# LECTURE NOTES

# 1 PART I: Numerical Methods for the Root-Finding Problem

## 1.1 Introduction

In this lecture, we will discuss numerical methods for the **Root-Finding Problem**. 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 $x = \xi$ such that $f(\xi) = 0$.

The number $\xi$ is called a **root** of $f(x) = 0$ or a **zero** of the function $f(x)$. The function of $f(x)$ may be algebraic or trigonometric functions. **The well-known examples of algebraic functions are polynomials.** A polynomial of degree $n$ is written as $P_n(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n$. The numbers $a_{i,i=0,...,n}$ are called the coefficients of the polynomial. If $a_0 = 1$, the polynomial $P_n(x)$ is called the **monic polynomial.**

**Applications of the Root-Finding Problem**

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. For other applications of the root-finding problem, see Section.

**(INCOMPLETE)**

## 1.2 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 $c = \dfrac{a+b}{2}$ 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 warranted by the desired accuracy.

To implement the idea, we need to know which of the two intervals in each iteration contains the root of $f(x) = 0$. The **Intermediate Mean-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., ).

**Intermediate Mean-Value Theorem**

Let $f(x)$ be a continuous function defined on $[a, b]$, such that

f(a) and $f(b)$ are of opposite signs (i.e., $f(a)f(b) < 0$).

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

**Algorithm 1.1 The Bisection Method for Rooting-Finding**

**Inputs:** (i) $f(x)$ - The given function

(ii) $a_0, b_0$ - 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 $k = 0, 1, 2, \cdots$, do until satisfied:

- Compute $c_k = \dfrac{a_k + b_k}{2}$.
- Test 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 1.1** $f(x) = x^3 - 6x^2 + 11x - 6$.

*Let* $a_0 = 2.5,\ b_0 = 4$. *Then* $f(a_0)f(b_0) < 0$. *Then there is a root in* $[2.5, 4]$.

**Iteration 1.** $k = 0$ :

$c_0 = \dfrac{a_0 + b_0}{2} = \dfrac{4 + 2 \cdot 5}{2} = \dfrac{6 \cdot 5}{2} = 3 \cdot 25.$

Since $f(c_0)f(a_0) = f(3 \cdot 25)f(2 \cdot 5) < 0$, set $b_1 = c_0,\ a_1 = a_0$.

**Iteration 2.** $k = 1$ :

$c_1 = \dfrac{3 \cdot 25 + 2 \cdot 5}{2} = 2 \cdot 8750.$

Since $f(c_1)f(a_1) > 0$, set $a_2 = 2 \cdot 875,\ b_2 = b_1$

**Iteration 3.** $k = 2$ :

$c_2 = \dfrac{a_2 + b_2}{2} = \dfrac{2 \cdot 875 + 3 \cdot 250}{2} = 3 \cdot 0625$

Since $f(c_2)f(a_2) = f(3 \cdot 0625)f(2 \cdot 875) < 0$, set $b_3 = c_2,\ a_3 = a_2$.

**Iteration 4.** $k = 3$

$c_3 = \dfrac{a_3 + b_3}{2} = \dfrac{2 \cdot 875 + 3 \cdot 0625}{2} = 2.9688.$

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

**Notes:**

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.

**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 $\epsilon$ be the tolerance; that is, we would like to obtain the root with an error of at most of $\epsilon$. Then

1. Accept $x = c_k$ as a root of $f(x) = 0$ if any of the following criteria is satisfied:
2. $|f(c_k)| \leq \epsilon$ (*The functional value is less than or equal to the tolerance*).
3. $\frac{|c_{k-1} - c_k|}{|c_k|} \leq \epsilon$ (*The relative change is less than or equal to the tolerance*).
4. $\frac{(b-a)}{2^k} \leq \epsilon$ (*The length of the interval after k iterations is less than or equal to the tolerance*).
5. *The number of iterations k is greater than or equal to a predetermined number, say N.*

**Comments:**

1. The 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).**
2. The Criterion 3 is based on the fact that after $k$ steps, the root will be computed with error at most $\left(\frac{b_0 - a_0}{2^k}\right)$.

**Number of Iterations Needed in the Bisection Method to Achieve 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.

The 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 $\epsilon$, using the Criterion 3, we must have $\frac{b_0 - a_0}{2^N} \leq \epsilon$.

That is, $2^{-N}(b_0 - a_0) \leq \epsilon$

or $2^{-N} \leq \dfrac{\epsilon}{(b_0 - a_0)}$

or $-N \log_{10} 2 \leq \log_{10}\left(\dfrac{\epsilon}{b_0 - a_0}\right)$

or $N \log_{10} 2 \geq -\log_{10}\left(\dfrac{\epsilon}{b_0 - a_0}\right)$.

or $N \geq \dfrac{-\log_{10}\left(\frac{\epsilon}{b_0 - a_0}\right)}{\log_{10} 2}$.

or $N \geq \dfrac{[\log_{10}(b_0 - a_0) - \log_{10}(\epsilon)]}{\log_{10} 2}$

**Theorem 1.1** *The number of iterations $N$ needed in the bisection method to obtain an accuracy of $\epsilon$ is given by*

$$N \geq \frac{[\log_{10}(b_0 - a_0) - \log_{10}(\epsilon)]}{\log_{10} 2}$$

■

**Example 1.2** *Suppose we would like to determine a priori the minimum number of iterations needed in the bisection algorithm, given $a_0 = 2.5, b_0 = 4$, and $\epsilon = 10^{-3}$. By Theorem 1.1, we have*

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

**Remarks:** 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 Fixed-Point Iteration

A number $\xi$ 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 $\xi$ 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 correspond 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

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

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 $f(x) = 0$, how to write $f(x) = 0$ in the form $x = g(x)$,
> so that starting with any $x_0$ in $[a,\ b]$, the sequence $\{x_k\}$ defined
> by $x_{k+1} = g(x_k)$ is guaranteed to converge?

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 1.1** 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:
$$\begin{aligned} x_1 &= g(x_0) = g(3.5)5.3750 \\ x_2 &= g(x_1) = g(5.3750)40.4434 \\ x_3 &= g(x_2) = g(40.4434)5.6817 \times 10^4 \\ x_4 &= g(x_3) = g(5.6817 \times 10^4)1.8340 \times 10^{14} \end{aligned}$$

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

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

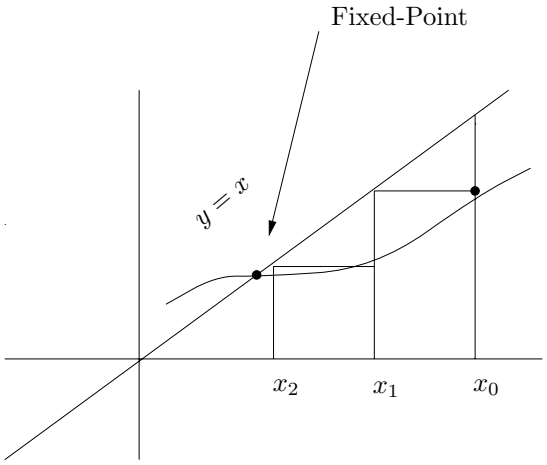



**Figure 1.1: Convergence of the Fixed-Point Iteration**

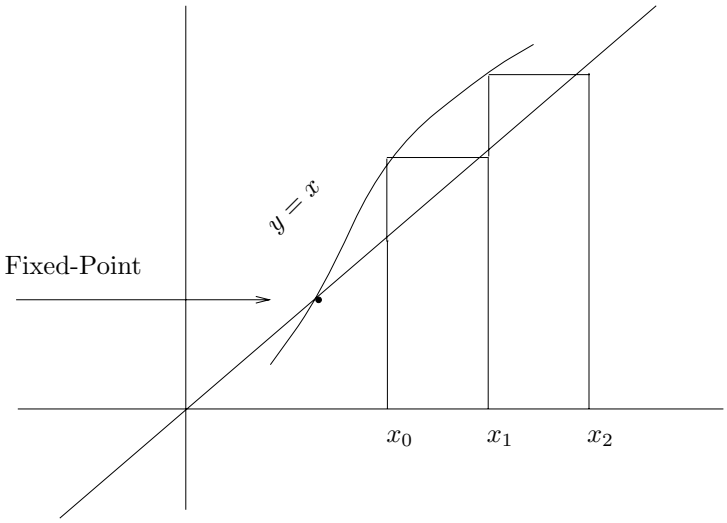



**Figure 1.2: Divergence of the Fixed-Point Iteration**

The following theorem gives a sufficient condition on $g(x)$ which ensures the convergence of the sequence $\{x_k\}$.

**Theorem 1.2 (Fixed-Point Iteration Theorem):** *Let $f(x) = 0$ be written in the form $x = g(x)$. Assume that $g(x)$ has the following properties:*

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

2. *$g'(x)$ exists on $(a,\ b)$ with the property that there exists a positive constant $r < 1$ 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 $\xi$ of $f(x) = 0$.

■

The proof of the Theorem requires the **Mean Value Theorem** of Calculus;

| **The Mean Value Theorem** |
|---|
| Let $f(x)$ be a continuous function on $[a,\ b]$, and differentiable on $(a, b)$. |
| Then there is a number $c$ in $(a, b)$ such that $\dfrac{f(b) - f(a)}{b - a} = f'(c)$. |

**Proof of the Fixed-Proof Theorem:**

The proof comes into three parts: **Existence, Uniqueness, and Convergence**. We first prove that there is a root of $f(x) = 0$ in $[a,\ b]$ and it is unique. Since a fixed point of $x = g(x)$ is a root of $f(x) = 0$, this amounts to proving that there is a fixed point in $[a,\ b]$ and it is unique. We then prove that the sequence $x_{k+1} = g(x_k)$ converges to the root.

**Existence:**

If $a = g(a)$, then $a$ is a fixed point. If $b = g(b)$, then $b$ is a fixed point. If not, **because of the assumption** 1, $g(a) > a$, and $g(b) < b$. Thus, $f(a) = g(a) - a > 0$, and $f(b) = g(b) - b < 0$. Also $f(x) = g(x) - x$ is continuous on $[a, b]$.

Thus, by the **Intermediate Mean Value Theorem**, there is a root of $f(x)$ in $[a,\ b]$. This proves the existence.

**Uniqueness:**

To prove uniqueness, suppose $\xi_1$ and $\xi_2$ are for two fixed points in $[a,\ b]$, and $\xi_1 \neq \xi_2$.

Then by the Mean Value Theorem, there is a number $c$ in $(a, b)$ such that

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

Since $g(\xi_1) = \xi_1$, and $g(\xi_2) = \xi_2$, we have

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

That is, $g'(c) = 1$, which is a contradiction to our assumption 2. Thus, $p$ can not be different from $q$.

**Convergence:**

Let $\xi$ be the root of $f(x) = 0$. Then the **absolute error** at step $k + 1$ is given by $e_{k+1} = |x_{k+1} - \xi|$. *To prove that the sequence convergence, we need to show that* $\lim_{k\to\infty} e_{k+1=0}$.

Now $e_{k+1} = |x_{k+1} - \xi| \ = \ |g(x_k) - g(\xi)|$ (note that $\xi = g(\xi)$).

Thus by the Mean Value Theorem, we have $e_{k+1} = |g'(c)|\ |x_k - \xi| = |g'(c)|e_k$, where $c$ is in $(x_k,\ \xi)$.

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

Similarly, we have $e_k \leq re_{k-1}$. Thus $\boxed{e_{k+1} \leq r^2 e_{k-1}.}$

Continuing in this way, we finally have $\boxed{e_{k+1} \leq r^{k+1} e_0}$, 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 the root $x = \xi$.

**Example 1.3** *Let* $f(x) = x^3 - 6x^2 + 11x - 6 = 0$. *Choose* $a = 2.5,\ b = 4$. *Let's write* $f(x) = 0$ *in the form* $x = g(x)$ *as follows:* $x = \frac{1}{11}(-x^3 + 6x^2 + 6) = g(x)$.

Then

$$g'(x) = \frac{1}{11}(-3x^2 + 12x).$$

It is easy to verify graphically and analytically that $|g'(x)|$ is less than 1 for all $x$ is $(2.5, 4)$, (note that $g'(4) = 0$, and $g'(2.5) = 1.0227$).

So, by the Fixed-Point Theorem, the iteration $x_{k+1} = g(x_t)$ should converge to the root $x = 3$ for any $x_0 \in [2.5, 4]$. This is verified as follows:

$$\begin{array}{lcl} x_0 & = & 3.5 \\ x_1 = g(x_0) & = & 3.3295 \\ x_2 = g(x_1) & = & 3.2368 \\ x_3 = g(x_2) & = & 3.1772 \\ x_4 = g(x_x) & = & 3.1359 \\ x_5 = g(x_4) & = & 3.1059 \\ x_6 = g(x_5) & = & 3.0835 \\ x_7 = g(x_6) & = & 3.0664 \\ x_8 = g(x_7) & = & 3.0531 \\ x_9 = g(x_8) & = & 3.0426 \\ x_{10} = g(x_9) & = & 3.0344 \\ x_{11} = g(x_{10}) & = & 3.0778 \end{array}$$

The sequence is clearly converging to the root $x = 3$. This can be justified from the above theorem.

**Example 1.4** *Find a root of* $f(x) = x - \cos x = 0$ *in* $[0, \frac{\pi}{x}]$.

*Let's write* $x - \cos x = 0$ *as* $x = \cos x = g(x)$. *Then* $|g'(x)| = |\sin x| < 1$ *in* $(0, \frac{\pi}{2})$. *Thus, the fixed-point iteration should converge. Indeed,*

$$\begin{array}{ll} x_0 & = 0 \\ x_1 & = \cos x_o = 1 \\ x_2 & = \cos x_1 = 0.54 \\ x_3 & = \cos x_2 = 0.86 \\ \vdots & \\ x_{17} & = 0.73956 \\ x_{18} & = \cos x_{17} = 0.73956 \end{array}$$

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 1.1** *Let $g(x)$ be continuously differentiable in some open interval containing the fixed point $\xi$, and let $|g'(\xi)| < 1$, then there exists a number $\epsilon > 0$ so that the iteration $x_{k+1} = g(x_k)$ converges whenever $x_0$ is chosen in $|x_0 - \xi| \leq \epsilon$.*

## 1.4 The Newton-Raphson Method

¿From the Fixed-Point Theorem, we have just seen that if $f(x) = 0$ is written in the form $x = g(x)$ and $g(x)$ has the property that it takes every value in $[a,\ b]$ (that is $g(x) \in\ [a,\ b]$ for all $x$ in $[a,\ b]$), and $|g'(x)| \leq k < 1$ for all $x\ \in\ (a,\ b)$, then the fixed point iteration

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

converges to the unique root $x = \xi$ of $f(x) = 0$ in $[a,\ b]$.

Assume further that $f''(x)$ exists and is continuous on $[a,\ b]$ and $\xi$ 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 $\quad g'(x) = 1 - \dfrac{(f'(x))^2 - f(x)f''(x)}{(f'(x))^2} = \dfrac{f(x)f''(x)}{(f'(x))^2}.$

Thus, $g'(\xi) = \dfrac{f(\xi)f''(\xi)}{(f'(\xi))^2} = 0$, 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, according to the corollary above.

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

(**Remark:** In many text books and literature, the Newton-Raphson Method is simply called the **Newton Method**. But, the title "The Newton-Raphson Method" is justified. See an article by [ ] in SIAM Review (1996)).

**Algorithm 1.2** ***The Newton-Raphson Method***

**Inputs:** $f(x)$ - The given function
$x_0$ - The initial approximation
$\epsilon$ - 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 $x_{k+1} = x_k - \dfrac{f(x_k)}{f'(x_k)}$.
- Test for convergence or failure:

$$\left.\begin{array}{ll} \text{If} & |f(x_k)| < \epsilon \\ \text{or} & \dfrac{|x_{k+1} - x_k|}{|x_k|} < \epsilon \end{array}\right] \text{ stopping criteria.}$$

  or $k > N$, Stop.

**End**

**Definition 1.1** *The iterations* $x_{k+1} = x_k - \dfrac{f(x_k)}{f'(x_k)}$ *are called* ***Newton's iterations.***

**Remarks:**

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-Raphson Method to a desired root.

**Some Familiar Computations Using the Newton-Raphson Method**

1. **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-Raphson Method to $f(x) = x^2 - A$.

   Since $f'(x) = 2x$, we have the following Newton iterations to generate $\{x_k\}$:

**Newton-Raphson Iterations for Computing $\sqrt{A}$**

**Input:** $A$ - A positive number
**Output:** An approximation to $\sqrt{A}$.
**Step 1.** Guess 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

$$x_{k+1} = x_k - \frac{x_k^2 - A}{2x_k}$$

End

**Example 1.5** *Let* $A = 2,\ x_0 = 1.5$

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

**Iteration 2.** $x_2 = \dfrac{x_1^2 + A}{2x_1} = 1.4142$

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

2. **Computing the nth Root**

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

Thus Newton's Iterations in this case are:

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

3. **Finding the Reciprocal of a Nonzero Number** $A$

The problem here is to compute $\dfrac{1}{A}$, where $A \neq 0$. Again, the Newton-Raphson Method can be applied with $f(x) = \dfrac{1}{x} - A = 0$. We have $f'(x) = -\dfrac{1}{x^2}$. The sequence $x_{k+1}$ is given by

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

**The Newton-Raphson Iteration for Computing $\dfrac{1}{A}$**

**Input:** $A$ - A nonzero number
**Output:** An approximation to $\frac{1}{A}$.
**Step 1.** Guess 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 1.6** *Let* $A = 3$

*The problem is to find* $\frac{1}{3}$

$f(x) = \frac{1}{x} - 3.$

*Take* $x_0 = 0.25$

*Then* $x_1 = x_0(2 - Ax_0) = 0.3125$

$x_2 = x_1(2 - Ax_1) = 0.3320$

$x_3 = x_2(2 - A_2) = 0.333$

*Note that* 0.333 *is the* 4*-digit approximation of* $\frac{1}{3}$ *(correct up to* 4 *decimal figures).*

**A Geometric Interpretation of the Newton-Raphson 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.

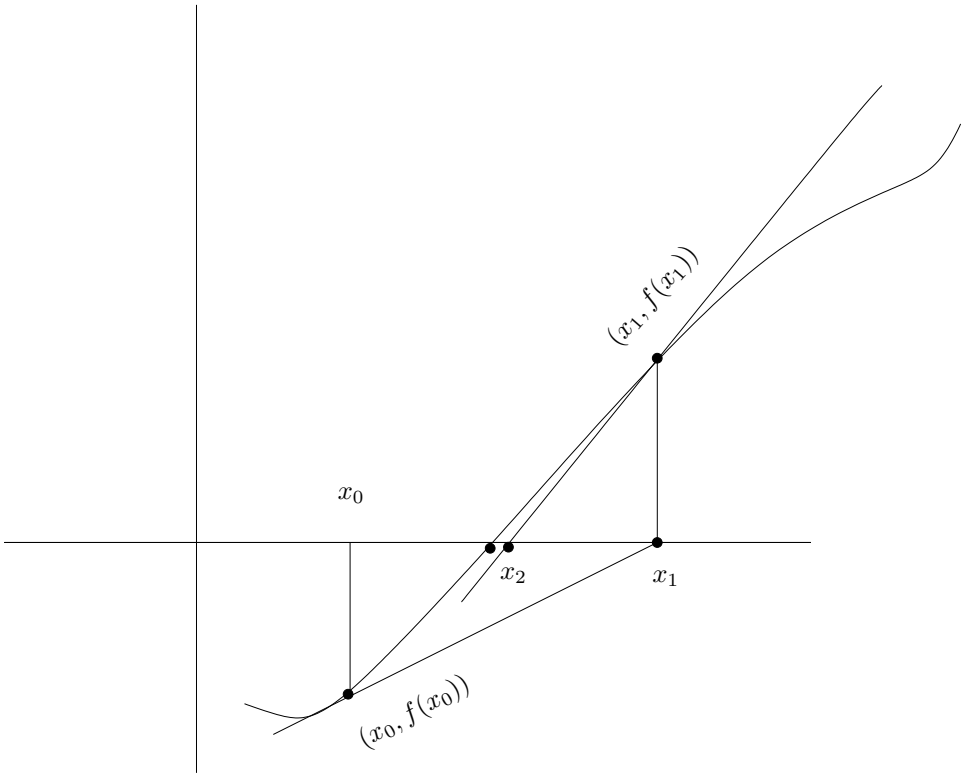



**Figure 1.3: Graphical Representation of the Newton-Raphson Method.**

## 1.5 The Secant Method

**A major disadvantage of the Newton-Raphson Method is the requirement of finding 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:

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_{k-1} - x_{k-2}}.$$

Then, the Newton iterations in this case are:

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

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

**Algorithm 1.3** *The Secant Method*

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

**Output:** An approximation of the exact solution $\xi$ or a message of failure.

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

- Compute $f(x_k)$ and $f(x_{k-1})$.
- Compute the next approximation: $x_{k+1} = x_k - \dfrac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}$
- 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 $x_0,\ x_1$ to start with, whereas the Newton-Raphson method just needs one, namely, $x_0$.

**Example 1.7** $f(x) = x^2 - 2 = 0$.

Let $x_0 = 1.5,\ x_1 = 1$

**Iteration 1**. $k = 1$.

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

**Iteration 2**

$$\begin{aligned} x_3 &= x_2 - \frac{f(x_2)(x_2 - x_1)}{f(x_2) - f(x_1)} \\ &= 1.4 - \frac{(-0.04)(-0.1)}{(-0.04) - (-1)} \\ &= 1.4 + 0.0042 \\ &= 1.4042 \end{aligned}$$

**Iteration 3**

$$\begin{aligned} x_4 &= x_3 - \frac{f(x_3)(x_3 - x_2)}{f(x_3) - f(x_2)} \\ &= 1.4143. \end{aligned}$$

**Iteration 4**

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

**Remarks:** By comparing the results of this example with those obtained by the Newton-Raphson **(Example 1.5)** method, we see that *it took* 5 *iterations by the secant method to obtain a* 4*-digit accuracy of* $\sqrt{2} = 1.4142$*; whereas for the Newton-Raphson method, this accuracy was obtained just after* 2 *iteration.* In general, **the Newton-Raphson 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.

### 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.

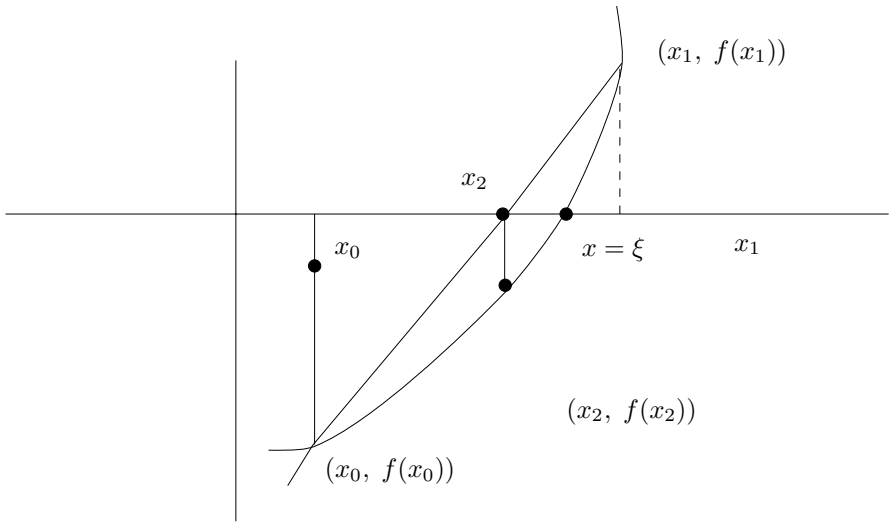



**Figure 1.4: Geometric Interpretation of the Secant Method.**

## 1.6 The Method of False Position (The Regular False 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 interval brackets the root. However, the Newton-Raphson 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 1.8** *Consider solving* $f(x) = \tan(\pi x) - 6 = 0$*, using the* **Secant Method**.

*Take:* $x_0 = 0 \;\; x_1 = 0.48$

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

**Iteration 1**

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

**Iteration 2**

$x_0 = 0.48, \;\; x_1 = 0.181194$

$x_2 = 0.286187$

**Iteration 3**

$x_0 = 0.181194, \;\; x_1 = 0.286187$

$x_2 = 1.091987$. Clearly, *The iterations are diverging.*

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

- 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 $x_2$ is 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, $x_3$ will be computed 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 Regular Falsi**.

**Algorithm 1.4** *The Method of False Position*

**Input:** *(i)* $f(x)$ - *The given function*
*(ii)* $x_0, x_1$ - *The two initial approximations* $f(x_0)f(x_1) < 0$ .
*(iii)* $\epsilon$ - *The tolerance*
*(iv)* $N$ *The maximum number of iterations*

**Output:** *An approximation to the root* $x = \xi$.

**Step 0.** Set $i = 1$.
**Step 1.** Compute $x_{i+1} = x_i - f(x_i)\dfrac{(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}$
If $|\dfrac{x_{i+1} - x_i}{x_i}| < \epsilon$ or $i \geq N$. Stop.
**Step 2.** If $f(x_i).f(x_{i+1}) < 0$, then, set $x_i \equiv x_{i+1}$, $x_{i-1} \equiv x_i$.
Otherwise, set $x_i \equiv x_{i+1}$, $x_{i-1} \equiv x_{i-1}$. Return to step 0. End

**Example 1.9** *Consider again the previous example:*

$$f(x) = \tan(\pi x) - 6 = 0.$$

*Let the interval be* [0, 0.48].

*Then* $\xi = 0.44743154$ *is the root of* $f(x)$ *in* [0, 0.48].

$$x_0 = 0, \;\; x_1 = 0.48, \;\; f(x_0).f(x_1) < 0$$

**Iteration 1**

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

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

Set $x_0 = 0.48, \;\; x_1 = 0.181194$

**Iteration 2**

$x_2 = x_1 - f(x_1)\dfrac{x_1 - x_0}{f(x_1) - f(x_0)} = 0.286187$

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

Set $x_0 = 0.48,\ \ x_1 = 0.286187$

**Iteration 3** $(i = 3)$

$x_2 = 0.348981$

$f(x_2) \cdot f(x_1) > 0$

Set $x_0 = 0.48,\ x_1 = 0.348981$

**Iteration 4**

$x_2 = 0.387051$

**Iteration 5**

$x_2 = 0.410304$

## 1.7 Convergence Analysis of the Iterative Methods

We have seen that the Bisection method always converges, and each of the methods: the Newton-Raphson Method, the Secant method, and the method of False Position, converges under certain conditions. The question now arises: **when a method converges, how fast it converges?** In other words, what is the rate of convergence? To this end, we first define:

**Definition 1.2 (Rate of Convergence of an Iterative Method).** *Suppose that the sequence* $\{x_k\}$ *converges to* $\xi$*. Then the sequence* $\{x_k\}$ *is said to converge to* $\xi$ *with the order of convergence* $\alpha$ *if there exists a positive constant* $p$ *such that*

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

*Thus if* $\alpha = 1$*, the convergence is* **linear**. *If* $\alpha = 2$*, the convergence is* **quadratic***; and so on. The number* $\alpha$ *is called the* ***convergence factor***.

*Using the above definition, we will now show that the rate of convergence for the Fixed Point Iteration method is usually linear but that of the Newton-Raphson method is quadratic, and that of the Secant method is superlinear.*

**Rate of Convergence of the Fixed-Point Iteration Method**

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

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

where $c$ is a number between $x_k$ and $\xi$.

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 $(x_k, \xi)$, 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**

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

**converges linearly**.

**Rate of Convergence of the Newton-Raphson Method**

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

**Taylor's Theorem of order $n$**

Suppose that the function $f(x)$ possesses continuous derivatives of order up to $(n+1)$ in the interval $[a,\ b]$, and $p$ is a point in this interval. 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 $\eta_k$ lies between $x$ and $\xi$.

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

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

or $\quad g(x_k) = g(\xi) + \dfrac{(x_k-\xi)^2}{2!}\ g''(\eta_k).$

or $\quad g(x_k) - g(\xi) = \dfrac{(x_k-\xi)^2}{2!}\ g''(\eta_k).$

Since $g(x_k) = x_{k+1}$, and $g(\xi) = \xi$, this give

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

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

or $\dfrac{e_{k+1}}{e_k^2} = \dfrac{|g''(\eta_k)|}{2}$.

Since $\eta_k$ lies between $x$ and $\xi$ for every $k$, it follows that

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

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

This proves

**The Newton-Raphson Method Converges Quadratically.**

**The quadratic convergence roughly means that the accuracy gets doubled at each iteration.**

**The Rate of Convergence for the Secant Method.**

It can be shown [**Exercise**] that if the Secant method converges, then the convergence factor $\alpha$ is approximately 1.6 for sufficiently large $k$. This is said as "**The rate of convergence of the Secant method is superlinear**".

## 1.8 A Modified Newton-Raphson Method for Multiple Roots

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

The question, therefore, is:

**How can the Newton-Raphson method be modified for a multiple root so that the quadratic convergence can be retained?**

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

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

where $f^{(m-1)}(\xi)$ denotes the $(m-1)$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-Raphson method.

**The Newton-Raphson Iterations for Multiple Roots:**
**Method I**

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

Since in practice, the multiplicity $m$ is not known apriori, another **useful modification is to apply the Newton-Raphson iteration to the function:**

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

in place of $f(x)$.

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

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

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

we have

**The Newton-Raphson 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:**

$f(x) = e^x - x - 1$

$f'(x) = e^x - 1$

$f''(x) = e^x$

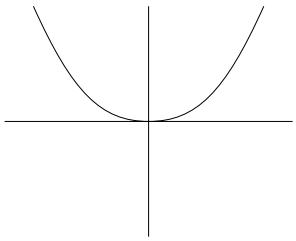


**Figure 1.5: The Graph of $e^x - x - 1$.**

There are two roots at $x = 0$ (See the graph above!). So, $m = 2$.

**Method I**

$x_{i+1} = x_i - \dfrac{2f(x_i)}{f'(x_i)}$

$x_0 = 0.5$

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

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

**Method II**

$f(x) = e^x - x - 1$.

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

$x_0 = 0.5$

**Iteration 1.** $x_1 = -0.0493 = -4.93 \times 10^{-2}$

**Iteration 2.** $x_2 = -3.9848 \times 10^{-4}$

## 1.9 The Newton-Raphson 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 $x_k$.

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-Raphson Method to compute a zero of $P_n(x)$.

(Note that the Newton-Raphson Method requires evaluation of $P_n(x)$ and its derivate at successive iteration).

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

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

Thus, knowing the coefficients $a_n, a_{n-1}, ..., a_1, a_0$ of $P_n(x)$, the coefficients $b_n, b_{n-1}, ..., b_1$ of $Q_{n-1}(x)$ can be computed recursively starting with $b_n = a_n$, as shown above. That is, $b_k = a_k + b_{k+1}z, k = n-1, k = 2, \ldots, 1, 0$.

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 of a given point out of these coefficients recursively as shown above.*

| **Algorithm 1.5 Evaluating Polynomial:** $P_n(x) = a_0 + a_1x_p + a_2x^2 + \ldots + a_nx^n$ **at** $x = z$. | |
|---|---|
| **Inputs:** | (1) $a_0, a_1, ..., a_n$ of $P_n(x)$-The coefficients of the polynomial $P_n(x)$. |
| | (2) $z$-The number at which $P_n(x)$ has to be evaluated. |
| **Outputs:** | $b_0 = P_n(z)$. |
| **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*:

**Finding $P_n'(x)$**

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

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

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

This means that knowing the coefficients of $Q_{n-1}(x)$ we can compute $P_n'(z)$ recursively out of those coefficients. The coefficients of the polynomial $Q_{n-1}(z)$ can be obtained exactly in the same way as above, by replacing $a_k,\ k = n, \cdots 1$ by $b_k, k = n,\ n-1, \cdots, 1$ and labeling $b$'s by $c$'s, and $a$'s by $b$'s. And, then

$Q_{n-1}(z) = P'_n(z) = c_1$. Note that in computing $P_n(z)$, we generated the numbers $b_0,\ b_1, \cdots, b_n$, from the coefficients $a_0,\ a_1, \cdots, a_n$ of $P_n(x)$. So, in computing $P'_n(z)$, we *need to generate the numbers* $c_1,\ c_2, \cdots, c_{n-1}$ *from the coefficients* $b_1,\ b_2, \cdots, b_{n-1}, b_n$ *of the polynomial* $Q_{n-1}(z)$*, which hav already been computed by Algorithm 1.5.*

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

**Computing $P'_n(z)$**

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

(2) A number $z$.

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

**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 1.10** *Let* $P_3(x) = x^3 - 7x^2 + 6x + 5$.

*Let* $z = 2$.

*Here* $a_0 = 5,\ a_1 = 6,\ a_2 = -7,\ a_3 = 1$.

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

$b_3 = a_3 = 1$

$b_2 = a_2 + b_3 z = -7 + 2 = -5$

$b_1 = a_1 + b_2 z = 6 - 10 = -4$

$b_0 = a_0 + b_1 z = 5 - 8 = -3$.

So, $P_3(2) = b_0 = -3$

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

$c_3 = b_3 = 1$

$c_2 = b_2 + c_3 z = -5 + 2 = -3$

$c_1 = b_1 + c_2 z = -4 - 6 = -10.$

So, $P_3'(2) = -10.$

**The Newton-Raphson Method with Horner's Scheme**

We now describe the Newton-Raphson Method for a polynomial using Horner's scheme. Recall that the Newton-Raphson 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:

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

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 1.6** ***The Newton-Raphson 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
$\epsilon$ - 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.
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. $b_0 = a_0 + z\ b_1$

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

**Notes:**

1. The outer-loop corresponds to the Newton-Raphson Method

2. The inner-loop and the statement $b_0 = a_0 + z\ b_1$ corresponds to the evaluation of $P_n(x)$ and its derivative $P_n(x)$ at successive approximations.

**Example 1.11** *Find a root of* $P_3(x) = x^3 - 7x^2 + 6x + 5 = 0$, *using the Newton-Raphson method and Horner's scheme.*

$P_3(x)$ *has a root between 1.5 and 2.*

*Let* $x_0 = 2$

**Iteration 1.** $k = 0$:

$b_0 = -3,\ c_1 = -10$

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

**Iteration 2.** $k = 1$:

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

$b_2 = a_2 + z\, b_3 = -7 + 1.7 = -5.3$

$c_2 = b_2 + z\, c_3 = -5.3 + 1.7 = -3.6$

$b_1 = a_1 + z\, b_2 = 6 + 1.7(-5.3) = -3.0100$

$c_1 = b_1 + z\, c_2 = -3.01 + 1.7(-3.6) = -9.130$

$b_0 = a_0 + z\, b_1 = 5 + 1.7(-3.0100) = -0.1170$

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

(The exact root, correct upto four decimal places is 1.6872).

## 1.10 The Müler Method: Finding a Pair of Complex Roots.

The Müler Method is an extension of the Secant Method. It is conveniently used to approximate a complex root of a real equation.

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

The Müler Method is capable of doing so.

- Recall that starting with two initial approximations $x_0$ and $x_1$, the Secant Method finds the next approximation $x_2$ as the $x$-intercept of the line passing throught $(x_0,\ f(x_0))$, and $(x_1,\ f(x_1))$.

- In the Müler Method, starting with three initial approximations $x_0,\ x_1$ and $x_2$, the next approximaiton $x_3$ is obtained as the $x$-intercept of the parabola $P$ passing through $(x_0,\ f(x_0))$, $(x_1,\ f(x_1))$, and $(x_2,\ f(x_2))$.

- Since a parabola can intersect $x$-axis at two points, the one that is close to $x_2$ is taken as $x_3$.

- So, in Müler's method, $x_3$ is obtained as follows:

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

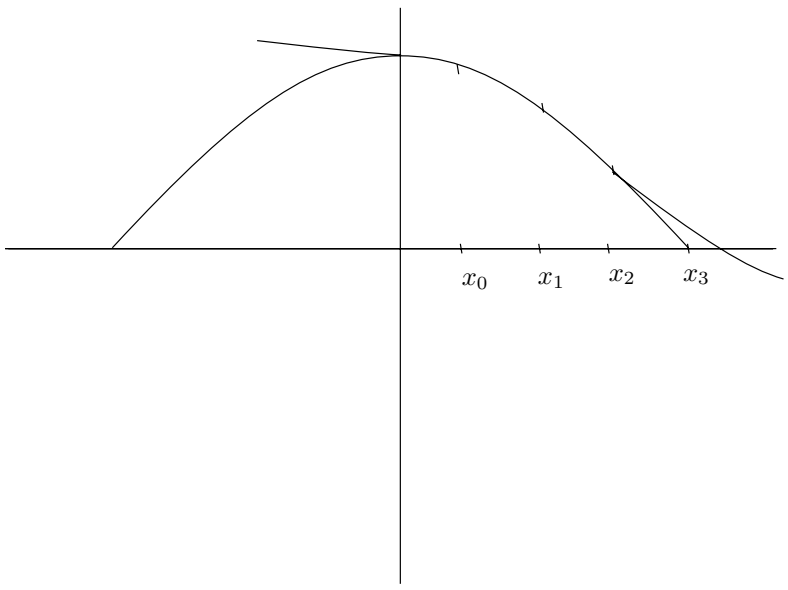
  **2.** Solve the quadratic equation representing the parabola $P$ and take $x_3$ as the root closest to $x_2$.

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

The process, of course, can be continued to obtain other successive approximations. That is, $x_4$ 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 zero 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.

**Geometric Interpretation of the Müler Method**




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

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

$P(x_0) = f(x_0) = a(x_0 - x_2)^2 + b(x_0 - x_2) + c$

$P(x_1) = f(x_1) = a(x_1 - x_2)^2 + b(x_1 - x_2) + c$

$P(x_2) = f(x_2) = c.$

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

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

**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**, instead of using the traditional formula, we use the following formula for solving the quadratic equation $ax^2 + bx + c = 0$.

**Solving the Quadratic Equation**

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

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

In our case, $x_3 - x_2 = \dfrac{-2c}{b \pm \sqrt{b^2 - 4ac}}$

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

**Note:** one can also give on explicit expresions 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 accuracy, it may be easier to solve the following $2 \times 2$ linear system to obtain $a$ and $b$, once $c = f(x_2)$ is computed.

**Algorithm 1.7** ***The Müler Method***

**Inputs:** (i) $f(x)$ - The given functions
(ii) $x_0,\ x_1,\ x_2$ - Initial approximations
(iii) $\epsilon$ - Tolerance
(iv) $N$ - The maximum number of iterations.
**Output:** An approximation of the root $x = \xi$
**Step 1:** Evaluate $f(x_0),\ f(x_1)$, and $f(x_2)$.
**Step 2:** Compute $a, b$, and $c$, as follows:
2.1 $c = f(x_2)$.
2.2 Solve the $2 \times 2$ linear system for $a$ and $b$:

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

**Step 3:** Compute $x_3 = x_2 + \dfrac{-2c}{b \pm \sqrt{b^2 - 4ac}}$ choosing $+$ or $-$ so that the denominator is the largest in magnitude.

**Step 4:** Test if $|x_3 - x_2| < \epsilon$ or if the number of iterations exceeds $N$. If so, stop. Otherwise go to Step 5.

**Step 5:** Relabel $x_3$ as $x_2,\ x_2$ as $x_1$, and $x_1$ as $x_0$, and return to Step 1.

**Example 1.12** $f(x) = x^3 - 2x^2 - 5$

*The roots of* $f(x)$ *are* $\{2.6906, -0.3453 \pm 1.3187i\}$

*We will try to approximate the pair of complex roots here.*

**Iteration 1:** Initial Approximations: $x_0 = -1,\ x_1 = 0,\ x_2 = 1$

**Step 1:** $f(x_0) = -8,\ f(x_1) = -5,\ f(x_2) = -6.$

**Step 2:** Compute $a, b$, and $c$:

**2.1** $c = f(x_2) = 6$

**2.2** $a$ and $b$ are computed by solving the $2 \times 2$ linear system:

$$\left.\begin{array}{l} 4a - 2b = -2 \\ a - b = 1 \end{array}\right]$$

This gives $a = -2,\ b = -3$.

**Step 3:** Compute $x_3 = x_2 - \dfrac{2c}{b - \sqrt{b^2 - 4ac}} = x_2 + \dfrac{12}{-3 - \sqrt{9 - 48}} = 0.25 + 1.5612i$

**Iteration 2.** Relabeling $x_3$ as $x_2$, $x_2$ as $x_1$, and $x_1$ as $x_0$, we have

$x_0 = 0,\ x_1 = 1,\ x_2 = 0.25 + 1.5612i$

**Step 1:** $f(x_0) = -5,\ f(x_1) = -6$, and $f(x_2) = 0.25 - 1.5612i$

**Step 2:** $c = f(x_2) = -2.0625 + 5.0741i$

$a$ and $b$ are obtained by solving the $2 \times 2$ 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 $a = -0.75 + 1.5612i$, $b = -5.4997 - 3.1224i$.

**Step 3:** $x_3 = x_2 - \dfrac{2c}{b - \sqrt{b^2 - 4ac}} = x_2 - (0.8388 + 0.3702i) = -0.5888 + 1.1910i$

**Iteration 3:** Again, relabeling $x_3$ as $x_2$, $x_2$ as $x_1$, and $x_1$ as $x_0$, we have

$x_0 = 1,\ x_1 = 0.25 + 1.5612i,\ x_2 = -0.5888 + 1.1910i$

**Step 1:** $f(x_0) = -6,\ f(x_1) = -2.0627 - 5.073i,\ f(x_2) = -0.5549 + 2.3542i$

**Step 2:** $c = f(x_2) = -0.5549 + 2.3542i$

$a$ and $b$ are obtained by solving the 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 $a = -1.3888 + 2.7522i, b = -2.6339 - 8.5606i$.

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

**Iteration 4**

$x_3 = -0.3563 + 1.3238i$

**Iteration 5**

$x_3 = -0.3492 + 1.3184i$

**Iteration 6**

$x_3 = -0.3464 + 1.3180i$

**Iteration 7**

$x_3 = -0.3455 + 1.3183i$

Since $|\dfrac{x_3 - x_2}{x_2}| < 6.8738 \times 10^{-4}$, we stop here.

Thus, an approximate pair of complex roots is: $\alpha_1 = -0.3455 + 1.3183i$, $\alpha_2 = 0.3455 - 1.3183i$.

**Verify:**

$(x - \alpha_1)(x - \alpha_2)(x - 2.6906) = x^3 - 2.006x^2 - 4.9975$.

**Example 1.13** $f(x) = x^3 - 7x^2 + 6x + 5$

*All three roots are real:* $5.8210,\ 1.6872,\ -0.5090$. *We will try to approximate the root* $1.6872$.

**Iteration 1.**

$x_0 = 0,\ x_1 = 1,\ x_2 = 2$

**Step 1:** $f(x_0) = 5,\ f(x_1) = 5,\ f(x_2) = -3.$

**Step 2:** Compute $c = f(x_2) = -3$.

Then $a$ and $b$ are obtained by solving the $2 \times 2$ 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

$\begin{pmatrix} 4 & -2 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 8 \\ 8 \end{pmatrix}$, which gives $a = -4, b = -12$.

**Step 3:** Compute $x_3 = x_2 - \dfrac{2c}{b - \sqrt{b^2 - 4ac}}$

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

**Iteration 2.**

$x_2 = x_3 = 1.7247,\ x_1 = x_2 = 2,\ \ x_0 = x_1 = 1$

**Step 1:** $f(x_0) = 5,\ f(x_1) = -3,\ f(x_2) = -0.3441$

**Step 2:** Compute $c = -0.3441$

Then $a$ and $b$ are obtained by solving the $2 \times 2$ linear system:

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

which gives $a = -2.2753, b = -9.0227$.

**Step 3:** Compute $x_3 = x_2 - \dfrac{2c}{b - \sqrt{b^2 - 4ac}} = x_2 - 0.0385 = 1.6862$

## 1.11 Sensitivity of the Roots of a Polynomial

*The roots of a polynomial can be very sensitive to small perturbutions of the coefficients of the polynomial.*

For example, take

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

The roots $x_1 = -1,\ x_2 = -1$. 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 happend 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 equation are 1, 2, ..., 20 and thus very well-separated. But, a small perturbation of the coefficient $x^{19}$ which is -210 will change the roots completely. Some of them will even become complex. Try this yourself using MATLAB function **roots**.

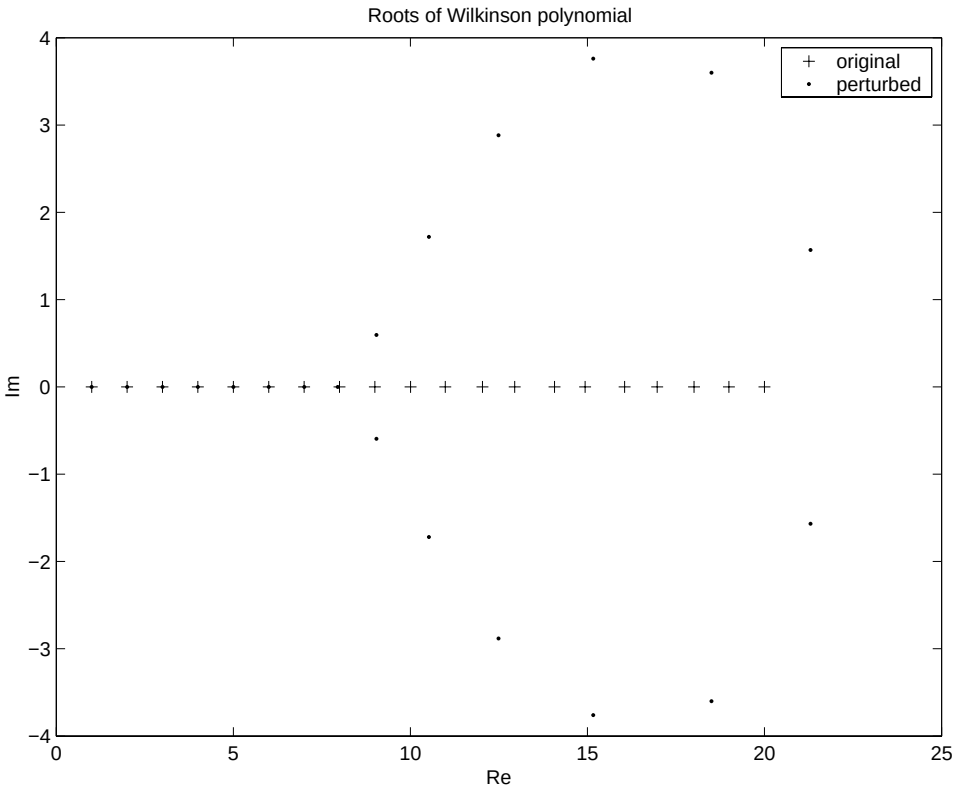



**Figure 1.6: The Graph of $e^x - x - 1$.**

## 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 $Q(x)$ in either case can be generated using **synthetic division** as in Horner's method, and then any of the root-finding

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

**Example 1.14** *Consider finding 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*

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

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

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

*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.*

# PART II

# 2 Interpolation

## 2.1 Problem Statement and Applications

Consider the following table:

| $x_0$ | $f_0$ |
|---|---|
| $x_1$ | $f_1$ |
| $x_2$ | $f_2$ |
| $\vdots$ | $\vdots$ |
| $x_k$ | $f_k$ |
| $\vdots$ | $\vdots$ |
| $x_n$ | $f_n$ |

In the above table, $f_k, k = 0, \cdots, n$ are assumed to be the values of a certain function $f(x)$, evaluated at $x_k, k = 0, \cdots, n$ in an interval containing these points. **Note that only the functional values are known, not the function $f(x)$ itself.** The problem is to find $f_u$ corresponding to a nontabulated intermediate value $x = u$.

Such a problem is called an **Interpolation Problem**. The numbers $x_0,\ x_1, \cdots, x_n$ are called the **nodes**.

**Interpolation Problem**

Given $(n+1)$ points: $(x_0, f_0), (x_1, f_1), \cdots (x_n, f_n)$, find $f_u$
corresponding to $x_u$, where $x_0 < x_u < x_n$; assuming that
$f_0, f_1, \cdots, f_n$ are the values of a certain function $f(x)$
at $x = x_0, x_1, \cdots, x_n$, respectively.

The Interpolation problem is also a classical problem and dates back to the time of **Newton** and **Kepler**, who needed to solve such a problem in analyzing data on the positions of stars and planets. It is also of interest in numerous other practical applications. Here is an example.

## 2.2 Existence and Uniqueness

It is well-known that a continuous function $f(x)$ on $[a, b]$ can be approximated as close as possible by means of a polynomial. Specifically, for each $\epsilon > 0$, there exists a polynomial $P(x)$ such that $|f(x) - P(x)| < \epsilon$ for all $x$ in $[a, b]$. This is a classical result, known as **Weierstrass Approximation Theorem**.

Knowing that $f_k, k = 0, \cdots, n$ are the values of a certain function at $x_k$, the most obvious thing then to do is to construct a polynomial $P_n(x)$ of degree at most $n$ that passes through the $(n + 1)$ points: $(x_0, f_0),\ (x_1, f_1),\ \cdots, (x_n, f_n)$.

**Indeed, if the nodes $x_0, x_1, ..., x_n$ are assumed to be distinct, then such a polynomial always does exist and is unique, as can be seen from the following.**

Let $P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$ be a polynomial of degree at most $n$. If $P_n(x)$ interpolates at $x_0, x_1, \cdots, x_n$, we must have, by definition

$$\begin{aligned} P_n(x_0) &= f_0 = a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_n x_0^n \\ P_n(x_1) &= f_1 = a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_n x_1^n \\ &\vdots \\ P_n(x_n) &= f_n = a_0 + a_1 x_n + a_2 x_n^2 + \cdots + a_n x_2^n \end{aligned} \tag{2.1}$$

These equations can be written in matrix form:

$$\begin{pmatrix} 1 & x_0 & x_0^2 \cdots x_0^n \\ 1 & x_1 & x_1^2 \cdots x_1^n \\ \vdots & & \\ 1 & x_n & x_n^2 \cdots x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{pmatrix}$$

Because $x_0,\ x_1, \cdots, x_n$ are distinct, it can be shown [**Exercise**] that the matrix of the above system is nonsingular. Thus, the linear system for the unknowns $a_0,\ a_1, \cdots, a_n$ has a unique solution, in view of the following well-known result, available in any linear algebra text book.

| |
|---|
| The $n \times n$ algebraic linear system $Ax = b$ has a unique solution for every $b$ if and only if $A$ is nonsingular. |

This means that $P_n(x)$ exists and is unique.

**Theorem 2.1** ***(Existence and Uniqueness Theorem for Polynomial Interpolation)*** *Given $(n+1)$ distinct points $x_0,\ x_1, \cdots, x_n$ and the associated values $f_0,\ f_1, \cdots, f_n$ of a function $f(x)$ at these points (that is, $f(x_i) = f_i,\ i = 0, 1, \cdots, n$)), there is a* **unique polynomial** *$P_n(x)$ of degree at most $n$ such that $P_n(x_i) = f_i, i = 0, 1, \cdots, n$.*

The polynomial $P_n(x)$ in Theorem 3.1 is called the **interpolating polynomial**.

## 2.3 The Lagrange Interpolation

Once we know that the interpolating polynomial exists and is unique, the problem then becomes how to construct an interpolating polynomial; that is, how to construct a polynomial $P_n(x)$ of degree at most $n$, such that

$$P_n(x_i) = f_i,\ i = 0, 1, \cdots, n.$$

It is natural to obtain the polynomial by solving the linear system (3.1) in the previous section. Unfortunately, the matrix of this linear system, known as the **Vandermonde Matrix**, is usually **highly ill-conditioned**, and the **solution of such an ill-conditioned system, even by the use of a stable method, may not be accurate.** There are, however, several other ways to construct such a polynomial, that do not require solution of a Vandermonde system. We describe one such in the following:

Suppose $n = 1$, that is, suppose that we have only two points $(x_0, f_0),\ (x_1, f_1)$, then it is easy to see that the linear polynomial

$$P_1(x) = \frac{x - x_1}{(x_0 - x_1)} f_0 + \frac{(x - x_0)}{(x_1 - x_0)} f_1$$

is an interpolating polynomial, because

$$P_1(x_0) = f_0,\ P_1(x_1) = f_1.$$

For convenience, we shall write the polynomial $P_1(x)$ in the form

$$P_1(x) = L_0(x)f_0 + L_1(x)f_1,$$

where, $L_1(x) = \dfrac{x - x_0}{x_1 - x_0}$, and $L_1(x) = \dfrac{x - x_0}{x_1 - x_0}$.

Note that both the polynomials $L_0(x)$ and $L_1(x)$ are polynomials of degree 1.

The concept can be generalized easily for polynomials of higher degrees.

To generate polynomials of higher degrees, let's define the set of polynomials $\{L_k(x)\}$ recursively, as follows:

$$L_k(x) = \frac{(x - x_0)(x - x_1)\cdots(x - x_{k-1})(x - x_{k+1})\cdots(x - x_n)}{(x_k - x_0)(x_k - x_1)\cdots(x_k - x_{k-1})(x_k - x_{k+1})\cdots(x_k - x_n)},\ k = 0, 1, 2, \cdots, n. \tag{2.2}$$

We will now show that the polynomial $P_n(x)$ defined by

$$P_n(x) = L_0(x)f_0 + L_1(x)f_1 + \cdots + L_n(x)f_n \tag{2.3}$$

is an interpolating polynomial.

To see this, note that

$$\begin{aligned}
L_0(x) &= \frac{(x - x_1)\cdots(x - x_n)}{(x_0 - x_1)\cdots(x_0 - x_n)} \\
L_1(x) &= \frac{(x - x_0)(x - x_2)\cdots(x - x_n)}{(x_1 - x_0)(x_1 - x_2)\cdots(x_1 - x_n)} \\
&\vdots \\
L_n(x) &= \frac{(x - x_0)(x - x_1)(x - x_2)\cdots(x - x_{n-1})}{(x_n - x_0)(x_n - x_1)(x_n - x_2)\cdots(x_n - x_{n-1})}
\end{aligned}$$

Also, note that

$$L_0(x_0) = 1,\ L_0(x_1) = L_0(x_2) = \cdots = L_0(x_n) = 0$$
$$L_1(x_1) = 1,\ L_1(x_0) = L_1(x_2) = \cdots = L_n(x_n) = 0$$

In general

$L_k(x_k) = 1$ and $L_k(x_i) = 0,\ i \neq k$.

Thus

$$P_n(x_0) = L_0(x_0)f_0 + L_1(x_0)f_1 + \cdots + L_n(x_0)f_n = f_0$$
$$P_n(x_1) = L_0(x_1)f_0 + L_1(x_1)f_1 + \cdots + L_n(x_1)f_n = 0 + f_1 + \cdots + 0 = f_1$$
$$\vdots$$
$$P_n(x_n) = L_0(x_n)f_0 + L_1(x_n)f_1 + \cdots + L_n(x_n)f_n = 0 + 0 + \cdots + 0 + f_n = f_n$$

That is, the polynomial $P_n(x)$ has the property that $P_n(x_k) = f_k,\ k = 0, 1, \cdots, n.$

The polynomial $P_n(x)$ defined by (3.3) is known as the **Lagrange Interpolating Polynomial**.

**Example 2.1** *Interpolate $f(x)$ from the following table:*

| | |
|---|---|
| 0 | 7 |
| 1 | 13 |
| 2 | 21 |
| 4 | 43 |

*and find and approximation to $f(3)$.*

$$L_0(x) = \frac{(x-1)(x-2)(x-4)}{(-1)(-2)(-4)}$$
$$L_1(x) = \frac{(x-0)(x-2)(x-4)}{1\cdot(-1)(-3)}$$
$$L_2(x) = \frac{(x-0)(x-1)(x-4)}{2\cdot 1\cdot(-2)}$$
$$L_3(x) = \frac{(x-0)(x-1)(x-2)}{4\cdot 3\cdot 2}$$

$P_3(x) = 7L_0(x) + 13L_1(x) + 21L_2(x) + 43L_3(x)$

*Thus,* $P_3(3) = 7L_0(3) + 13L_1(3) + 21L_2(3) + 43L_3(3) = 31.$

*Now,* $L_0(3) = \frac{1}{4},\ L_1(3) = -1,\ L_2(3) = \frac{3}{2},\ L_3(3) = \frac{1}{4}.$

*So,* $P_3(3) = 7L_0(3) + 13L_1(3) + 21L_2(3) + 43L_3(3) = 31.$

**Verify:** Note that $f(x)$ in this case is $f(x) = x^2 + 5x + 7$, and the exact value of $f(x)$ at $x = 3$ is 31.

**Example 2.2** *Given*

| $i$ | $x_i$ | $f(x_i)$ |
|---|---|---|
| 0 | 2 | $\frac{1}{2}$ |
| 1 | 2.5 | $\frac{1}{2.5}$ |
| 2 | 4 | $\frac{1}{4}$ |

*We want to find* $f(3)$.

$$L_0(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} = \frac{(x-2.5)(x-4)}{(-0.5)(-2)} = (x-2.5)(x-4)$$

$$L_1(x) = \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} = \frac{(x-2)(x-4)}{(0.5)(-1.5)} = -\frac{1}{0.75}(x-2)(x-4)$$

$$L_2(x) = \frac{(x-x_1)(x-x_0)}{(x_2-x_1)(x_2-x_0)} = \frac{1}{3}(x-2.5)(x-2)$$

$$P_2(x) = f(x_0)L_0(x) + f(x_1)L_1(x) + f(x_2)L_2(x) = \frac{1}{2}L_0(x) + \frac{1}{2.5}L_1(x) + \frac{1}{4}L_2(x)$$

$$P_2(3) = \frac{1}{2}L_0(3) + \frac{1}{2.5}L_1(3) + \frac{1}{4}L_2(3) = \frac{1}{2}(-0.5) + \frac{1}{2.5}\left(\frac{1}{0.75}\right) + \frac{1}{4}\left(\frac{.5}{3}\right) = 0.3250$$

**Verify:** (The value of $f(x)$ at $x = 3$ is $\frac{1}{3} = 0.3333$).

## 2.4 Error in Interpolation

If $f(x)$ is approximated by an interpolating polynomial $P_n(x)$, we would like to obtain an expression for the error of interpolation at a give intermediate point, say, $\bar{x}$.

That is, we would like to calculate $E(\bar{x}) = f(\bar{x}) - P_n(\bar{x})$.

Note that, since $P_n(x_i) = f(x_i),\ E(x_i) = 0,\ i = 0, 1, 2, \cdots, n)$, that is, **there are no errors of interpolating at a tabulated point.**

Here is a result for the expression of $E(\bar{x})$.

**Theorem 2.2** ***(Interpolation-Error Theorem)*** *Let $P_n(x)$ be the interpolating polynomial that interpolates at $(n+1)$ distinct numbers in $[a, b]$, and let $f(x)$ be $(n+1)$ times continuously differentiable on $[a, b]$. Then for every $\bar{x}$ in $[a, b]$, there exists a number $\xi = \xi(\bar{x})$ (depending on $\bar{x}$) such that*

$$E_n(\bar{x}) = f(\bar{x}) - P_n(\bar{x}) = \frac{f^{(n+1)}(\xi(\bar{x}))}{(n+1)!} \prod_{i=0}^{n} (\bar{x} - x_i). \tag{2.4}$$

**Proof:** If $\bar{x}$ is one of the numbers $x_0, x_1, \cdots, x_n$: then the result follows trivially. Because, the error in this case is zero, and the result will hold for any arbitrary $\xi$.

Next, assume that $\bar{x}$ is not one of the numbers $x_0, x_1, \cdots, x_n$.

Define a function $g(t)$ in variable $t$ in $[a, b]$:

$$g(t) = f(t) - P_n(t) - [f(\bar{x}) - P_n(\bar{x})] * \left[ \frac{(t - x_0)(t - x_1) \cdots (t - x_n)}{(\bar{x} - x_0)(\bar{x} - x_1) \cdots (\bar{x} - x_n)} \right]. \tag{2.5}$$

Then, noting that $f(x_k) = P_n(x)$, for $k = 0, 1, 2, \cdots, n$, we have

$$\begin{aligned} g(x_k) &= f(x_k) - P_n(x_k) - [f(\bar{x}) - P_n(\bar{x})] \left[ \frac{(x_k - x_0) \cdots (x_k - x_n)}{(\bar{x} - x_0) \cdots (\bar{x} - x_n)} \right] \\ &= P_n(x_k) - P_n(x_k) - [f(\bar{x}) - P_n(\bar{x})] \ \times 0 \\ &= 0 \end{aligned} \tag{2.6}$$

(Note that the numerator of the fraction appearing above contains the term $(x_k - x_k) = 0$).

Furthermore,

$$\begin{aligned} g(\bar{x}) &= f(\bar{x}) - P_n(\bar{x}) - [f(\bar{x}) - P_n(\bar{x})] * \left[ \frac{(\bar{x} - x_0) \cdots (\bar{x} - x_n)}{(\bar{x} - x_0) \cdots (\bar{x} - x_n)} \right] \\ &= f(\bar{x}) - P_n(\bar{x}) - f(\bar{x}) + P_n(\bar{x}) \\ &= 0 \end{aligned} \tag{2.7}$$

Thus, $g(t)$ becomes identically zero at $(n + 2)$ distinct points: $x_0, x_1, \cdots, x_n$, and $\bar{x}$. Furthermore, $g(t)$ is $(n + 1)$ times continuously differentiable, since $f(x)$ is so.

Therefore, by **generalized Rolle's theorem**, [ ], there exists a number $\xi(\bar{x})$ in $(a, b)$ such that $g^{(n+1)}(\xi) = 0$.

Let's compute $g^{(n+1)}(t)$ now. ¿From (2.5) we have

$$g^{(n+1)}(t) = f^{(n+1)}(t) - P_n^{(n+1)}(t) - [f(\bar{x}) - P_n(\bar{x})] \frac{d^{n+1}}{d^{n+1}} \left[ \frac{(t - x_0)(t - x_1) \cdots (t - x_n)}{((\bar{x} - x_0)(\bar{x} - x_1) \cdots (\bar{x} x_n)} \right]$$

Then

$$\begin{aligned} \frac{d^{n+1}}{dt^{n+1}} & \left[ \frac{(t - x_0)(t - x_1) \cdots (t - x_n)}{(\bar{x} - x_0)(\bar{x} - x_1) \cdots (\bar{x} x_n)} \right] \\ &= \frac{1}{(\bar{x} - x_0)(\bar{x} - x_1) \cdots (\bar{x} - x_n)} \cdot \frac{d^{n+1}}{dt^{n+1}} [(t - x_0)(t - x_1) \cdots (t - x_n)] \\ &= \frac{1}{(\bar{x} - x_0)(\bar{x} - x_1) \cdots (\bar{x} - x_n)} (n + 1)! \end{aligned}$$

(note that the expression within [ ] is a polynomial of degree $n+1$).

Also, $P_n^{(n+1)}(t) = 0$, because $P_n$ is a polynomial of degree at most $n$. Thus, $P_n^{(n+1)}(\xi) = 0$.

So,

$$g^{(n+1)}(\xi) = f^{(n+1)}(\xi) - \frac{(f(\bar{x}) - P_n(\bar{x}))}{(\bar{x} - x_0)\cdots(\bar{x} - x_n)}(n+1)! \tag{2.8}$$

Since $g^{(n+1)}(\xi) = 0$, from (2.8), we have $E_n(\bar{x}) = f(\bar{x}) - P_n(\bar{x}) = \dfrac{f^{(n+1)}(\xi)}{(n+1)!}(\bar{x} - x_0)\cdots(\bar{x} - x_n)$.

**Remark:** To obtain the error of interpolation using the above theorem, we need to know the $(n+1)$th derivative of the $f(x)$ or its absolute maximum value on the interval $[a, b]$. Since in practice this value is hardly known, this error formula is of limited use only.

**Example 2.3** *Let's compute the maximum absolute error for Example 3.2.*

*Here* $n = 2$.

$$\begin{aligned} E_2(\bar{x}) &= f(\bar{x}) - P_2(\bar{x}) \\ &= \frac{f^{(3)}(\xi)}{3!}(\bar{x} - x_0)(\bar{x} - x_1)(\bar{x} - x_2) \end{aligned}$$

*To know the maximum value of* $E_2(\bar{x})$, *we need to know* $f^{(3)}(x)$.

*Let's compute this now:*

$$f(x) = \frac{1}{x}, \;\; f'(x) = -\frac{1}{x^2}, \;\; f''(x)\frac{2}{x^3}, \;\; f^{(3)}(x) = -\frac{6}{x^4}.$$

*So,* $|f^{(3)}(\xi)| < \dfrac{6}{2^4} = \dfrac{6}{16}$ *for* $0 < x \leq 2$.

*Since* $\bar{x} = 3,\ x_0 = 2,\ x_1 = 2.5,\ x_2 = 4$, *we have*

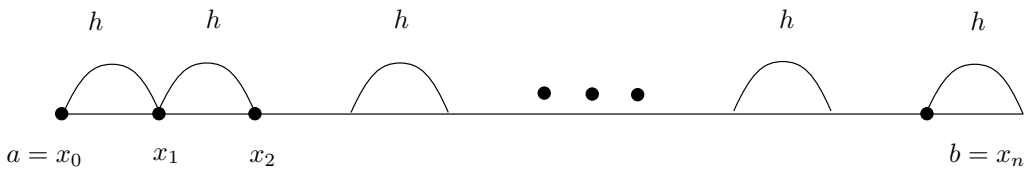
$$|E_2(\bar{x})| \leq |\ \frac{6}{16} \times \frac{1}{6}\ (3-2)(3-2.5)(3-4)| = 0.0625.$$

*Note that in four-digit arithmetic, the difference between the value obtained by interpolation and the exact value is* $0.3333 - 0.3250 = 0.0083$.

## 2.5 Simplification of the Error Bound for Equidistant Nodes

The error formula in Theorem 3.2 can be simplified in case the tabulated points (nodes) are equally spaced; because, in this case it is possible to obtain a nice bound for the expression: $(\bar{x} - x_0)(\bar{x} - x_1)\cdots(\bar{x} - x_n)$.

Suppose the nodes are equally spaced with spacing $h$; that is $x_{i+1} - x_i = h$.




Then it can be shown **[Exercise]** that

$$|(\bar{x} - x_0)(\bar{x} - x_1)\cdots(\bar{x} - x_n)| \leq \frac{h^{n+1}}{4} n!$$

If we also assume that $|f^{(n+1)}(x)| \leq M$, then we have

$$|E_n(\bar{x})| = |f(\bar{x}) - P_n(\bar{x})| \leq \frac{M}{(n+1)!} \frac{h^{n+1}}{4} n! = \frac{Mh^{n+1}}{4(n+1)}. \tag{2.9}$$

**Example 2.4** *Suppose a table of values for* $f(x) = \cos x$ *has to be prepared in* $[0, 2\pi]$ *with equal spacing nodes of spacing* $h$*, using* **linear interpolation***, with an error of interpolation of at most* $5 \times 10^{-8}$*. How small should* $h$ *be?*

*Here* $n = 1$.

$f(x) = \cos x,\ f'(x) = -\sin x,\ f^2(x) = f''(x) = -\cos x$

$\max |f^{(2)}(x)| = 1,$ *for* $0 \leq x \leq 2\pi$

*Thus* $M = 1$.

So, by (3.9) above we have

$$|E_1(\bar{x})| = |f(\bar{x}) - P_1(\bar{x})| \leq \frac{h^2}{8}.$$

Since the maximum error has to be $5 \times 10^{-7}$, we must have:

$\frac{h^2}{8} \leq 5 \times 10^{-7} = \frac{1}{2} \times 10^{-6}$. That is, $h\ \leq 6.3246 \times 10^{-4}$.

**Example 2.5** *Suppose a table is to be prepared for the function* $f(x) = \sqrt{x}$ *on* $[1, 2]$. *Determine the spaceing* $h$ *in a table such that the interpolation with a polynomial of degree 2 will give accuracy* $\epsilon = 5 \times 10^{-8}$.

*We first compute the maximum absolute error.*

*Since* $f^{(3)}(x) = \frac{3}{8}x\frac{-5}{2}$,

$$M = \left|f^{(3)}(x)\right| \leq \frac{3}{8} \quad \textit{for} \;\; 1 \leq x \leq 2.$$

Thus, taking $n = 2$ in (2.9) the maximum (absolute) error is $\frac{3}{8} \times \frac{h^3}{4x^3} = \frac{1}{32}h^3$.

Thus, to have an accuracy of $\epsilon = 5 \times 10^{-8}$, we must have $\frac{1}{32}h^3 < 5 \times 10^{-8}$ or $h^3 < 160 \times 10^{-8}$.

This means that a spacing $h$ of about $h = \sqrt[3]{160 \times 10^{-8}} = 0.0117$ will be needed in the Table to guarantee the accuracy of $5 \times 10^{-8}$.

## 2.6 Divided Differences and the Newton-Interpolation Formula

A major difficulty with the Lagrange Interpolation is that one is not sure about the degree of interpolating polynomial needed to achieve a certain accuracy. Thus, if the accuracy is not good enough with polynomial of a certain degree, one needs to increase the degree of polynomial, and **computations need to be started all over again.**

Furthermore, computing various Lagrangian polynomials is an expensive procedure. **It is, indeed, desirable to have a formula which makes use of** $P_{k-1}(x)$ **in computing** $P_k(x)$**.**

The following form of interpolation, known as **Newton's interpolation** allows us to do so.

The idea is to obtain the interpolating polynomial $P_n(x)$ in the following form:

$$P_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}) \tag{2.10}$$

The constants $a_0$ through $a_n$ can be determined as follows:

For $x = x_0$, $\;P_n(x_0) = a_0 = f_0$

For $x = x_1$, $\;P_n(x_1) = a_0 + a_1(x_1 - x_0) = f_1$,

which gives

$$a_1 = \frac{f_1 - a_0}{x_1 - x_0} = \frac{f_1 - f_0}{x_1 - x_0}.$$

The other numbers $a_i, i = 2, ..., n$ can similarly be obtained.

It is convenient to introduce the following notation, because we will show how the numbers $a_0, ..., a_n$ can be obtained using these notations.

$$f(x_i) = f[x_i] \text{ and } f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}$$

Similarly,

$$f[x_i, x_{i+1}, \cdots, x_{i+k-1}, x_{i+k}] = \frac{f[x_{i+1}, ..., x_{i+k-1}, x_{i+k}] - f[x_i, x_{i+1}, ..., x_{i+k-1}]}{x_{i+k} - x_i}$$

With these notations, we then have

$$a_0 = f_0 = f(x_0) = f[x_0]$$

$$a_1 = \frac{f_1 - f_0}{x_1 - x_0} = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{f[x_1] - f[x_0]}{x_1 - x_0} = f[x_0, x_1].$$

Continuing this, it can be shown that **[Exercise]**

$$a_k = f[x_0, x_1, \cdots, x_k]. \tag{2.11}$$

The number $f[x_0, x_1, \cdots, x_k]$ is called the $k$-th **divided difference**.

Substituting these expressions of $a_k$ in (2.11), the interpolating polynomial $P_n(x)$ now can be written in terms of the **divided differences:**

$$\begin{aligned} &P_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots + \\ &f[x_0, x_1, \cdots, x_n](x - x_0)(x - x_1) \cdots (x - x_{n-1}). \end{aligned} \tag{2.12}$$

**Notes:**

(i) Each divided difference can be obtained from two previous ones of lower orders.

For example, $f[x_0, x_1, x_2]$ can be computed from $f[x_0, x_1]$, and $f[x_1, x_2]$, and so on. Indeed, they can be arranged in form of a table as shown below:

**Table of Divided Differences ($n = 4$)**

| $x$ | $f(x)$ | $1^{st}$ Divide Difference | $2^{nd}$ Divided Difference | $3^{rd}$ Divided Difference |
|---|---|---|---|---|
| $x_0$ | $f_0$ | | | |
| $x_1$ | $f_1$ | $f[x_0, x_1]$ | $f[x_0, x_1, x_2]$ | |
| $x_2$ | $f_2$ | $f[x_1, x_2]$ | $f[x_1, x_2, x_3]$ | $f[x_0, x_1, x_2, x_3]$ |
| $x_3$ | $f_3$ | $f[x_2, x_3]$ | $f[x_2, x_3, x_4]$ | $f[x_1, x_2, x_3, x_4]$ |
| $x_4$ | $f_4$ | $f[x_3, x_4]$ | | |

(ii) Note that in computing $P_n(x)$ we need only the diagonal entries of the above table; that is, we need only $f[x_0], f[x_0, x_1], \cdots, f[x_0, x_1, \cdots, x_n]$.

(iii) Since the divided differences are generated recursively, the interpolating polynomials of successively higher degrees can also be generated recursively. **Thus the work done previously can be used gainfully.**

For example,

$P_1(x) = f[x_0] + f[x_1, x_0](x - x_0)$

$P_2(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1)$

$= P_1(x) + f[x_0, x_1, x_2](x - x_0)(x - x_1).$

Similarly, $P_3(x) = P_2(x) + f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2)$

$$P_4(x) = P_3(x) + f[x_0, x_1, x_2, x_3, x_4](x - x_0)(x - x_1)(x - x_2)(x - x_3)$$

and so on.

Thus, in computing $P_2(x)$, $P_1(x)$ has been gainfully used; in computing $P_3(x)$, $P_2(x)$ has been gainfully used, etc.

**Example 2.6** *Interpolate at $x = 2.5$ using the following Table, with polynomials of degree 3 and 4.*

| $n$ | $x$ | $f$ | 1st diff. | 2nd diff. | 3rd diff. | 4th diff | 5th diff |
|---|---|---|---|---|---|---|---|
| 0 | 1.0 | 0 | 0.35218 | $-0.1023$ | .0265533 | $-.006408$ | .001412 |
| 1 | 1.5 | 0.17609 | 0.24988 | $-0.0491933$ | .0105333 | $-.002172$ | |
| 2 | 2.0 | 0.30103 | 0.17609 | $-0.0281267$ | .0051033 | | |
| 3 | 3.0 | 0.47712 | 0.1339 | $-0.01792$ | | | |
| 4 | 3.5 | 0.54407 | 0.11598 | | | | |
| 5 | 4.0 | 0.60206 | | | | | |

¿From (2.12), with $n = 3$, we have

$$\begin{aligned} P_3(2.5) &= 0 + (2.5 - 1.0)(.35218) + (2.5 - 1.0)(2.5 - 1.5)(-.1023) \\ &\quad +(2.5 - 1.0)(2.5 - 1.5)(2.5 - 2.0)(.0265533) \\ &= .52827 - .15345 + .019915 \\ &= \boxed{0.394735} \end{aligned}$$

Similarly, with $n = 4$, we have

$$\begin{aligned} P_4(2.5) &= P_3(2.5) + (2.5 - 1.0)(2.5 - 1.5)(2.5 - 2.0)(2.5 - 3.0)(-.006408) \\ &= .394735 + .002403 = \boxed{0.397138} . \end{aligned}$$

Note that $P_4(2.5) - P_3(2.5) = \boxed{0.002403}$.

**Verification**. The above is a table for $\log(x)$.

The exact value of $\log(2.5)$ (correct up to 5 decimal places) is 0.39794.

**(Note that in computing $P_4(2.5)$, $P_3(2.5)$ computed previously has been gainfully used)**.

**Algorithm 2.1** ***Algorithm for Generating Divided Differences***

**Inputs:**

The definite numbers $x_0, x_1, \cdots, x_n$ and the values $f_0, f_1, \cdots, f_n$.

**Outputs:**

The Divided Differenced $D_{00}, D_{11}, \cdots D_{nn}$.

**Step 1:** (Initialization). Set

$d_{i,0} = 0 = f_i,\ i = 0, 1, 2, \cdots, n.$

For $i = 1, 2, \cdots, n$ do

$i = 1, 2, \cdots i$ do

$$D_{ij} - \frac{D_{i,j-1} - D_{i-1,j-1}}{x_i - x_{i-j}}$$

**End**

**A Relationship Between $n$th Divided Difference and the $n^{th}$ Derivative**

The following theorem shows how the $n$th derivative of a function $f(x)$ is related to the $n$th divided difference. The proof is omitted. It can be found in any advanced numerical analysis text book (e.g., Atkins on (1978), p. 144).

**Theorem 2.3** *Suppose $f$ is $n$ times continuously differentiable and $x_0, x_1, \cdots, x_n$ are $(n+1)$ distinct numbers in $[a, b]$. Then there exists a number $\xi$ in $(a, b)$ such that*

$$f[x_0, x_1, \cdots, x_n] = \frac{f^{(n)}(\xi)}{n!}$$

**The Newton Interpolation with Equally Spaced Nodes**

Suppose again that $x_0, \cdots, x_n$ are equally spaced with spacing $h$;

that is $x_{i+1} - x_i = h,\ i = 0, 1, 2, ..., n-1$. Let $x - x_0 = sh$.

Then $x - x_0 = sh$

$$x - x_1 = x - x_0 + x_0 - x_1 = (x - x_0) - (x_1 - x_0) = sh - h = (s-1)h$$

In general, $x - x_i = (s - i)h$.

So,

$$\begin{aligned} P_n(x) &= P_n(x_0+sh) = f[x_0] + f[x_0,x_1](x-x_0) + \cdots + f[x_0,x_1,\cdots,x_n](x-x_0)\cdots(x-x_{n-1}) \\ &= f[x_0] + sh\ f[x_0,x_1] + s(s-1)h^2 f[x_0,x_1,x_2] + \cdots + s(s-1)\cdots(s-h+1)h^n f[x_0,\cdots,x_h] \\ &\sum_{k=0}^{n} s(s-1)\cdots(s-k+1)h^k f[x_0,\cdots,x_k]. \end{aligned} \tag{2.13}$$

Invoke now the notation:

$$\left( \begin{array}{c} s \\ k \end{array} \right) = \frac{(s)(s-1)\cdots(s-k+1)}{k!}$$

We can then write

$$P_n(x) = P_n(x_0+sh) = \sum_{k=0}^{n} \left( \begin{array}{c} s \\ k \end{array} \right) h^k k! f[x_0,\cdots,x_n] \tag{2.14}$$

**The Newton Forward-Difference Formula**

Let's introduce the notations:

$\Delta f_i = f(x_{i+1}) - f(x_i)$.

Then, $\Delta f_0 = f(x_1) - f(x_0)$

So,

$$f[x_0,x_1] = \frac{f(x_1)-f(x_0)}{x_1-x_0} = \frac{\Delta f_0}{h}. \tag{2.15}$$

Also, note that

$$\begin{aligned} \Delta^2 f_0 &= \Delta(\Delta f_0) = \Delta(f(x_1) - f(x_0)) \\ &= \Delta f(x_1) - \Delta f(x_0) \\ &= f(x_2) - f(x_1) - f(x_1) + f(x_0) \\ &= f(x_2) - 2f(x_1) + f(x_0) \end{aligned} \tag{2.16}$$

So,

$$\begin{aligned} f[x_0,x_1,x_2] &= \frac{f[x_1,x_2]-f[x_0,x_1]}{(x_2-x_0)} \\ &= \frac{\frac{f(x_2)-f(x_1)}{x_2-x_1} - \frac{f(x_1)-f(x_0)}{x_1-x_0}}{(x_2-x_0)} \\ &= \frac{f(x_2)-2f(x_1)-f(x_0)}{h\times 2h} = \frac{\Delta^2 f_0}{2h^2} \end{aligned} \tag{2.17}$$

In general, we have

$$f[x_0, x_1, \ldots, x_k] = \frac{1}{k!h^k}\Delta^k f_0. \tag{2.18}$$

**Proof is by induction on $k$.**

For $k = 0$, the result is trivially true. We have also proved the result for $k = 1$, and $k = 2$.

Assume now that the result is true $k = m$. Then we need to show that the result is also true for $k = m + 1$.

Now, $f[x_0, \cdots, x_{m+1}] = f[x_1, \cdots, x_{m+1}] - \dfrac{f[x_0, \cdots, x_m]}{x_{m+1} - x_0}$

$= \dfrac{\left(\frac{\Delta^m f_1}{m!h^m} - \frac{\Delta^m f_0}{m!h^m}\right)}{(m+1)h} = \dfrac{\Delta^m(f_1 - f_0)}{m!(m+1)h^{m+1}} = \dfrac{\Delta^{m+1} f_0}{(m+1)!h^{m+1}}$

The numbers $\Delta^k f_i$ are called $k^{th}$ order forward differences of $f$ at $x = i$.

We now show how the interpolating polynomial $P_n(x)$ given by (2.14) can be written using forward differences.

$$\begin{aligned} P_n(x) = P_n(x_0 + sh) &= f[x_0] + shf[x_0, x_1] + s(s-1)h^2 f[x_0, x_1, x_2] \\ &\quad + \cdots + s(s-1)\cdots(s-n+1)h^n f[x_0, x_1.\cdots, x_n] \\ &= \sum_{k=0}^{n} s(s-1)\cdots(s-k+1)h^k f[x_0, x_1, \cdots, x_k] \\ &= \sum_{k=0}^{n} \binom{s}{k} k!h^k f[x_0, x_1, \cdots, x_n] = \sum_{k=0}^{n} \binom{s}{k} \Delta^k f_0 \text{ using (2.18).} \end{aligned}$$

**Newton's Forward-Difference Interpolation Formula**

Let $x_0, x_1, ..., x_n$ be $(n+1)$ equidistant points with distance $h$; that is $x_{i+1} - x_i = h$. Then Newton's interpolating polynomial $P_n(x)$ of degree at most $n$, using forward-differences $\Delta^k f_0,\ k = 0, 1, 2, ..., n$ is given by

$$P_n(x) = \sum_{k=0}^{n} \binom{s}{k} \Delta^k f_0,$$

where $s = \dfrac{x - x_0}{h}$.

**Illustration: The Forward Difference Table with $n = 4$**

| $x$ | $f(x)$ | $\Delta f$ | $\Delta^2 f$ | $\Delta^3 f$ | $\Delta^4 f$ |
|---|---|---|---|---|---|
| $x_0$ | $f_0$ | | | | |
| $x_1$ | $f_1$ | $\Delta f_0$ | | | |
| $x_2$ | $f_2$ | $\Delta f_1$ | $\Delta^2 f_0$ | | |
| $x_3$ | $f_3$ | $\Delta f_2$ | $\Delta^2 f_1$ | $\Delta^3 f_0$ | |
| $x_4$ | $f_4$ | $\Delta f_3$ | $\Delta^2 f_2$ | $\Delta^3 f_1$ | $\Delta^3 f_0$ |

**Example 2.7** *Let* $f(x) = e^x$.

| $x$ | $f$ | $\Delta f$ | $\Delta^2 f$ | $\Delta^3 f$ | |
|---|---|---|---|---|---|
| 0 | 1 | | | | |
| 1 | 2.7183 | 1.7183 | | | |
| 2 | 7.3891 | 4.6709 | 2.8817 | 5.2147 | |
| 3 | 20.0855 | 12.6964<br>34.5127 | 8.0964<br>21.8163 | 13.7199 | 8.5052 |
| 4 | 54.5982 | | | | |

*Let* $x = 1.5$

*Then* $s = \dfrac{x - x_0}{h} = \dfrac{1.5 - 0}{1} = 1.5$

$$P_4(1.5) = 1.7183 \times 1 + (1.5)(1.7183) + (1.5)\frac{(1.5-1)(1.5-2)}{2}$$
$$\times 2.8817 + (1.5)\frac{(1.5-1)(1.5-2)(1.5-3)}{6} \times 5.0734 = 4.0852.$$

*The correct answer up to* 4 *decimal digits is* 4.4817.

**Interpolation using Newton-Backward Differences**

While interpolating at some value of $x$ near the end of the difference table, it is logical to reorder the nodes

so that the end-differences can be used in computation. The backward differences allow us to do so.

The backward differences are defined by

$\nabla f_n = f_n - f_{n-1}, n = 1, 2, 3, \ldots,$

and

$\nabla^k f_n = \nabla(\nabla^{k-1} f_n), k = 2, 3, \ldots$

Thus, $\nabla^2 f_n = \nabla(\nabla f_n) = \nabla(f_n - f_{n-1})$

$$= f_n - f_{n-1} - (f_{n-1} - f_{n-2})$$

$$= f_n - 2f_{n-1} + f_{n-2},$$

and so on.

The following a relationship between the backward-differences and the divided differences can be obtained.

$$f[x_{n-k}, \cdots, x_{n-1}, x_n] = \frac{1}{k!h^k}\nabla^k f_n.$$

Using these backward-differences, we can write the Newton interpolation formula as:

$$P_n(x) = f_n + s\nabla f_n + \frac{s(s+1)}{2!}\nabla^2 f_n + ... + \frac{s(s+1)...(s+h-1)}{n!}\nabla^n f_n.$$

Again, using the notation $\binom{-s}{k} = \dfrac{(-s)(-s-1)\cdots(-s-k+1)}{k!}$

we can rewrite the above formula as:

**Newton's Backward-Difference Interpolations Formula**

$$P_n(x) = \sum_{k=0}^{n}(-1)^k \binom{-s}{k} \bigtriangledown^k f_n.$$

## 2.7 Spline-Interpolation

So far we have been considering interpolation by means of a single polynomial in the entire range.

Let's now consider interpolation using different polynomials (but of the same degree) at different intervals.

Let the function $f(x)$ be defined at the nodes $a = x_0, x_1, x_2, ..., x_n = b$. The problem now is to construct

piecewise polynomials $S_j(x)$ on each interval $[x_j, x_{j+1}]$, $j = 0, 1, 2, ..., n-1$, so that the resulting function $S(x)$ is an interpolant for the function $f(x)$.

The simplest such polynomials are of course, linear polynomials (straight lines). The interpolating polynomial in this case is called **linear spline**. Two most disadvantages of a linear spline are (i) the *convergence is rather very slow*, and (ii) *not suitable for applications demanding smooth approximations*, since these splines have corner at the knots.

"Just imagaine a cross section of an airplane wing in the form of a linear spline and you quickly decide to go by rail. (Stewart (1998), p. 93)."

Likewise the quadratic splines have also certain disadvantes.

The most common and widely used splines are **cubic splines**.

Let $S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$.

Since $S_j(x)$ contains four unknowns, to construct $n$ cubic polynomials $S_0(x), S_1(x), ..., S_{n-1}(x)$, we need $4n$ conditions. To have these $4n$ conditions, a cubic spline $S(x)$ for the function $f(x)$ can be conveniently defined as follows:

**Cubic Spline Interpolant**

A function $S(x)$, denoted by $S_j(x)$ over the interval $[x_j, x_{j+1}]$, $j = 0, 1, ..., n-1$ is called a cubic spline interpolant if the following conditions hold:

(0) $S(x)$ is a cubic polynomial $S_j(x)$ in the interval $[x_j, x_{j+1}]$.

(i) $S_j(x_j) = f_j$, $j = 0, 1, 2, ..., n$.

(ii) $S_{j+1}(x_{j+1}) = f_{j+1}$, $j = 0, 1, 2, ..., n-2$.

(iii) $S'_{j+1}(x_{j+1}) = S'_j(x_{j+1})$, $j = 0, 1, 2, ..., n-2$.

(iv) $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1})$, $j = 0, 1, 2, ..., n-2$.

Then, since for $j$ ther are four unkowns: $a_j, b_j, c_j$, and $d_j$ and there are $n$ such polynomials to be determined, all together there are $4n$ unknowns. However, conditions (i)-(iv) above give only $(4n-2)$ equations: (i) gives $n+1$, and each of (ii)-(iv) gives $n-1$. So, *to completely determine the cubic spline, we must need two more equations.* To obtain these two additional equations, we can invoke *boundary conditions.*

- If the second derivative of $S(x)$ can be approximated, than we can take: (i) $S''(x_0) = S''(x_n) = 0$

**(free or natural boundary)**.

- On the other hand, if only the first derivative can be estimated, we can use the following boundary condition: (ii) $S'(x_0) = f'(x_0)$, and $S'(x_n) = f'(x_n)$ **(clamped boundary)**.

- We will discuss only the clamped cubic spline here.

  ¿From the condition (i), it immediately follows that

$$\boxed{a_j = f_j,\ j = 0, 1, 2, ..., n.}$$

Set $h_j = x_{j+1} - x_j$. With some mathematical manipulations, it can then be shown (see for e.g., Burden and Faires, pp. 144-145), that once $a_j$'s are known, the coefficients $c_j,\ j = 0, 1, 2, ..., n$ can be found by solving the following linear algebraic system of equations:

$$Ax = r,$$

where

A =
$$\begin{bmatrix} 2h_0 & h_0 & 0 & & & 0 \\ h_0 & 2(h_0+h_1) & h_1 & \ddots & & \\ 0 & h_1 & 2(h_1+h_2) & h_2 & \ddots & \\ & \ddots & \ddots & \ddots & \ddots & 0 \\ & & \ddots & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & & & 0 & h_{n-1} & 2h_{n-1} \end{bmatrix}$$

$$x = \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix},\ r = \begin{pmatrix} \frac{3}{h_0}(a_1 - a_0) - 3f'(a) \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \vdots \\ 3f'(b) - \frac{3}{h_{n-1}}(a_n - a_{n-1}) \end{pmatrix}.$$

Once $c_0, c_1, ..., c_n$ are known by solving the above system of equations, the quantities $b_j$ and $d_j$ can be computed as follows:

$$b_j = \frac{(a_{j+1} - a_j)}{h_j} - \frac{h_j}{3}(c_{j+1} + 2c_j), j = n-1, n-2, \cdots, 0$$
$$d_j = \frac{c_{j+1} - c_j}{3h_j}, j = n-1, n-2, \cdots, 0.$$

**Note**:

The matrix $A$ is a symmetric strictly diagonally daminant matrix and can be solved in $O(n)$ flops, as described before. (See the book, *Numerical Linear Algebra and Application*, by **Biswa Nath Datta**, Brooks-Cole Publishing Co., Pacific Grove, California, (1995)).

**Algorithm 2.2 Computing Clamped Cubic Spline**

**Inputs:**

(i) The nodes $x_0, x_1, ..., x_n$.

(ii) The functional values:
$$f(x_i) = f_i,\ i = 0, 1, ..., n.$$
(Note that $a = x_0$ and $b = x_n$).

(iii) $f'(x_0) = f'(a)$ and $f'(x_n) = f'(b)$.

**Outputs:** The coefficients $a_0, ..., a_n$; $b_0, ..., b_n$; $c_0, c_1, ..., c_n$, and $d_0, d_1, ..., d_n$ of the $n$ polynomials $S_0(x), S_1(x), ..., S_{n-1}(x)$ of which the cubic interpolant $S(x)$ is composed.

**Step 1.** For $i = 0, 1, ..., n-1$ do
Set $h_j = x_{j+1} - x_j$

**Step 2.** Compute $a_0, a_1, ..., a_n$:
For $i = 0, 1, ..., n-1$ do
$$a_i = f(x_i),\ i = 0, 1, ..., n.$$

End
**Step 3.** Compute the coefficients $c_0, c_1, ..., c_n$ by solving the system $Ax = r$, where $A, x$, and $r$ are as defined above.

**Step 4.** Compute the coefficients $b_0, ..., b_n$ and $d_0, d_1, ..., d_n$ as above.

# 3 Discrete Least-squares Approximations

Given a set of data points $(x_1, y_1)$, $(x_2, y_2), ..., (x_m, y_m)$, a normal and useful practice in many applications in Statistics, Engineering and other applied sciences is to construct a curve that is considered to be the fits best for the data, in some sense.

Several types of "fits" can be considered. But the one that is used most in applications is the **"least-squares fit"**. Mathematically, the problem is the following:

**Discrete Least-Squares Approximation Problem**

Given a set of data points $(x_k, y_i)$, $i = 1, ..., m$, find an algebraic polynomial $P_n(x) = a_0 + a_1 x + ... + a_n x^n$ $(n < m)$ such that the error in the least-squares sense in minimized; that is, $E = \sum_{i=1}^{m} (y_i - a_0 - a_1 x_i - a_n x_i^n)^2$ is minimum.

For $E$ to be minimum, we must have

$$\frac{\partial E}{\partial a_j} = 0,\ j = 1, ..., n.$$

Now,

$$\frac{\partial E}{\partial a_0} = -2 \sum_{i=1}^{m} (y_i - a_0 - a_1 x_i - - - - a_n x_i^n)$$
$$\frac{\partial E}{\partial a_1} = -\sum_{i=1}^{m} x_i (y_i - a_0 - a_1 x_i - ... - a_n x_i^n)$$
$$\vdots$$
$$\frac{\partial E}{\partial a_n} = -2 \sum_{i=1}^{m} x_i^n (y_i - a_0 - a_1 x_i - ... - a_n x_i^n)$$

Setting these equations to be zero we have

$$m\ a_0 + a_1 \sum_{i=1}^{m} x_i + a_2 \sum_{i=1}^{m} x_i^2 + ... + a_n \sum_{i=1}^{m} x_i^m = \sum_{i=1}^{m} y_i$$
$$a_0 \sum_{i=1}^{m} x_i + a_1 \sum_{i=1}^{m} x_i^2 + ... + a_n \sum_{i=1}^{m} x_i^{n+1} = \sum_{i=1}^{m} x_i y_i$$
$$a_0 \sum_{i=1}^{m} x_i^n + a_1 \sum_{i=1}^{m} x_i^{n+1} + ... + a_n \sum_{i=1}^{m} x_i^{2n} = \sum_{i=1}^{m} x_i^n y_i$$

Set now $\sum_{i=1}^{m} x_i^k = S_k,\ k = 0, 1, ..., 2n$, and denoting the right hand sige entries as $b_0, ..., b_n$, the above equation can be written as:

$$S_0 a_0 + S_1 a_1 + ... + S_n a_n = b_0 \text{ (Note that } \textstyle\sum_{i=1}^{m} x_i^0 - S_0 = m).$$
$$S_1 a_0 + S_2 a_1 + ... + S_{n+1} a_n = b_1$$
$$\vdots$$
$$S_n a_0 + S_{n+1} a_1 ... + S_{2n} a_n = b_n$$

This is a system of $(n + 1)$ equations in $(n + 1)$ unknowns $a_0, a_1, ..., a_n$. This system now can be solved to obtain these $(n + 1)$ unknowns, provided a solution to the system exists. We will not **show that this**

**system has a unique solution if $x_i$'s are distinct.**

The system can be written in the following matrix form:

$$\begin{pmatrix} S_0 & S_1 & ... & S_n \\ S_1 & S_2 & ... & S_{n+1} \\ \vdots & & & \\ S_n & S_{n+1} & ... & S_{2n} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{pmatrix}.$$

or

$$Sa = b$$

where

$$S = \begin{pmatrix} S_0 & S_1 & ... & S_n \\ S_1 & S_2 & ... & S_{n+1} \\ \vdots & & & \\ S_n & S_{n+1} & ... & S_{2n} \end{pmatrix}, \; a = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix}, \; b = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{pmatrix}.$$

Define

$$V = \begin{pmatrix} 1 & x_1 & x_1^2 & ... & x_1^n \\ 1 & x_2 & x_2^2 & ... & x_2^n \\ 1 & x_3 & x_3^2 & ... & x_3^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_m & x_m^2 & ... & x_m^n \end{pmatrix}$$

Then the above system is has the form:

$$V^T V a = b.$$

The matrix $V$ is known as the **Vandermonde matrix,** and it can be shown [**Exercise**] that it has full rank if $x_i$'s are distinct. In this case the matrix $S = V^T V$ is symmetric and positive definite [**Exercise**] and is therefore nonsingular. Thus, if $x_i$'s are distinct, the equation $Sa = b$ has a unique solution.

**Theorem 3.1** ***(Existence and uniqueness of Discrete Least-Squares Solutions).*** *Let $(x_1, y_1)$, $(x_2, y_2)$, ..., $(x_n, y_n)$ be the n distinct points. Then the discrete least-square approximation problem has a unique solution.*