (v) $f(x) = \cos x - \sqrt{x}$ in $[0, 1]$

   (vi) $f(x) = x - \tan x$ in $[4, 4.5]$

   (vii) $f(x) = e^{-x} - x$ in $[0, 1]$

   (viii) $f(x) = e^x - 1 - x - \frac{x^2}{2}$ in $[-1, 1]$

3.

4. (Computational) Construct a simple example to show that a small functional value of $f(x_k)$ at an iteration k does not necessarily mean that x_k is near the root $x = \xi$ of $f(x)$.

5. (Analytical)

   (a) Prove that if $g(x)$ is continuously differentiable in some open interval containing the fixed point ξ, and if $|g'(\xi) < 1$, then there exists a number $\epsilon > 0$ such that the iteration

   $$x_{k+1} = g(x_k)$$

   converges whenever x_0 is chosen such that $|x_0 - \xi| \le \epsilon$.

   (b) Using the above result, find the iteration function $g(x)$ and an interval $[a, b]$ for

   $$f(x) = x - \tan x$$

   such that the iteration always converges. Find the solution with an accuracy of 10^{-4}.

   (c) Find a zero of the function $f(x) = e^{-x} - \sin x$ in $[0.5, 0.7]$, by choosing $g(x) = x + f(x)$ and using the result in (a).

6. (Computational) For each of the following functions, find an *interval* and an *iteration function* $x = g(x)$ so that the fixed-point iteration:

   $$x_{k+1} = g(x_k)$$

   will converge for any choice of x_0 in the interval. Then apply two iterations to approximate a zero in that interval choosing a suitable initial approximation x_0.

(a) $f(x) = x - \cos x$

(b) $f(x) = \frac{3}{x^2} - x - 2$

(c) $f(x) = x - e^{-x}$

(d) $f(x) = x^2 - 4\sin(x)$

(e) $f(x) = \frac{1}{1+e^{-x^2}} + \cos x$

(f) $f(x) = \frac{x^2}{4} - 1$

(g) $f(x) = x - 2\sin x$

(h) $f(x) = x^3 - 7x + 6$

(i) $f(x) = 1 - \tan \frac{x}{4}$

7. (Computational) Use fixed-point iteration to compute $\sqrt{2}$ with an accuracy of 10^{-4}.

8. (Computational) Study the convergence of all fixed-point iterations of the function $f(x) = x^3 - 7x + 6$ at $x = 2$ with all possible iteration functions $g(x)$. Represent these fixed point iteration schemes graphically.

9. (Computational)

   (a) Sketch a graph of $f(x) = x + \ln x$ and show that $f(x)$ has one and only one zero ξ in $0 < x < \infty$, but that the related fixed point iteration $x_{i+1} = g(x_i)$, x_0 given, $x_0 \neq \xi$, with $g(x) = -\ln x$, does not converge to ξ even if x_0 is arbitrarily close to ξ.

   (b) How can one rewrite the iteration equation so as to obtain convergence? Show for this iteration the uniqueness of the fixed point ξ. Give its rate of convergence.

10. (Computational) The iteration $x_{n+1} = 2 - (1+c)x_n + cx_n^3$ will converge to $\xi = 1$ for some values of c (provided x_0 is chosen sufficiently close to ξ). Find the values of c for which this is true. For what value of c will the convergence be quadratic?

11. (Computational) Apply Newton's method, the secant method and the method of false position to determine a zero with an accuracy of 10^{-4} to each of the functions in Exercise 6.

12. (Computational) Write down Newton's iterations for the following functions:

    (a) $x^3 - \cos(x^2 + 1) = 0$

    (b) $2\sin \pi x - \sqrt{x^2 - x + 1} = 0$

13. (Analytical and Computational) Develop Newton's method to compute the following:

    (a) $\ln(a)$ (natural log of a) $(a > 0)$

    (b) $arc\cos a$ and $arc\sin a$

(c) e^a

Do three iterations for each problem by choosing a suitable function and a suitable x_0. Compare your results with those obtained by MATLAB built-in function **fzero**.

14. (Computational) Explain what happens when Newton's method is applied to find a root of $x^3 - 3x + 6 = 0$, starting with $x_0 = 1$.

15. (Computational) Construct your own example to demonstrate the fact that Newton's method may diverge if x_0 is not properly chosen.

16. (Computational) Show that the Newton sequence of iterations for the function $f(x) = x^3 - 2x + 2$ with $x_0 = 0$ will oscillate between 0 and 1. Give a graphical representation of this fact.

    Find now a positive root of $f(x)$ using Newton's method by choosing a suitable x_0.

17. (Computational) Study the convergence behavior of Newton's method to approximate the zero $x = 2$ of the following two polynomials:

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

    starting with the same initial approximation $x_0 = 1.9$.

18. (Computational) Write down the Newton-sequence of iterations to find a minimum or a maximum of a function $f(x)$.

    Apply it to find the minimum value of $f(x) = x^2 + \sin x$

19. (Computational and Analytical) Derive an algorithm to compute $\sqrt[n]{A}$ based on Newton's method and apply your algorithm to compute $\sqrt[5]{30}$, by choosing a suitable initial approximation.

20. (Analytical and Computational) Repeat the previous problem with the secant method and compare the speed of convergence.

21. (Analytical) Prove that the rate of convergence of Newton's method for a multiple root is only linear, not quadratic. Construct a small example to support this statement.

22. (Analytical) Prove that the convergence rate of the secant method is superlinear.

23. (Computational) Apply the secant method to $f(x) = x^3 - 3x + 2$ to approximate the root $x = 1$ with $x_0 = 0$ and $x_1 = 2$. Show that the convergence factor is approximately 1.6.

24. (Computational) The polynomial $P_3(x) = x^3 - 2x^2 + x - 2$ has a pair of zeros $x = +i$. Approximate them using Muller's method.

25. (Computational) The polynomial $P_8(x) = x^8 + 2x^7 + x^6 + 2x^5 + 5x^3 + 7x^2 + 5x + 7$ has a pair of complex-conjugate zeros $x = \pm i$. Approximate them using Muller's method (*Do three iterations.*) Choose $x_0 = 0$, $x_1 = 0.1$, $x_2 = 0.5$.

26. (Computational) The polynomial $P_6(x) = 3x^6 - 7x^3 - 8x^2 + 12x + 3 = 0$ has a double root at $x = -1$. Approximate this root first by using standard Newton's method and then by two modified Newton's methods, starting with $x_0 = -0.5$.

    Use Horner's method to find the polynomial value and its derivative at each iteration. Present your results in tabular form. (*Do three iterations.*)

27. (Applied) Consider the Kepler equation of motion

    $$M = E - e \sin E$$

    relating the **mean anomaly** M to the **eccentric anomaly** E of an elliptic orbit with eccentricity e. To find E, we need to solve the nonlinear equation:

    $$f(E) = M + e \sin E - E = 0$$

    (a) Show that the fixed point iteration

    $$E = g(E) = M + e \sin E$$

    converges.

    (b) Given $e = 0.0167$ (**Earth's eccentricity**)and $M = 1$ (in radians) compute E using

    i. bisection method,

    ii. fixed-point iteration $E = g(E)$ as above,

    iii. Newton's method; **and**

    iv. secant method.

    To choose appropriate initial approximations or an interval containing the root (required for the bisection method), plot the function $f(E)$ and then determine graphically where it crosses the E-axis.

    (c) Referring to the case study in Section 1.1, find the position of the planet Earth for $t = 300$ days with $a = 150 \times 10^6$km.

28. (Applied) Consider the **van der Walls equation** of state:

    $$\left(P + \frac{a}{V^2}\right)(V - b) = RT$$

    Use Newton's method to compute the specific volume V of *carbon dioxide* at a temperature of $T = 300^\circ$K, given $P = 1$ atom, $R = 0.082054$ J (kg$^\circ$K), $a = 3.592$ Pa$\cdot$m^6/kg^2, $b = 0.04267$m^2/kg.

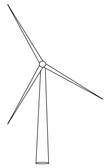
    Obtain the initial approximation V_0 from the ideal gas law: $PV = RT$.

29. (Applied) Repeat the last problem for *oxygen* for which $a = 1.360$ and $b = 0.03183$.

30. (Applied) **Windmill Electric Power**

    It is becoming increasingly common to use wind turbines to generate electric power. The energy output of the power generated by a windmill depends upon the blade's diameter and velocity of the wind. A good estimate of the energy output is given by the following formula:

    $$EO = 0.01328D^2V^3$$

    where EO = Energy Output, D = Diameter of the windmill blade (m), V = Velocity of the wind in m/s.

    Use Newton's method to determine what should be the diameter of the windmill blade if one wishes to generate 500 watts of electric power when the wind speed is 10mph.

31. (Applied) A spherical tank of radius $r = \Lambda m$ contains a liquid of volume $V = 0.5m^3$. What is the depth, h, of the liquid?

    Use the following formula: $V = \frac{\pi h^2}{3}(3r - h)$ to solve.

32. (Applied) If the displacement of a body at time t is given by

    $$x(t) = \cos 10t - \frac{1}{2}\sin 10t$$

    At what time $t(s)$ will the displacement be 20 meters?

33. (Applied) Suppose that the displacement of a body at time t under a damping oscillation is given by

    $$x(t) = e^{-t}\left(A\cos\frac{\sqrt{12}}{2}t + B\sin\frac{\sqrt{12}}{2}t\right)$$

    Determine the time required for the body to come to rest ($x(t) = 0$) using the initial conditions: $x(0) = 1, x'(0) = 5$.

    (**Hints:** First, find A and B using the initial conditions. Then solve the above equation for t setting $x(t) = 0$.)

34. (Analytical and Computational) **(Haley's Method)** The following iteration for finding a root of $f(x) = 0$:

    $$x_{i+1} = x_i - \frac{2f(x_i)f'(x_i)}{2[f(x_i)^2 - f(x_i)f''(x_i)]}$$

    is known as **Haley's iteration**.

(i) Prove that if $f(x)$ is three times continuously differentiable and $x = \xi$ is a root of $f(x) = 0$; but not of its derivative, then Haley's iteration converges to the zero if the initial approximation x_0 is sufficiently close to ξ, and that the convergence is cubic; that is,

$$\frac{e_{i+1}}{e_i^3} \leq C, \quad \text{for some } C > 0$$

(ii) Using Haley's iteration, compute a positive zero of $f(x) = x^3 - 3x + 1$ starting with $x_0 = 0.5$ with an error tolerance of $\epsilon = 10^{-4}$.

MATLAB EXERCISES

M1.1 Write MATLAB functions in the following formats to implement the methods: **Bisection**, **Fixed-Point Iteration**, **Newton**, **Secant**, and **Muller**.

- function [xval, funval, iter, error] = **bisect**(fnc, intv, tol, maxitr)
- function [xval, funval, iter, erro] = **newton**(fnc, deriv, x0, to, maxitr)
- function [xval, funval, iter, error] = **fixed-point**(fnc, x0, tol, maxitr)
- function [xval,funval,iter, error] = **secant**(fnc, x0, x1, tol, maxitr)
- function [xval, funval, iter, error] = **muller**(fnc, x0, x1, x2, tol, maxitr)

intv	= Interval containing the root
xval	= Value of the root at the present iteration
iter	= Iteration number
funval	= Value of the function at the present iteration
error	= Error message in case of nonconvergence
fnc	= The function input as a string
deriv	= Derivative of function also given as a string
xo, x1, x2	= Initial approximations
tol	= Error tolerance
maxitr	= Maximum number of iterations to be performed
fncval	= Function value of the root at each iteration

Presentation of Results: Present your results both in tabular and graphic forms. The tables and graphs should display the values of the function and the approximate root at each iteration. Results from all the methods need to be presented in the same table and same graph.

Table : Comparison of Different Root-Finding Methods

Method	Iteration	x-Value	Function Value

Data: Implement the methods using the functions given in Exercises 6(a), 6(b), 6(d), 6(e), 6(h), and 6(i).

M1.2 Apply now the built-in MATLAB function **fzero** to the function of Exercise 2 and present your results along with those of other methods in the same table and same graph using the format of M1.1.

M1.3 Write a MATLAB program, using the built-in MATLAB functions, **polyval**, and **polyder** to find a zero of a polynomial, based on Newton's Method with Horner's scheme. Implement the program using

$$P(x) = x^5 - x^4 - 5x^3 - x^2 + 4x + 3$$

Present results of each iteration both in tabular and graphic forms.

M1.4 Choosing the values of a, b, R and P in appropriate units, find v at temperatures $T = 100°\text{K}$, $T = 250°\text{K}$, $T = 450°\text{K}$ by solving the *Van der Waals equation* state using each of the methods developed in M1.1. Present the results in tabular form.

M1.5 Apply the MATLAB program **newton** to compute (a) $\sqrt[5]{1000}$ and (b) $\frac{1}{12000}$, (c) e^5, and (d) $\ln(10)$, choosing a suitable initial approximation in each case. For each problem, present your results of iterations in tabular forms.

M1.6 Consider the following equation:

$$A = \frac{R}{i}\left[1 = (1+i)^{-n}\right]$$

where A = Amount of money borrowed; i = Interest rate per time period; R = Amount of each payment; n = Number of payments (all payments are of equal value).

If the amount of the loan is \$20,000 and the payment amount per month is \$450.00 and the loan is paid off in 5 years, what is the monthly interest rate? (Solve the problem using the **newton** program.)

M1.7 The following formula gives the upward velocity v of a rocket:

$$v = u \ln \frac{m}{m - qt} - gt$$

where u = The velocity of the fuel expelled relative to the rocket; m = Initial mass of the rocket (at $t = 0$); g = Gravity's acceleration; t = Time.

Use any bracket method to find the time taken by the rocket to attain the velocity of 500 m/s, given

$u = 950$ m/s, $m = 2 \times 10^5$ kg, $q = 3,000$ kg/s, and $g = 9.8$ m/s^2.

Choose $10 \le t \le 25$.

M1.8 Suppose the concentration c of a chemical in a mixed reactor is given by:

$$c = c_{in}(1 - e^{-0.05t}) + c_0 e^{-0.05t}$$

where c_{in} = Inflow concentration; and c_0 = Original concentration.

Find the time t required for c to be 90% of c_{in}, if $c_0 = 5$, and $c_{in} = 15$.

M1.9 Display the results of the following MATLAB operations:

$f =$ **inline** ('$3/\hat{x}^2 - x - 2$')
fzero = ($f, 1.5$, optimset('display', 'iter'))

Interpret. What do these results mean?

M1.10 (**Study of the Root-Sensitivities of Polynomials**)

Using MATLAB functions, **roots**, **poly**, and **polyval**, do the following for each of the following polynomials.

A. Compute all the zeros of the unperturbed polynomial.

B. Put perturbations of orders $10^{-3}, 10^{-4}$, and 10^{-5} to the second highest coefficient and recompute the zeros of each of the perturbed polynomial.

C. Present your results in tabular form that should contain (i) all the zeros of the original, (ii) zeros of the perturbed polynomials with each perturbation and, (iii) functional values at each computed zero.

D. Plot the zeros of the original and perturbed polynomials on the same graph for each polynomial.

E. Write your observations in each case.

Data:

(i) The polynomial $P_{20}(x)$ with the zeros $1, 2, 3, 4, \ldots, 20$.

(ii) The polynomial $P_3(x)$ with all the zeros equal to 1.

(iii) The polynomial $P_4(x)$ with two doable zeros equal to 1 and two other zeros as 10 and 20.

M1.11 Modify the program **Muller** to accommodate **deflation** for computing more than one root.

The polynomial

$$P_7(x) = x^7 - 41x^6 + 690x^5 - 6130x^4 + 30689x^3 - 84969x^2 + 1116460x - 56700$$

has zeros: $10, 9, 7, 6, 5, 3,$ and 1.

Compute approximations to all of them using Muller's method and the technique of deflation.