1 Introduction

1.1 Motivation

Before going to study the course, let us try to motivate ourselves by asking the following five **Wh** questions.

1. **What** is Numerical Analysis ?
2. **Where** is Numerical Analysis used ?
3. **When** is Numerical Analysis required ?
4. **Who** requires Numerical Analysis ?
5. **Why** should we learn Numerical Analysis ?

And, finally we ask the following:

How to do Numerical Analysis ?

1. **What** is Numerical Analysis ?

   Numerical Analysis is a branch of mathematics that deals with

   (a) Developing approximation procedure (called Numerical Methods) to solve mathematical problems.

(b) Developing efficient algorithm to implement the above approximation procedure as computer codes (called Implementation).

(c) Analyzing error for the above approximation procedure (a bit of error analysis, convergence, stability).

Numerical Analysis is not about playing with numerical values in computer. It is about mathematical ideas, insights.

What are Numerical methods?

They are algorithms to compute:

(i) approximations of functions,

(ii) approximations of derivatives of functions at some point,

(iii) approximations of integrals of functions in some interval,

(iv) approximations of solution/s of linear/nonlinear single/system of algebraic/differential equation/s.

2. **Where** is Numerical Analysis used ?
(I will put a picture describing whole situation.)

3. **When** is Numerical Analysis required ?

In the following situation one can go for numerical approximations:

(a) The mathematical problem is known to have solution but does not have any method to obtain exact analytical expression.

(b) Mathematical problem is very difficult to handle exactly.

(c) There is a lack of enough data about the inputs for the mathematical problem and therefore not possible to use the available exact methods.

4. **Who** requires Numerical Analysis ?

Everyone working in the field of science and engineering field. For eg:

- Mathematicians,
- Statisticians,
- Physicists,
- Chemists,
- Engineers, etc.

5. **Why** should we learn Numerical Analysis ?

There are at least three reasons for which we need to learn and understand numerical methods.

(a) Learning different numerical methods and their analysis will make a person more familiar with the techniques of developing new numerical methods. This is important because when the available methods are not enough or not efficient for a specific problem to solve.

(b) In many situations, one has more methods for a given problem. Hence choosing an appropriate method is important for producing an accurate result in lesser time.

(c) With a good enough knowledge of numerical methods, one can use it properly and efficiently to get solution of problems and understand what is going wrong if the results obtained are not expected.

Through the rest of the course we will answer 'How to do numerical analysis?'

1.2 Syllabus we are going to cover

(Already I mentioned in the class. This portion will be filled.)

1.3 Prerequisite

(Already I mentioned in the class. This portion will be filled.)

2

Floating Point Approximation of Real Numbers and Error Analysis

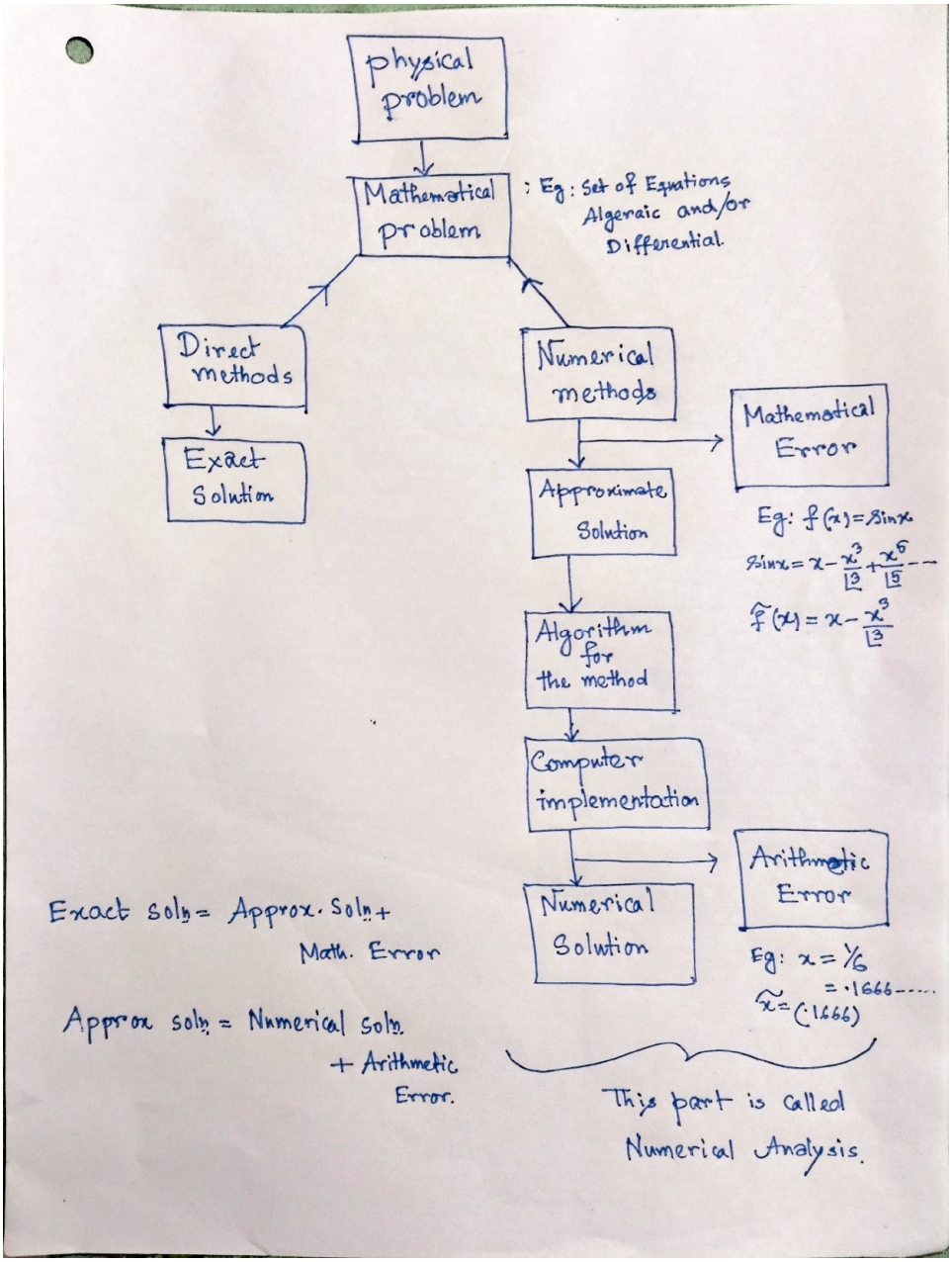
We have known that Numerical Analysis is all about

- Developing approximation procedure called numerical methods, to approximate a solution of a given Mathematical problem (whenever a solution exists),
- Developing algorithms and implement them as computer codes,
- Analyzing error in numerical approximation.

Therefore, the approximate solution, obtained by numerical method, will involve an error which is precisely the difference between the exact solution and the approximate solution. Thus, we have

Exact Solution = Approximate Solution + Error.

This error is called the **Mathematical** error.
In the next step while developing the algorithm and implementing them as computer code, we use a set of numerical values to evaluate the approximate solution obtained in numerical method. After evaluation we obtain a set of numerical values which is called the **numerical solution** to the given Mathematical problem. Due to memory restrictions, a computing device can store only a finite number of digits. Therefore, a real number cannot be stored exactly. Certain approximation needs to be done, and only an



approximate value of the given number is stored in the device. For further calculations, this approximate value is used instead of the exact value of the number. Hence, during the process of computation, the computer introduces a new error, called the **arithmetic error**. Then, we note that

Approximate Solution = Numerical Solution + Arithmetic Error.

Therefore, we have

Exact Solution = Numerical Solution + Mathematical Error + Arithmetic Error.

The Total Error is defined as

Total Error = Mathematical Error + Arithmetic Error.

The error (arithmetic error) involved in the numerical solution when compared to the exact solution can be worse than the mathematical error.
In this chapter, we introduce the floating-point representation of a real number and see two method to obtain floating-point approximation of a given real number. We introduce different types of errors that we come across in numerical analysis and their effects in the computation. At the end of this chapter, we will be familiar with the arithmetic errors, their effect on computed results and some ways to minimize this error in the computation.

2.1 Floating-Point Representation

Let $\beta \in \mathbb{N}$, with $\beta \geqslant 2$. Let $x \in \mathbb{R}$ be a real number. The real number x can be represented in base β as

$$x := (-1)^s \times (.d_1 d_2 \dots d_t d_{t+1} \dots)_\beta \times \beta^e, \tag{2.1.1}$$

where $d_i \in \{0, 1, \dots, \beta - 1\}$ for all $i \in \mathbb{N}$ with $d_1 \neq 0$ or $d_i = 0$, for all $i \in \mathbb{N}$, $s \in \{0, 1\}$, and $e \in \mathbb{Z}$. Here in above

- β is called the **base** or **radix**,
- s is called the **sign**,
- $(.d_1 d_2 \dots d_n d_{n+1} \dots)_\beta$ is called the **mantissa**,
- e is called the **exponent**,

of the given number x. The representation (2.1.1) of a real number is called the **floating-point representation**. The mantissa part can also be written as

$$(.d_1 d_2 \dots d_n d_{n+1} \dots)_\beta = \sum_{i=1}^{\infty} \frac{d_i}{\beta^i}. \tag{2.1.2}$$

We note that the right hand side of (2.1.2) is convergent series in $\mathbb{R}$ and hence is well defined.

Remark 2.1.1. *We note the following.*

- *When $\beta = 2$, the floating-point representation* (2.1.1) *is called the* **binary floating-point representation**.
- *When $\beta = 10$, the floating-point representation* (2.1.1) *is called the* **decimal floating-point representation**.

There are other representation of real numbers. For example **octal** *with $\beta = 8$,* **hexadecimal** *(used in ancient China) with $\beta = 16$,* **vigesimal** *(used in France) with $\beta = 20$ etc. In the course we will consider $\beta = 10$.*

2.1.1 Floating-Point Number System in a Computing Device

Due to memory restrictions, a computing device can store only a finite number of digits in the mantissa. In this section, we introduce the floating-point approximation and discuss how a given real number can be approximated.
A computing device stores a real number with only a finite number of digits in the mantissa. Although different computing devices have different ways of representing the numbers, here we introduce a mathematical form of this representation, which we will use throughout this course.

Definition 2.1.1 (*n*-**digit floating point number**). *Let $\beta \in \mathbb{N}$, with $\beta \geqslant 2$. An n-***digit floating point number** x_f *in* **base** β *is of the form*

$$x_f := (-1)^s \times (.d_1 d_2 \dots d_n)_\beta \times \beta^e \tag{2.1.3}$$

where $d_i \in \{0, 1, \dots, \beta - 1\}$ for all $i = 1, 2, \dots, n$, with $d_1 \neq 0$ or $d_i = 0$, for all $i = 1, 2, \dots, n$, $s \in \{0, 1\}$, $e \in \mathbb{Z}$ is an appropriate exponent. Here

$$(.d_1 d_2 \dots d_n)_\beta = \sum_{i=1}^{n} \frac{d_i}{\beta^i}. \tag{2.1.4}$$

The number of digits n in the mantissa is called the **precision** *or* **length** *of the floating-point number.*

Remark 2.1.2. *We note the following.*

- *When $\beta = 2$, the n-digit floating-point representation* (2.1.1) *is called the n-***digit binary floating-point representation**. *This is used in computer.*
- *When $\beta = 10$, the n-digit floating-point representation* (2.1.1) *is called the n-***digit decimal floating-point representation**. *It is used in daily life computation.*

Example 2.1.1. *The following are examples of real numbers in the decimal floating point representation.*

(i). The real number $x_f = 2347$ is represented in the decimal floating-point representation as

$$x_f = (-1)^0 \times .2347 \times 10^4.$$

In the above, we note that $\beta = 10, s = 0, d_1 = 2, d_2 = 3, d_3 = 4, d_4 = 7, e = 1$.

(ii). The real number $y_f = -0.000739$ is represented in the decimal floating-point representation as

$$y_f = (-1)^1 \times .739 \times 10^{-3}.$$

In the above, we note that $\beta = 10, s = 1, d_1 = 7, d_2 = 3, d_3 = 9, e = -3$.

Remark 2.1.3. *Any computing device has its own memory limitations in storing a real number. In terms of the floating-point representation, these limitations lead to the restrictions in the range of the exponent (e) and the number of digits in the mantissa (n).*

2.1.2 Characterization of Floating-Point Number System in a Computing Device

For a given computing device, there are integers $L, U, n \in \mathbb{Z}$ such that the exponent e is limited to a range

$$L \leqslant e \leqslant U,$$

and n is the precision. Therefore, a floating point number system, denoted by $\mathbb{F}$, is characterized by four integer $\beta, n \in \mathbb{N}$, $L, U \in \mathbb{Z}$ such that

- β= base or radix,
- n= precision,
- L=lower bound of the exponent range,
- U=upper bound of the exponent range.

Any floating point number $x_f \in \mathbb{F}$, can be represented as in (2.1.3).

2.1.3 Properties of $\mathbb{F}$

(i). The n-digit floating-point representation of a number is unique.

(ii). A floating point number system $\mathbb{F}$ is finite and discrete.

(iii). The total numbers in a given floating point number system is

$$2(\beta-1)\beta^{n-1}(U-L+1)+1.$$

Indeed, while counting the numbers, there are

- 2 choices of the sign s,
- $(\beta-1)$ number of choices of leading digit d_1,
- β number of choices of each of the remaining digits $d_2, d_2, \ldots d_n$,
- $(U-L+1)$ number of possible values of the exponent e.

Therefore, in total $2(\beta-1)\beta^{n-1}(U-L+1)$ numbers. Finally, 1 for number zero.

(iv). The mantissa $m(x_f) := (.d_1 d_2 \dots d_n)_\beta = \sum_{i=1}^{n} \frac{d_i}{\beta^i}$ satisfies

$$\frac{1}{\beta} \leqslant m(x_f) < 1.$$

(v). The smallest positive floating point number is

$$\text{UFL} = \beta^{L-1}.$$

Indeed, for the smallest positive floating point number

- the sign, $s = 0$,
- the leading digit, $d_1 = 1$,
- the remaining digits, $d_2 = d_2 = \cdots = d_n = 0$,
- the exponent, $e = L$.

Hence,

$$\text{UFL} = (.10\dots,0)_\beta \beta^L = \frac{1}{\beta}\beta^L = \beta^{L-1}.$$

It is also called Under Flow Level (UFL).

(vi). The largest positive floating point number is

$$\text{OFL} = (1-\beta^{-n})\beta^U.$$

Indeed, for the largest positive floating point number

- the sign, $s = 0$,
- all the digits, $d_1 = d_2 = d_2 = \cdots = d_n = (\beta - 1)$,
- the exponent, $e = U$.

Hence,

$$\begin{aligned}
\text{UFL} &= (.(\beta-1)(\beta-1)\dots(\beta-1))_\beta \beta^U, \\
&= \sum_{i=1}^{n} \frac{(\beta-1)}{\beta^i}\beta^U, \\
&= \frac{(\beta-1)}{\beta}\beta^U \left(1 + \frac{1}{\beta} + \cdots \frac{1}{\beta^{n-1}}\right), \\
&= \frac{(\beta-1)}{\beta}\beta^U \left(\frac{1-\left(\frac{1}{\beta}\right)^n}{1-\left(\frac{1}{\beta}\right)}\right), \\
&= \frac{(\beta-1)}{\beta}\beta^U \frac{(1-\beta^{-n})}{\frac{(\beta-1)}{\beta}},
\end{aligned}$$

$$= (1-\beta^{-n})\beta^{U}.$$

It is also called Under Flow Level (UFL).

(vii). All floating point number $x_f \in \mathbb{F}$ must satisfies

$$\text{UFL} \leqslant |x_f| \leqslant \text{OFL}.$$

(viii). Floating point numbers are not uniformly distributed throughout their range, but they are equally spaced only between successive powers of β.

(ix). If an arithmetic operation leads to a result in which a computed number x satisfies $|x| > \text{OFL}$, then, the calculation may terminate. Then error occurs in the computation. It is called an overflow error.

(x). If an arithmetic operation leads to a result in which a computed number x satisfies $|x| < \text{UFL}$, then, the the floating point representation of x (note that floating point representation of real number will be discussed in subsection 2.1.5) will be set to zero. This also makes error in the computation. It is called an underflow error.

In subsection 2.1.4, we introduce the concept of under and over flow of memory, which is a result of the restriction in the exponent. The restriction on the length of the mantissa is discussed in subsection 2.1.5.

2.1.4 Overflow and Underflow of Memory

- When the value of the exponent e in a floating-point number exceeds the maximum limit of the memory, we encounter the overflow of memory.
- When the value of the exponent e goes below the minimum of the range, then we encounter underflow.

During a computation,

- if some computed number x_f has an exponent $e > L$ (this also occurs if $|x_f| > \text{OFL}$) then we say, the memory **overflow** occurs.
- if $e < U$, (this also occurs if $|x_f| < \text{UFL}$) we say the memory **underflow** occurs.

and

Remark 2.1.4. *We note the following.*

(i).
- *In the case of overflow of memory in a floating-point number, a computer will usually produce meaningless results or simply prints the symbol* inf *or* NaN.

- *When a computation involves an undetermined quantity (like* $0 \times \infty, \infty - \infty, 0/0$*), then the output of the computed value on a computer will be the symbol* NaN *(means 'not a number'). For instance, if* x_f *is a sufficiently large number that results in an overflow of memory when stored on a computing device, and* y_f *is another number that results in an underflow, then their product will be returned as* NaN.

(ii). On the other hand, we feel that the underflow is more serious than overflow in a computation. Because, when underflow occurs, a computer will simply consider the number as zero without any warning. The error is disguised in the computation and computer will not inform you about that whereas for the case of overflow computer will give you warning to modify the computer algorithm(code). If a computation involves such an underflow of memory, then there is a danger of having a large difference between the actual value and the computed value. However, by writing a separate subroutine, one can monitor and get a warning whenever an underflow occurs.

2.1.5 Floating-Point Approximation of Real Numbers

In general, a real number can have infinitely many digits, which a computing device cannot hold in its memory. Rather, each computing device will have its own limitation on the length of the mantissa. If a given real number has infinitely many digits in the mantissa of the floating-point form as in (2.1.1), then the computing device converts this number into an n-digit floating-point form as in (2.1.3). Such an approximation is called the **floating-point approximation** of a real number.
There are many ways to get floating-point approximation of a given real number. Here we introduce two types of floating-point approximations.

Definition 2.1.2 (Chopping)**.** *Let* $x \in \mathbb{R}$ *be a real number given in the floating-point representation* (2.1.1) *as*

$$x = (-1)^s \times (.d_1 d_2 \dots d_n d_{n+1} \dots)_\beta \times \beta^e. \tag{2.1.5}$$

The floating-point approximations of x *using* n**-digit chopping** *is given by*

$$fl(x) = (-1)^s \times (.d_1 d_2 \dots d_n)_\beta \times \beta^e. \tag{2.1.6}$$

The floating-point approximations of x *using* n**-digit rounding** *is given by*

$$fl(x) = \begin{cases} (-1)^s \times (.d_1 d_2 \dots d_n)_\beta \times \beta^e & , \quad 0 \leqslant d_{n+1} < \frac{\beta}{2}, \\ (-1)^s \times (.d_1 d_2 \dots (d_n + 1))_\beta \times \beta^e & , \quad \frac{\beta}{2} \leqslant d_{n+1} < \beta, \end{cases} \tag{2.1.7}$$

where

$$(.d_1d_2\ldots(d_n+1))_\beta = (.d_1d_2\ldots(d_n)_\beta + (.\underbrace{00\ldots0}_{(n-1)\text{- times}}1)_\beta,$$
$$= \sum_{i=1}^{n}\frac{d_i}{\beta^i} + \frac{1}{\beta^n}.$$

Remark 2.1.5. *We note the following. If for a real number x, $d_{n+1} \geqslant \frac{\beta}{2}$ and $d_n = \beta - 1$, then in case of n-th digit rounding $d_n + 1$ in $fl(x)$ would be β. Therefore, we put $d_n = 0$ and add 1 to d_{n-1}. Again, if $d_{n-1} = \beta - 1$, we put $d_{n-1} = 0$ and add 1 to d_{n-2}. We continue further if required until all the digits are exhausted. It may happen that $d_1 = d_2 = \cdots = d_n = \beta - 1$ and $d_{n+1} \geqslant \frac{\beta}{2}$, then for $fl(x)$ we put $d_1 = 1$, all other $d_i = 0$ for $i = 2, 3, \ldots, n$ and increase the exponent by 1, if*

$$x = (-1)^s \times (.\underbrace{(\beta-1)(\beta-1)\ldots(\beta-1)}_{n\text{-times}}d_{n+1})_\beta \times \beta^e, \quad d_{n+1} \geqslant \frac{\beta}{2}$$

then in case of n-th digit rounding

$$fl(x) = (-1)^s \times (.1\ \underbrace{00\ldots0}_{(n-1)\text{-times}})_\beta \times \beta^{e+1}.$$

Example 2.1.2. *(i). The floating-point representation of π is given by*

$$\pi = (-1)^0 \times (.31415926\ldots)_{10} \times 10^1.$$

The floating-point approximation of π using 5-digit chopping is given by

$$fl(\pi) = (-1)^0 \times (.31415)_{10} \times 10^1.$$

The floating-point approximation of π using 5-digit rounding is given by

$$fl(\pi) = (-1)^0 \times (.31416)_{10} \times 10^1.$$

because $d_6 = 9 \geqslant 5$. So in $fl(\pi)$, $d_5 = 5 + 1 = 6$.

Remark 2.1.6. *Most of the modern processors, including Intel, uses IEEE 754 standard format. This format uses 52 bits in mantissa, (64-bit binary representation), 11 bits in exponent and 1 bit for sign. This representation is called the* **double precision**. *number. When we perform a computation without any floating-point approximation, we say that the computation is done using* **infinite precision** *(also called* **exact arithmetic**).

2.1.6 Arithmetic Operations of Floating-Point Approximations

In this subsection, we describe the procedure of performing arithmetic operations using n-digit rounding. The procedure of performing arithmetic operation using n-digit chopping can be done in a similar way.

Procedure of performing arithmetic operations

Let $\odot$ denote any one of the basic arithmetic operations $+$, $-$, $\times$, $\div$. Let $x, y \in \mathbb{R}$ be two real numbers. The process of computing $x \odot y$ using n-digit rounding is as follows.

- **Step 1:** First, we consider the n-digit rounding approximations $\text{fl}(x)$ and $\text{fl}(y)$ of the numbers x and y, respectively.
- **Step 2:**Then, we perform the calculation $\text{fl}(x) \odot \text{fl}(y)$ using exact arithmetic.
- **Step 3:** Finally, we consider the n-digit rounding approximations $\text{fl}(\text{fl}(x) \odot \text{fl}(y))$ of $\text{fl}(x) \odot \text{fl}(y)$.

The result from **Step 3** is the value of$x \odot y$ using n-digit rounding.

Example 2.1.3. *Let us consider the function $f : [0, \infty) \to \mathbb{R}$ given by*

$$f(x) = x(\sqrt{x+1} - \sqrt{x}), \quad x \in \mathbb{R}.$$

We evaluate $f(100000)$ using 6-digit rounding arithmetic. We have

$$f(100000) = 100000(\sqrt{100001} - \sqrt{100000})$$

We see that

$$\sqrt{100001} = 316.229347\cdots = .316229347\cdots \times 10^3,$$

and

$$\sqrt{100000} = 316.227766\cdots = .316227766\cdots \times 10^3.$$

Using 6-digit rounding we note that

$$fl(\sqrt{100001}) = .316229 \times 10^3, \quad \textit{and} \quad fl(\sqrt{100000}) = .316228 \times 10^3.$$

Then,

$$fl(\sqrt{100001}) - fl(\sqrt{100000}) = .000001 \times 10^3 = .1 \times 10^{-2}.$$

Hence,

$$fl(fl(\sqrt{100001}) - fl(\sqrt{100000})) = .1 \times 10^{-2}.$$

Again we see that $fl(100000) = .1 \times 10^6$. *Then,*

$$fl(100000) \times fl(fl(\sqrt{100001}) - fl(\sqrt{100000})) = .1 \times 10^6 \times .1 \times 10^{-2} = .01 \times 10^4 = .1 \times 10^3.$$

Finally, we have

$$fl(f(100000)) = .1 \times 10^3 = 100.$$

Similarly, using 6-digit chopping, the value of $fl(f(100000)) = 200$.

2.2 Types of Errors

The approximate representation of a real number obviously differs from the actual number, whose difference is called an **error**.

Definition 2.2.1 (Errors)**.** *(i). The* **error** *in a computed quantity is defined as*

$$\textbf{Error} = \textbf{True Value} - \textbf{Approximate Value}.$$

(ii). Absolute value of an error is called the **absolute error**.

(iii). The **relative error** *is a measure of the error in relation to the size of the true value as given by*

$$\textbf{Relative Error} = \frac{\textbf{Error}}{\textbf{True Value}}, \qquad \textit{Provided} \quad \textbf{True Value} \neq 0.$$

(iv). Absolute value of the relative error is called the **absolute relative error**.

(v). The **percentage error** *is defined as*

$$\textbf{Percentage Error} = 100 \times |\textbf{Relative Error}|.$$

Remark 2.2.1. *Let* x_A *denote the approximation to the real number* $x \in \mathbb{R}$. *We use the following notations.*

$$E(x_A) := Error(x_A) = x - x_A. \tag{2.2.1}$$

$$E_a(x_A) := Absolute\ Error(x_A) = |x - x_A|. \tag{2.2.2}$$

$$E_r(x_A) := Relative\ Error(x_A) = \frac{x - x_A}{x}, \quad x \neq 0. \tag{2.2.3}$$

$$E_{ar}(x_A) := Absolute\ Relative\ Error(x_A) = \left|\frac{x - x_A}{x}\right|, \quad x \neq 0. \tag{2.2.4}$$

The absolute error has to be understood more carefully because a relatively small difference between two large numbers can appear to be large, and a relatively large

difference between two small numbers can appear to be small. On the other hand, the relative error gives a percentage of the difference between two numbers, which is usually more meaningful as illustrated below.

Example 2.2.1.

2.3 Sources of Errors

- Mathematical modeling of a physical problem.
- Uncertainty in physical data or empirical measurements.
- Errors from previous computations.
- Blunders in arithmetic computations or in methods.
- **Machine Errors**.
- **Mathematical Truncation Error**.

2.4 Machine Errors

In this chapter we will be discussing Machine Errors.

2.4.1 Errors in Floating-point Approximations

With most of real numbers $x \in \mathbb{R}$, we have $\mathrm{fl}(x) \neq x$. We now measure the errors for floating for approximations of real numbers.

Theorem 2.4.1. *Let $x \in \mathbb{R}, x \neq 0$ be such that UFL $\leqslant |fl(x)| \leqslant$ OFL. Then,*

$$E_{ar}(fl(x)) \leqslant \begin{cases} \beta^{-n+1}, & \text{in case of chopping,} \\ \frac{1}{2}\beta^{-n+1}, & \text{in case of rounding.} \end{cases}$$

Proof. Let us assume that x has floating-point representation

$$x = (-1)^s \times (.d_1 d_2 \dots d_n d_{n+1} \dots)_\beta \times \beta^e = (-1)^s \beta^e \sum_{i=1}^{\infty} \frac{d_i}{\beta^i}.$$

First, We note that

$$|x| = \left| (-1)^s \beta^e \left(\sum_{i=1}^{\infty} \frac{d_i}{\beta^i} \right) \right| = \beta^e \sum_{i=1}^{\infty} \frac{d_i}{\beta^i} \geqslant \beta^e \frac{d_1}{\beta} \geqslant \beta^{e-1}. \tag{2.4.1}$$

We prove the result for two cases separately.

Case of chopping : In case of n-digit chopping

$$\text{fl}(x) = (-1)^s \times (.d_1 d_2 \dots d_n)_\beta \times \beta^e = (-1)^s \beta^e \sum_{i=1}^{n} \frac{d_i}{\beta^i}.$$

Hence,

$$\begin{aligned}
E_a(\text{fl}(x)) = |x - \text{fl}(x)| &= \left| (-1)^s \beta^e \left(\sum_{i=1}^{\infty} \frac{d_i}{\beta^i} - \sum_{i=1}^{n} \frac{d_i}{\beta^i} \right) \right|, \\
&= \left| \beta^e \left(\sum_{i=n+1}^{\infty} \frac{d_i}{\beta^i} \right) \right|, \\
&= \beta^e \sum_{i=n+1}^{\infty} \frac{d_i}{\beta^i} \leqslant \beta^e \sum_{i=n+1}^{\infty} \frac{\beta - 1}{\beta^i}, \\
&= \beta^e (\beta - 1) \sum_{i=n+1}^{\infty} \frac{1}{\beta^i}, \\
&= \beta^e (\beta - 1) \frac{\frac{1}{\beta^{n+1}}}{1 - \frac{1}{\beta}} = \frac{\beta^e}{\beta^n} = \beta^{e-n}.
\end{aligned}$$

Using (2.4.1), we have

$$E_{ar}(\text{fl}(x)) = \frac{|x - \text{fl}(x)|}{|x|} \leqslant \frac{\beta^{e-n}}{\beta^{e-1}} = \beta^{-n+1}.$$

Case of rounding : Sub-case: $0 \leqslant d_{n+1} < \frac{\beta}{2}$. We note that

$$\text{fl}(x) = (-1)^s \times (.d_1 d_2 \dots d_n)_\beta \times \beta^e = (-1)^s \beta^e \sum_{i=1}^{n} \frac{d_i}{\beta^i}.$$

Hence,

$$\begin{aligned}
E_a(\text{fl}(x)) = |x - \text{fl}(x)| &= \left| (-1)^s \beta^e \left(\sum_{i=1}^{\infty} \frac{d_i}{\beta^i} - \sum_{i=1}^{n} \frac{d_i}{\beta^i} \right) \right|, \\
&= \left| \beta^e \left(\sum_{i=n+1}^{\infty} \frac{d_i}{\beta^i} \right) \right|, \\
&= \beta^e \sum_{i=n+1}^{\infty} \frac{d_i}{\beta^i} = \beta^e \left\{ \frac{d_{n+1}}{\beta^{n+1}} + \sum_{i=n+2}^{\infty} \frac{d_i}{\beta^i} \right\}, \\
&\leq \beta^e \left\{ \frac{\frac{\beta}{2} - 1}{\beta^{n+1}} + (\beta - 1) \sum_{i=n+2}^{\infty} \frac{1}{\beta^i} \right\} \\
&\left(\because 0 \leqslant d_{n+1} < \frac{\beta}{2} \implies 0 \leqslant d_{n+1} \leqslant \frac{\beta}{2} - 1, \quad \text{and} \quad d_i \leqslant \beta - 1 \right)
\end{aligned}$$

$$
\begin{aligned}
&= \beta^e \left\{ \frac{\beta-2}{2\beta^{n+1}} + (\beta-1)\frac{\frac{1}{\beta^{n+2}}}{1-\frac{1}{\beta}} \right\}, \\
&= \beta^e \left\{ \frac{\beta-2}{2\beta^{n+1}} + \frac{1}{\beta^{n+1}} \right\}, \\
&= \beta^e \frac{\beta-2+2}{2\beta^{n+1}} = \beta^e \frac{\beta}{2\beta^{n+1}} = \frac{1}{2}\beta^{e-n}.
\end{aligned}
$$

Using (2.4.1), we have

$$
E_{ar}(\mathrm{fl}(x)) = \frac{|x-\mathrm{fl}(x)|}{|x|} \leqslant \frac{1}{2}\frac{\beta^{e-n}}{\beta^{e-1}} = \frac{1}{2}\beta^{-n+1}.
$$

Case of rounding : Sub-case: $\frac{\beta}{2} \leqslant d_{n+1} < \beta$. We note that

$$
\mathrm{fl}(x) = (-1)^s \times (.d_1 d_2 \ldots (d_n+1))_\beta \times \beta^e = (-1)^s \beta^e \left(\sum_{i=1}^{n} \frac{d_i}{\beta^i} + \frac{1}{\beta^n} \right).
$$

Hence,

$$
\begin{aligned}
E_a(\mathrm{fl}(x)) = |x-\mathrm{fl}(x)| &= \left| (-1)^s \beta^e \left(\sum_{i=1}^{\infty} \frac{d_i}{\beta^i} - \sum_{i=1}^{n} \frac{d_i}{\beta^i} - \frac{1}{\beta^n} \right) \right|, \\
&= \left| \beta^e \left(\sum_{i=n+1}^{\infty} \frac{d_i}{\beta^i} - \frac{1}{\beta^n} \right) \right|, \\
&\left(\because \sum_{i=n+1}^{\infty} \frac{d_i}{\beta^i} \leq (\beta-1) \sum_{i=n+1}^{\infty} \frac{1}{\beta^i} = (\beta-1)\frac{\frac{1}{\beta^{n+1}}}{1-\frac{1}{\beta}} = \frac{1}{\beta^n} \right) \\
&= \beta^e \left(\frac{1}{\beta^n} - \sum_{i=n+1}^{\infty} \frac{d_i}{\beta^i} \right) \\
&\leq \beta^e \left(\frac{1}{\beta^n} - \frac{d_{n+1}}{\beta^{n+1}} \right) \\
&\left(\because \frac{\beta}{2} \leqslant d_{n+1} \right) \\
&\leq \beta^e \left(\frac{1}{\beta^n} - \frac{\beta}{2\beta^{n+1}} \right) = \frac{1}{2}\beta^{e-n}.
\end{aligned}
$$

Finally, using (2.4.1), we have

$$
E_{ar}(\mathrm{fl}(x)) = \frac{|x-\mathrm{fl}(x)|}{|x|} \leqslant \frac{1}{2}\frac{\beta^{e-n}}{\beta^{e-1}} = \frac{1}{2}\beta^{-n+1}.
$$

This completes the proof. □

2.4.2 Accuracy of Floating-point Approximations

Unit round-off

We now introduce two measures that give a fairly precise idea of the possible accuracy in a floating-point approximation.

Definition 2.4.1 (Unit round-off (**u**)). *The* **unit round-off** *of a computer is denoted by* **u** *and is defined as*

$$\mathbf{u} = \sup\left\{E_{\mathbf{ar}}(fl(\mathbf{x})) : \mathbf{x} \in \mathbb{R},\ OFL \leq |\mathbf{x}| \leq UFL\right\}.$$

i.e., it is the maximum possible value of $E_{ar}(fl(x))$ for any x satisfying the condition as in Theorem 2.4.1. By Theorem 2.4.1 we note that

$$\mathbf{u} = \begin{cases} \beta^{-n+1}, & \text{in case of chopping,} \\ \frac{1}{2}\beta^{-n+1}, & \text{in case of rounding.} \end{cases}$$

Alternative characterization of unit round-off

Definition 2.4.2 (Machine Epsilon(e_{mach})). *The* **machine epsilon** *of a computer is denoted by ε_{mach} and is defined as*

$$\varepsilon_{mach} = \inf\left\{\delta \in \mathbb{F} : \delta > 0, \quad \text{and} \quad fl(1+\delta) > 1\right\}.$$

Theorem 2.4.2.

$$\varepsilon_{mach} = \begin{cases} \beta^{-n+1}, & \text{in case of chopping,} \\ \frac{1}{2}\beta^{-n+1}, & \text{in case of rounding.} \end{cases}$$

Proof. See book: Atkinson: Introduction ot Numerical Analysis, 2ed., p-15. □

Remark 2.4.1. *From the definition of ε_{mach}, we note that for any $\delta \in \mathbb{F}$ with $0 < \delta < \varepsilon_{mach}$ implies $fl(1+\delta) = 1$, i.e., in computer the number $1+\delta$ is identical with 1.*

Remark 2.4.2. *we note that $\varepsilon_{mach} \neq UFL$. ε_{mach} is determined by the precision i.e., the number of digits in the mantissa field of Floating-point system, whereas the UFL is determined by the number of lower bound in the exponent field.*

Example 2.4.1.

Maximal Accuracy

Definition 2.4.3 (Maximal Accuracy). *The maximal accuracy in a floating-point representation is denoted my M_{acc} and is defined as*

$$M_{acc} = \sup\left\{m \in \mathbb{Z} : m \geqslant 0, \quad \text{and} \quad fl(m) = m\right\}.$$

Remark 2.4.3. *From the definition of maximal accuracy, we note that for any* $fl(M_{acc}+1) \neq M_{acc}+1$.

Theorem 2.4.3.

$$M_{acc} = \beta^n.$$

Proof. See book: Atkinson: Introduction ot Numerical Analysis, 2ed., p-16. □

2.4.3 Loss of Significance

In place of relative error, we often use the concept of significant digits that is closely related to relative error.

Definition 2.4.4 (Significant digits). *Let us consider a floating-point number system with base* $\beta \in \mathbb{N}$ *with* $\beta \geqslant 2$. *For* $x \in \mathbb{R}$, *let* x_A *be an approximation of* x. *Let*

$$s = \sup\left\{k \in \mathbb{Z} : \beta^k \leqslant |x|\right\} \quad \text{and} \quad r = \sup\left\{t \in \mathbb{N} : |x - x_A| \leq \frac{1}{2}\beta^{s+1-t}\right\}$$

Then, we say x_A *has* r *significant digits to approximate* x.

Example 2.4.2. *We find significant digits for the following approximations.*

(i). $x = \frac{1}{3} = .3333\ldots$ *and* $x_A = .333$. *We note that*

$$10^{-1} = .1 < x, \quad \textit{hence,} \quad s = -1$$

and

$$\begin{aligned} |x - x_A| = (.0003333\ldots) &\leqslant (.3333\ldots) \times 10^{-3}, \\ &= \frac{1}{2} \times (.6666\ldots) \times 10^{-3}, \\ &< \frac{1}{2} \times 10^{-3} = \frac{1}{2} \times 10^{-1+1-3} \end{aligned}$$

Therefore, x_A *has* 3 *significant digits to approximate* x.

(ii). $x = 23.496$ *and* $x_A = 23.494$ *We note that*

$$10^1 < x, \quad \textit{hence,} \quad s = 1,$$

and

$$\begin{aligned} |x - x_A| = (.002) &= .2 \times 10^{-2}, \\ &= \frac{1}{2} \times .4 \times 10^{-2}, \end{aligned}$$

$$< \frac{1}{2} \times 10^{-2} = \frac{1}{2} \times 10^{1+1-4}$$

Therefore, x_A has 4 significant digits to approximate x.

(iii). $x = 0.02138$ *and* $x_A = 0.02144$ *We note that*

$$10^{-2} < x, \quad \textit{hence,} \quad s = -2,$$

and

$$\begin{aligned} |x - x_A| = (.00006) &= .6 \times 10^{-4}, \\ &= \frac{1}{2} \times 1.2 \times 10^{-4}, \\ &= \frac{1}{2} \times .12 \times 10^{-3}, \\ &< \frac{1}{2} \times 10^{-3} = \frac{1}{2} \times 10^{-2+1-2} \end{aligned}$$

Therefore, x_A has 2 significant digits to approximate x.

Remark 2.4.4. *(i). We note that*

$$E_{ar}(x_A) = \frac{|x - x_A|}{|x|} \leqslant \frac{1}{2} \frac{\beta^{s-r+1}}{\beta^s} = \frac{1}{2} \beta^{1-r}.$$

Hence, absolute relative error decreases with an increase in the number of significant digits.

(ii). Number of significant digits roughly measures the number of leading non-zero digits of x_A that are correct relative to the corresponding digits in the true value x. However, this is **not a precise way** *to get the number of significant digits as we have seen in the above examples.*

(iii). The role of significant digits in numerical calculations is very important in the sense that the loss of significant digits may result in drastic amplification of the relative error as illustrated in the following example.

Example 2.4.3 (Loss of Significance)**.** *(i). Let us consider two real numbers*

$$x = 7.6545428 = .76545428 \times 10^1, \quad y = 7.6544201 = .76544201 \times 10^1.$$

Let

$$x_A = 7.6545421 = .76545421 \times 10^1, \quad y_A = 7.6544200 = .76544200 \times 10^1$$