CHAPTER 3

Arithmetic Error Analysis

Numerical analysis deals with developing methods, called *numerical methods*, to approximate a solution of a given Mathematical problem (whenever a solution exists). The approximate solution obtained by a method involves an error, which we call the *mathematical error*, and is precisely the difference between the exact solution and the approximate solution. Thus, we have

$$\text{Exact Solution} = \text{Approximate Solution} + \text{Mathematical Error}.$$

The study of numerical methods is incomplete if we don't develop algorithms and implement the algorithms as computer codes. The outcome of the computer code is a set of numerical values to the approximate solution obtained using a numerical method. Such a set of numerical values is called the *numerical solution* to the given Mathematical problem. During the process of computation, the computer introduces a new error, called the *arithmetic error* , and we have

$$\text{Approximate Solution} = \text{Numerical Solution} + \text{Arithmetic Error}.$$

The error involved in the numerical solution when compared to the exact solution can be worser than the mathematical error and is now given by

$$\text{Exact Solution} = \text{Numerical Solution} + \text{Mathematical Error} + \text{Arithmetic Error}.$$

The *Total Error* is defined as

$$\text{Total Error} = \text{Mathematical Error} + \text{Arithmetic Error}.$$

A digital calculating device can hold only a finite number of digits because of memory restrictions. Therefore, a number cannot be stored exactly. Certain approximation needs to be done, and only an approximate value of the given number will finally be stored in the device. For further calculations, this approximate value is used instead of the exact value of the number. This is the source of arithmetic error.

In this chapter, we introduce the floating-point representation of a real number and illustrate a few ways to obtain floating-point approximation of a given real number. We further 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.

3.1 Floating-Point Representation

Let $\beta \in \mathbb{N}$ and $\beta \geq 2$. Any real number can be represented exactly in ***base*** β as

$$(-1)^s \times (.d_1 d_2 \cdots d_n d_{n+1} \cdots)_\beta \times \beta^e, \tag{3.1}$$

where $d_i \in \{0, 1, \cdots, \beta - 1\}$ with $d_1 \neq 0$ or $d_1 = d_2 = d_3 = \cdots = 0$, $s = 0$ or 1, and an appropriate integer e called the ***exponent***. Here

$$(.d_1 d_2 \cdots d_n d_{n+1} \cdots)_\beta = \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_n}{\beta^n} + \frac{d_{n+1}}{\beta^{n+1}} + \cdots \tag{3.2}$$

is a β-fraction called the ***mantissa***, s is called the ***sign*** and the number β is called the ***radix***. The representation (3.1) of a real number is called the ***floating-point representation***.

Remark 3.1.1.

When $\beta = 2$, the floating-point representation (3.1) is called the ***binary floating-point representation*** and when $\beta = 10$, it is called the ***decimal floating-point representation***.

Note Throughout this course, we always take $\beta = 10$.

3.1.1 Floating-Point Approximation

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.

Although different computing devices have different ways of representing numbers, here we introduce a mathematical form of this representation, which we will use throughout this course.

Definition 3.1.2 [n-Digit Floating-point Number].

Let $\beta \in \mathbb{N}$ and $\beta \geq 2$. An ***n-digit floating-point number*** in ***base*** β is of the form

$$(-1)^s \times (.d_1 d_2 \cdots d_n)_\beta \times \beta^e \tag{3.3}$$

where

$$(.d_1 d_2 \cdots d_n)_\beta = \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_n}{\beta^n} \tag{3.4}$$

where $d_i \in \{0, 1, \cdots, \beta - 1\}$ with $d_1 \neq 0$ or $d_1 = d_2 = d_3 = \cdots = 0$, $s = 0$ or 1, and an appropriate exponent e.

Remark 3.1.3.

When $\beta = 2$, the n-digit floating-point representation (3.3) is called the ***n-digit binary floating-point representation*** and when $\beta = 10$, it is called the ***n-digit decimal floating-point representation***.

Example 3.1.4.

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

1. The real number $x = 6.238$ is represented in the decimal floating-point representation as
$$6.238 = (-1)^0 \times 0.6238 \times 10^1,$$
in which case, we have $s = 0$, $\beta = 10$, $e = 1$, $d_1 = 6$, $d_2 = 2$, $d_3 = 3$ and $d_4 = 8$.

2. The real number $x = -0.0014$ is represented in the decimal floating-point representation as
$$x = (-1)^1 \times 0.14 \times 10^{-2}.$$
Here $s = 1$, $\beta = 10$, $e = -2$, $d_1 = 1$ and $d_2 = 4$.

Remark 3.1.5.

The floating-point representation of the number $1/3$ is

$$\frac{1}{3} = 0.33333\cdots = (-1)^0 \times (0.33333\cdots)_{10} \times 10^0.$$

An n-digit decimal floating-point representation of this number has to contain only n digits in its mantissa. Therefore, $1/3$ cannot be represented as an n-digit floating-point number.

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 number of digits in the mantissa (n) and the range of the exponent (e). In Section 3.1.2, 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 Section 3.1.3.

3.1.2 Underflow and Overflow 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, whereas when this value goes below the minimum of the range, then we encounter underflow. Thus, for a given computing device, there are integers m and M such that the exponent e is limited to a range

$$m \leq e \leq M. \tag{3.5}$$

During the calculation, if some computed number has an exponent $e > M$ then we say, the memory ***overflow*** occurs and if $e < m$, we say the memory ***underflow*** occurs.

Remark 3.1.6.

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 your 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 is a sufficiently large number that results in an overflow of memory when stored on a computing device, and x is another number that results in an underflow, then their product will be returned as `NaN`.

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. However, by writing a separate subroutine, one can monitor and get a warning whenever an underflow occurs.

Example 3.1.7 [Overflow].

Run the following MATLAB code on a computer with 32-bit intel processor:

```
i=308.25471;
fprintf('%f  %f\n',i,10^i);
i=308.25472;
fprintf('%f  %f\n',i,10^i);
```

We see that the first print command shows a meaningful (but very large) number, whereas the second print command simply prints `inf`. This is due to the overflow of memory while representing a very large real number.

Also try running the following code on the MATLAB:

```
i=308.25471;
fprintf('%f  %f\n',i,10^i/10^i);
i=308.25472;
fprintf('%f  %f\n',i,10^i/10^i);
```

The output will be

```
308.254710  1.000000
308.254720  NaN
```

If your computer is not showing `inf` for `i` = `308.25472`, try increasing the value of `i` till you get `inf`.

Example 3.1.8 [Underflow].

Run the following MATLAB code on a computer with 32-bit intel processor:

```
j=-323.6;
 if(10^j>0)
     fprintf('The given number is greater than zero\n');
 elseif (10^j==0)
     fprintf('The given number is equal to zero\n');
 else
     fprintf('The given number is less than zero\n');
 end
```

The output will be

```
The given number is greater than zero
```

When the value of `j` is further reduced slightly as shown in the following program

```
j=-323.64;
if(10^j>0)
    fprintf('The given number is greater than zero\n');
elseif (10^j==0)
    fprintf('The given number is equal to zero\n');
else
    fprintf('The given number is less than zero\n');
end
```

the output shows

```
The given number is equal to zero
```

If your computer is not showing the above output, try decreasing the value of `j` till you get the above output.

In this example, we see that the number $10^{-323.64}$ is recognized as zero by the computer. This is due to the underflow of memory. Note that multiplying any large number by this number will give zero as answer. 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.

3.1.3 Chopping and Rounding a Number

The number of digits in the mantissa, as given in Definition 3.1.2, is called the ***precision*** or *length* of the floating-point number. 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 (or sufficiently large number of) digits in the mantissa of the floating-point form as in (3.1), then the computing device converts this number into an n-digit floating-point form as in (3.3). 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 approximation.

Definition 3.1.9 [Chopped and Rounded Numbers].

Let x be a real number given in the floating-point representation (3.1) as

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

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

$$\text{fl}(x) = (-1)^s \times (.d_1 d_2 \cdots d_n)_\beta \times \beta^e. \tag{3.6}$$

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

$$\text{fl}(x) = \begin{cases} (-1)^s \times (.d_1 d_2 \cdots d_n)_\beta \times \beta^e & , \ 0 \le d_{n+1} < \frac{\beta}{2} \\ (-1)^s \times (.d_1 d_2 \cdots (d_n + 1))_\beta \times \beta^e & , \ \frac{\beta}{2} \le d_{n+1} < \beta \end{cases}, \tag{3.7}$$

where

$$(-1)^s \times (.d_1 d_2 \cdots (d_n+1))_\beta \times \beta^e := (-1)^s \times \left((.d_1 d_2 \cdots d_n)_\beta + (.\underbrace{0\,0 \cdots 0}_{(n-1)-\text{times}}\ 1)_\beta \right) \times \beta^e.$$

Note As already mentioned, throughout this course, we always take $\beta = 10$. **Also, we do not assume any restriction on the exponent** $e \in \mathbb{Z}$.

Example 3.1.10.

The floating-point representation of π is given by

$$\pi = (-1)^0 \times (.31415926 \cdots) \times 10^1.$$

The floating-point approximation of π using *five-digit chopping* is

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

which is equal to 3.1415. Since the sixth digit of the mantissa in the floating-point representation of π is a 9, the floating-point approximation of π using *five-digit rounding* is given by

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

which is equal to 3.1416.

Remark 3.1.11.

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*).

3.1.4 Arithmetic Using n-Digit Rounding and Chopping

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 is done in a similar way.

Let $\odot$ denote any one of the basic arithmetic operations '$+$', '$-$', '$\times$' and '$\div$'. Let x and y be real numbers. The process of computing $x \odot y$ *using n-digit rounding* is as follows.

Step 1: Get the n-digit rounding approximation $\text{fl}(x)$ and $\text{fl}(y)$ of the numbers x and y, respectively.

Step 2: Perform the calculation $\text{fl}(x) \odot \text{fl}(y)$ using exact arithmetic.

Step 3: Get the n-digit rounding approximation $\text{fl}\big(\text{fl}(x) \odot \text{fl}(y)\big)$ 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 3.1.12.

Consider the function

$$f(x) = x\Big(\sqrt{x+1} - \sqrt{x}\Big).$$

Let us evaluate $f(100000)$ using a six-digit rounding. We have

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

The evaluation of $\sqrt{100001}$ using six-digit rounding is as follows.

$$\begin{aligned}\sqrt{100001} &\approx 316.229347\\ &= 0.316229347 \times 10^3.\end{aligned}$$

The six-digit rounded approximation of 0.316229347×10^3 is given by 0.316229×10^3. Therefore,

$$\text{fl}(\sqrt{100001}) = 0.316229 \times 10^3.$$

Similarly,

$$\text{fl}(\sqrt{100000}) = 0.316228 \times 10^3.$$

The six-digit rounded approximation of the difference between these two numbers is

$$\text{fl}\Big(\text{fl}(\sqrt{100001}) - \text{fl}(\sqrt{100000})\Big) = 0.1 \times 10^{-2}.$$

Finally, we have

$$\begin{aligned} \text{fl}(100000) \times (0.1 \times 10^{-2}) &= (0.1 \times 10^6) \times (0.1 \times 10^{-2}) \\ &= 0.1 \times 10^3. \end{aligned}$$

Thus, the value of $f(100000)$ using six-digit rounding is 100. Similarly, we can see that using six-digit chopping, the value of $f(100000)$ is 200.

Definition 3.1.13 [Machine Epsilon].

The ***machine epsilon*** of a computer is the smallest positive floating-point number δ such that

$$\text{fl}(1 + \delta) > 1.$$

For any floating-point number $\hat{\delta} < \delta$, we have $\text{fl}(1 + \hat{\delta}) = 1$, and $1 + \hat{\delta}$ and 1 are identical within the computer's arithmetic.

Remark 3.1.14.

From Example 3.1.8, it is clear that the machine epsilon for a 32-bit intel processor lies between the numbers $10^{-323.64}$ and $10^{-323.6}$. It is possible to get the exact value of this number, but it is no way useful in our present course, and so we will not attempt to do this here.

3.2 Types of Errors

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

Definition 3.2.1 [Errors].

1. The ***error*** in a computed quantity is defined as
$$Error = True\ Value - Approximate\ Value.$$
2. Absolute value of an error is called the ***absolute error***.
3. The ***relative error*** is a measure of the error in relation to the size of the true value as given by
$$Relative\ Error = \frac{Error}{True\ Value}.$$
Here, we assume that the true value is non-zero.
4. The ***percentage error*** is defined as
$$Percentage\ Error = 100 \times \Big|Relative\ Error\Big|.$$

Remark 3.2.2.

Let x_A denote the approximation to the real number x. We use the following notations:

$$E(x_A) := \text{Error}(x_A) = x - x_A. \tag{3.8}$$
$$E_a(x_A) := \text{Absolute Error}(x_A) = |E(x_A)| \tag{3.9}$$
$$E_r(x_A) := \text{Relative Error}(x_A) = \frac{E(x_A)}{x}, \quad x \neq 0. \tag{3.10}$$

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 3.2.3.

Let $x = 100000$, $x_A = 99999$, $y = 1$ and $y_A = 1/2$. We have
$$E_a(x_A) = 1, \quad E_a(y_A) = \frac{1}{2}.$$
Although $E_a(x_A) > E_a(y_A)$, we have
$$E_r(x_A) = 10^{-5}, \quad E_r(y_A) = \frac{1}{2}.$$
Hence, in terms of percentage error, x_A has only $10^{-3}\%$ error when compared to x whereas y_A has 50% error when compared to y.

The errors defined above are between a given number and its approximate value. Quite often we also approximate a given function by another function that can be handled more easily. For instance, a sufficiently differentiable function can be approximated using Taylor's theorem (Theorem 1.1.2). The error between the function value and the value obtained from the corresponding Taylor's polynomial is defined as *truncation error* as defined in Section 1.2.

3.3 Loss of Significance

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

Definition 3.3.1 [Significant β-Digits].

Let β be a radix and $x \neq 0$. If x_A is an approximation to x, then we say that x_A approximates x to r ***significant β-digits*** if r is the largest non-negative integer such that

$$\frac{|x - x_A|}{|x|} \leq \frac{1}{2}\beta^{-r+1}. \tag{3.11}$$

We also say x_A *has r significant β-digits in x*.

Note When $\beta = 10$, we refer significant 10-digits by significant digits.

Example 3.3.2.

1. For $x = 1/3$, the approximate number $x_A = 0.333$ has three significant digits, since
$$\frac{|x - x_A|}{|x|} = 0.001 < 0.005 = 0.5 \times 10^{-2}.$$
Thus, $r = 3$.

2. For $x = 0.02138$, the approximate number $x_A = 0.02144$ has three significant digits, since
$$\frac{|x - x_A|}{|x|} \approx 0.0028 < 0.005 = 0.5 \times 10^{-2}.$$
Thus, $r = 3$.

3. For $x = 0.02132$, the approximate number $x_A = 0.02144$ has two significant

digits, since

$$\frac{|x - x_A|}{|x|} \approx 0.0056 < 0.05 = 0.5 \times 10^{-1}.$$

Thus, $r = 2$.

4. For $x = 0.02138$, the approximate number $x_A = 0.02149$ has two significant digits, since
$$\frac{|x - x_A|}{|x|} \approx 0.0051 < 0.05 = 0.5 \times 10^{-1}.$$
Thus, $r = 2$.

5. For $x = 0.02108$, the approximate number $x_A = 0.0211$ has three significant digits, since
$$\frac{|x - x_A|}{|x|} \approx 0.0009 < 0.005 = 0.5 \times 10^{-2}.$$
Thus, $r = 3$.

6. For $x = 0.02108$, the approximate number $x_A = 0.02104$ has three significant digits, since
$$\frac{|x - x_A|}{|x|} \approx 0.0019 < 0.005 = 0.5 \times 10^{-2}.$$
Thus, $r = 3$.

Remark 3.3.3.

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 it is evident from the above examples.

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 3.3.4.

Let us consider two real numbers

$$x = 7.6545428 = 0.76545428 \times 10^1 \text{ and } \quad y = 7.6544201 = 0.76544201 \times 10^1.$$

The numbers

$$x_A = 7.6545421 = 0.76545421 \times 10^1 \text{ and } \quad y_A = 7.6544200 = 0.76544200 \times 10^1$$

are approximations to x and y, correct to seven and eight significant digits, respectively. The exact difference between x_A and y_A is

$$z_A = x_A - y_A = 0.12210000 \times 10^{-3}$$

and the exact difference between x and y is

$$z = x - y = 0.12270000 \times 10^{-3}.$$

Therefore,

$$\frac{|z - z_A|}{|z|} \approx 0.0049 < 0.5 \times 10^{-2}$$

and hence z_A has only three significant digits with respect to z. Thus, we started with two approximate numbers x_A and y_A which are correct to seven and eight significant digits with respect to x and y, respectively. But their difference z_A has only three significant digits with respect to z. Hence, there is a loss of significant digits in the process of subtraction. A simple calculation shows that

$$E_r(z_A) \approx 53581 \times E_r(x_A).$$

Similarly, we have

$$E_r(z_A) \approx 375067 \times E_r(y_A).$$

Loss of significant digits is therefore dangerous. The loss of significant digits in the process of calculation is referred to as *loss of significance*.

The loss of significant digits results in accumulation of error during evaluation of a function value at a given point as illustrated in the following example.

Example 3.3.5.

Consider the function

$$f(x) = x(\sqrt{x+1} - \sqrt{x}).$$

From Example 3.1.12, the value of $f(100000)$ using six-digit rounding is 100, whereas the true value is 158.113. There is a drastic error in the value of the function, which is due to the loss of significant digits. It is evident that as x increases, the terms $\sqrt{x+1}$ and $\sqrt{x}$ comes closer to each other and therefore loss of significance in their computed value increases.

Such a loss of significance can be avoided by rewriting the given expression of f in such a way that subtraction of near-by non-negative numbers is avoided. For instance,

we can rewrite the expression of the function f as

$$f(x) = \frac{x}{\sqrt{x+1} + \sqrt{x}}.$$

With this new form of f, we obtain $f(100000) = 158.114000$ using six-digit rounding.

Example 3.3.6.

Consider evaluating the function

$$f(x) = 1 - \cos x$$

near $x = 0$. Since $\cos x \approx 1$ for x near zero, there will be loss of significance in the process of evaluating $f(x)$ for x near zero. So, we have to use an alternative formula for $f(x)$ such as

$$\begin{aligned} f(x) &= 1 - \cos x \\ &= \frac{1 - \cos^2 x}{1 + \cos x} \\ &= \frac{\sin^2 x}{1 + \cos x} \end{aligned}$$

which can be evaluated quite accurately for small x.

Remark 3.3.7.

Unlike the above examples, we may not be able to always write an equivalent formula of a given function to avoid loss of significance in the evaluation. In such cases, we have to go for a suitable approximation of the given function by other functions, for instance Taylor's polynomial of desired degree, that do not involve loss of significance.

3.4 Propagation of Relative Error in Arithmetic Operations

Once an error is committed, it affects subsequent results as this error propagates through subsequent calculations. We first study how the results are affected by using approximate numbers instead of actual numbers and then will take up the effect of errors on function evaluation in the next section.

Let x_A and y_A denote the approximate numbers used in the calculation, and let x_T and y_T be the corresponding true values. We will now see how relative error propagates with the four basic arithmetic operations.

3.4.1 Addition and Subtraction

Let $x_T = x_A + \epsilon$ and $y_T = y_A + \eta$ be positive real numbers. The relative error $E_r(x_A \pm y_A)$ is given by

$$\begin{aligned} E_r(x_A \pm y_A) &= \frac{(x_T \pm y_T) - (x_A \pm y_A)}{x_T \pm y_T} \\ &= \frac{(x_T \pm y_T) - (x_T - \epsilon \pm (y_T - \eta))}{x_T \pm y_T} \end{aligned}$$

Upon simplification, we get

$$E_r(x_A \pm y_A) = \frac{\epsilon \pm \eta}{x_T \pm y_T}. \tag{3.12}$$

The above expression shows that there can be a drastic increase in the relative error during subtraction of two approximate numbers whenever $x_T \approx y_T$ as we have witnessed in Example 3.3.4 and Example 3.3.5. On the other hand, it is easy to see from (3.12) that

$$|E_r(x_A + y_A)| \leq |E_r(x_A)| + |E_r(y_A)|,$$

which shows that the relative error propagates slowly in addition. Note that such an inequality in the case of subtraction is not possible.

3.4.2 Multiplication

The relative error $E_r(x_A \times y_A)$ is given by

$$\begin{aligned} E_r(x_A \times y_A) &= \frac{(x_T \times y_T) - (x_A \times y_A)}{x_T \times y_T} \\ &= \frac{(x_T \times y_T) - \big((x_T - \epsilon) \times (y_T - \eta)\big)}{x_T \times y_T} \\ &= \frac{\eta x_T + \epsilon y_T - \epsilon\eta}{x_T \times y_T} \\ &= \frac{\epsilon}{x_T} + \frac{\eta}{y_T} - \left(\frac{\epsilon}{x_T}\right)\left(\frac{\eta}{y_T}\right) \end{aligned}$$

Thus, we have

$$E_r(x_A \times y_A) = E_r(x_A) + E_r(y_A) - E_r(x_A)E_r(y_A). \tag{3.13}$$

Taking modulus on both sides, we get

$$|E_r(x_A \times y_A)| \leq |E_r(x_A)| + |E_r(y_A)| + |E_r(x_A)|\,|E_r(y_A)|$$

Note that when $|E_r(x_A)|$ and $|E_r(y_A)|$ are very small, then their product is negligible when compared to $|E_r(x_A)| + |E_r(y_A)|$. Therefore, the above inequality reduces to

$$|E_r(x_A \times y_A)| \lessapprox |E_r(x_A)| + |E_r(y_A)|,$$

which shows that the relative error propagates slowly in multiplication.

3.4.3 Division

The relative error $E_r(x_A/y_A)$ is given by

$$\begin{aligned} E_r(x_A/y_A) &= \frac{(x_T/y_T) - (x_A/y_A)}{x_T/y_T} \\ &= \frac{x_T(y_T-\eta) - y_T(x_T-\epsilon)}{x_T(y_T-\eta)} \\ &= \frac{\epsilon y_T - \eta x_T}{x_T(y_T-\eta)} \\ &= \frac{y_T}{y_T-\eta}(E_r(x_A) - E_r(y_A)) \end{aligned}$$

Thus, we have

$$E_r(x_A/y_A) = \frac{1}{1-E_r(y_A)}\big(E_r(x_A) - E_r(y_A)\big). \tag{3.14}$$

The above expression shows that the relative error increases drastically during division whenever $E_r(y_A) \approx 1$. This means that y_A has 100% error when compared to y, which is very unlikely because we always expect the relative error to be very small, ie., very close to zero. In this case the right hand side is approximately equal to $E_r(x_A) - E_r(y_A)$. Hence, we have

$$|E_r(x_A/y_A)| \lessapprox |E_r(x_A) - E_r(y_A)| \leq |E_r(x_A)| + |E_r(y_A)|,$$

which shows that the relative error propagates slowly in division.

3.4.4 Total Error

In Subsection 3.1.4, we discussed the procedure of performing arithmetic operations using n-digit floating-point approximation. The computed value $\text{fl}\big(\text{fl}(x) \odot \text{fl}(y)\big)$ involves an error (when compared to the exact value $x \odot y$) which comprises of

1. Error in $\text{fl}(x)$ and $\text{fl}(y)$ due to n-digit rounding or chopping of x and y, respectively, and
2. Error in $\text{fl}\big(\text{fl}(x) \odot \text{fl}(y)\big)$ due to n-digit rounding or chopping of the number $\text{fl}(x) \odot \text{fl}(y)$.

The ***total error*** is defined as

$$\begin{aligned} (x \odot y) - \text{fl}\big(\text{fl}(x) \odot \text{fl}(y)\big) = &\big\{(x \odot y) - \big(\text{fl}(x) \odot \text{fl}(y)\big)\big\} \\ &+ \big\{\big(\text{fl}(x) \odot \text{fl}(y)\big) - \text{fl}\big(\text{fl}(x) \odot \text{fl}(y)\big)\big\}, \end{aligned}$$

in which the first term on the right hand side is called the ***propagated error*** and the second term is called the ***floating-point error***. The ***relative total error*** is obtained by dividing both sides of the above expression by $x \odot y$.

Example 3.4.1.

Consider evaluating the integral

$$I_n = \int_0^1 \frac{x^n}{x+5}\, dx, \quad \text{for} \quad n = 0, 1, \cdots, 30.$$

The value of I_n can be obtained in two different iterative processes, namely,

1. The *forward iteration* for evaluating I_n is given by

$$I_n = \frac{1}{n} - 5I_{n-1}, \quad I_0 = \ln(6/5).$$

2. The *backward iteration* for evaluating I_{n-1} is given by

$$I_{n-1} = \frac{1}{5n} - \frac{1}{5}I_n, \quad I_{30} = 0.54046330 \times 10^{-2}.$$

The following table shows the computed value of I_n using both iterative formulas along with the exact value. The numbers are computed using MATLAB using double precision arithmetic and the final answer is rounded to 6 digits after the decimal point.

n	Forward Iteration	Backward Iteration	Exact Value
1	0.088392	0.088392	0.088392
5	0.028468	0.028468	0.028468
10	0.015368	0.015368	0.015368
15	0.010522	0.010521	0.010521
20	0.004243	0.007998	0.007998
25	11.740469	0.006450	0.006450
30	-36668.803026	Not Computed	0.005405

Clearly the backward iteration gives exact value up to the number of digits shown, whereas forward iteration tends to increase error and give entirely wrong values. This is due to the propagation of error from one iteration to the next iteration. In forward iteration, the total error from one iteration is magnified by a factor of 5 at the next iteration. In backward iteration, the total error from one iteration is divided by 5 at the next iteration. Thus, in this example, with each iteration, the total error tends to increase rapidly in the forward iteration and tends to increase very slowly in the backward iteration.

3.5 Propagation of Relative Error in Function Evaluation

For a given function $f : \mathbb{R} \to \mathbb{R}$, consider evaluating $f(x)$ at an approximate value x_A rather than at x. The question is **how well does $f(x_A)$ approximate $f(x)$?** To answer this question, we compare $E_r(f(x_A))$ with $E_r(x_A)$.

Assume that f is a C^1 function. Using the mean-value theorem, we get

$$f(x) - f(x_A) = f'(\xi)(x - x_A),$$

where ξ is an unknown point between x and x_A. The relative error in $f(x_A)$ when compared to $f(x)$ is given by

$$E_r(f(x_A)) = \frac{f'(\xi)}{f(x)}(x - x_A).$$

Thus, we have

$$E_r(f(x_A)) = \left(\frac{f'(\xi)}{f(x)}x\right) E_r(x_A). \tag{3.15}$$

Since x_A and x are assumed to be very close to each other and ξ lies between x and x_A, we may make the approximation

$$f(x) - f(x_A) \approx f'(x)(x - x_A).$$

In view of (3.15), we have

$$E_r(f(x_A)) \approx \left(\frac{f'(x)}{f(x)}x\right) E_r(x_A). \tag{3.16}$$

The expression inside the bracket on the right hand side of (3.16) is the amplification factor for the relative error in $f(x_A)$ in terms of the relative error in x_A. Thus, this expression plays an important role in understanding the propagation of relative error in evaluating the function value $f(x)$ and hence motivates the following definition.

Definition 3.5.1 [Condition Number of a Function].

The ***condition number*** of a continuously differentiable function f at a point $x = c$ is given by

$$\left|\frac{f'(c)}{f(c)}c\right|. \tag{3.17}$$

The condition number of a function at a point $x = c$ can be used to decide whether the evaluation of the function at $x = c$ is well-conditioned or ill-conditioned depending on whether this condition number is smaller or larger as we approach this point. It is not possible to decide a priori how large the condition number should be to say that the function evaluation is ill-conditioned and it depends on the circumstances in which we are working.

Definition 3.5.2 [Well-Conditioned and Ill-Conditioned].

The process of evaluating a continuously differentiable function f at a point $x = c$ is said to be ***well-conditioned*** if the condition number

$$\left| \frac{f'(c)}{f(c)} c \right|$$

at c is 'small'. The process of evaluating a function at $x = c$ is said to be ***ill-conditioned*** if it is not well-conditioned.

In the following example, we illustrate a well-conditioned computation.

Example 3.5.3.

Consider the function

$$f(x) = \sqrt{x},$$

for all $x \in [0, \infty)$. Then

$$f'(x) = \frac{1}{2\sqrt{x}}, \text{ for all } x \in [0, \infty).$$

The condition number of f is

$$\left| \frac{f'(x)}{f(x)} x \right| = \frac{1}{2}, \text{ for all } x \in [0, \infty),$$

which shows that taking square roots is a well-conditioned process.

From (3.16), we have

$$|E_r(f(x_A))| \approx \frac{1}{2} |E_r(x_A)|.$$

Thus, $E_r(f(x_A))$ is more closer to zero than $E_r(x_A)$.

We next illustrate an ill-conditioned computation.

Example 3.5.4.

Consider the function

$$f(x) = \frac{10}{1-x^2},$$

for all $x \in \mathbb{R}$. Then

$$f'(x) = \frac{20x}{(1-x^2)^2},$$

so that

$$\begin{aligned} \left|\frac{f'(x)}{f(x)}x\right| &= \left|\frac{\left(\frac{20x}{(1-x^2)^2}\right)x}{\frac{10}{(1-x^2)}}\right| \\ &= \frac{2x^2}{|1-x^2|} \end{aligned}$$

and this number can be quite large for $|x|$ near 1. Thus, for x near 1 or -1, the process of evaluating this function is ill-conditioned.

The above two examples gives us a feeling that if the process of evaluating a function is well-conditioned, then we tend to get less propagating relative error. But, this is not true in general as shown in the following example.

Example 3.5.5.

Consider the function

$$f(x) = \sqrt{x+1} - \sqrt{x}, \text{ for all } x \in (0, \infty).$$

For all $x \in (0, \infty)$, the condition number of this function is

$$\begin{aligned} \left|\frac{f'(x)}{f(x)}x\right| &= \frac{1}{2}\left|\frac{\left(\frac{1}{\sqrt{x+1}} - \frac{1}{\sqrt{x}}\right)}{\sqrt{x+1}-\sqrt{x}}x\right| \\ &= \frac{1}{2}\frac{x}{\sqrt{x+1}\sqrt{x}} \\ &\leq \frac{1}{2}, \end{aligned} \tag{3.18}$$

which shows that the process of evaluating f is well-conditioned for all $x \in (0, \infty)$. But, if we calculate $f(12345)$ using six-digit rounding, we find

$$\begin{aligned} f(12345) &= \sqrt{12346} - \sqrt{12345} \\ &= 111.113 - 111.108 \\ &= 0.005, \end{aligned}$$

while, actually, $f(12345) = 0.00450003262627751\cdots$. The calculated answer has 10% error.

The above example shows that a well-conditioned process of evaluating a function at a point is not enough to ensure the accuracy in the corresponding computed value. We need to check for the stability of the computation, which we discuss in the following subsection.

3.5.1 Stable and Unstable Computations

Suppose there are n steps to evaluate a function $f(x)$ at a point $x = c$. Then the total process of evaluating this function is said to have *instability* if at least one of the n steps is ill-conditioned. If all the steps are well-conditioned, then the process is said to be *stable*.

Example 3.5.6.

We continue the discussion in Example 3.5.5 and check the stability in evaluating the function f. Let us analyze the computational process. The function f consists of the following four computational steps in evaluating the value of f at $x = x_0$:

$$x_1 := x_0 + 1, \quad x_2 := \sqrt{x_1}, \quad x_3 := \sqrt{x_0}, \quad x_4 := x_2 - x_3.$$

Now consider the last two steps where we already computed x_2 and now going to compute x_3 and finally evaluate the function

$$f_4(t) := x_2 - t.$$

At this step, the condition number for f_4 is given by

$$\left| \frac{f_4'(t)}{f_4(t)} t \right| = \left| \frac{t}{x_2 - t} \right|.$$

Thus, f_4 is ill-conditioned when t approaches x_2. Therefore, the above process of evaluating the function $f(x)$ is *unstable*.

Let us rewrite the same function $f(x)$ as

$$\tilde{f}(x) = \frac{1}{\sqrt{x+1}+\sqrt{x}}.$$

The computational process of evaluating $\tilde{f}$ at $x = x_0$ is

$$x_1 := x_0 + 1, \quad x_2 := \sqrt{x_1}, \quad x_3 := \sqrt{x_0}, \quad x_4 := x_2 + x_3, \quad x_5 := 1/x_4.$$

It is easy to verify that the condition number of each of the above steps is well-conditioned. For instance, the last step defines

$$\tilde{f}_5(t) = \frac{1}{x_2 + t},$$

and the condition number of this function is approximately,

$$\left| \frac{\tilde{f}_5'(x)}{\tilde{f}_5(x)} x \right| = \left| \frac{t}{x_2 + t} \right| \approx \frac{1}{2}$$

for t sufficiently close to x_2. Therefore, this process of evaluating $\tilde{f}(x)$ is stable. Recall from Example 3.3.5 that the above expression gives a more accurate value for sufficiently large x.

Remark 3.5.7.

As discussed in Remark 3.3.7, we may not be lucky all the time to come out with an alternate expression that lead to stable evaluation for any given function when the original expression leads to unstable evaluation. In such situations, we have to compromise and go for a suitable approximation with other functions with stable evaluation process. For instance, we may try approximating the given function with its Taylor's polynomial, if possible.

3.6 Exercises

Floating-Point Approximation

1. Let X be a sufficiently large number which result in an overflow of memory on a computing device. Let x be a sufficiently small number which result in underflow of memory on the same computing device. Then give the output of the following operations:
   (i) $x \times X$ (ii) $3 \times X$ (iii) $3 \times x$ (iv) x/X (v) X/x.

2. In the following problems, show all the steps involved in the computation.
   i) Using 5-digit rounding, compute $37654 + 25.874 - 37679$.
   ii) Let $a = 0.00456$, $b = 0.123$, $c = -0.128$. Using 3-digit rounding, compute $(a + b) + c$, and $a + (b + c)$. What is your conclusion?
   iii) Let $a = 2$, $b = -0.6$, $c = 0.602$. Using 3-digit rounding, compute $a \times (b+c)$, and $(a \times b) + (a \times c)$. What is your conclusion?

3. To find the mid-point of an interval $[a, b]$, the formula $\frac{a+b}{2}$ is often used. Compute the mid-point of the interval $[0.982, 0.987]$ using 3-digit chopping. On the number line represent all the three points. What do you observe? Now use the more geometric formula $a + \frac{b-a}{2}$ to compute the mid-point, once again using 3-digit chopping. What do you observe this time? Why is the second formula more geometric?

4. Consider a computing device having exponents e in the range $m \leq e \leq M$, $m, M \in \mathbb{Z}$. If the device uses n-digit rounding binary floating-point arithmetic, then show that $\delta = 2^{-n}$ is the machine epsilon when $n \leq |m| + 1$.

Types of Errors

5. If $\text{fl}(x)$ is the approximation of a real number x in a computing device, and ϵ is the corresponding relative error, then show that $\text{fl}(x) = (1 - \epsilon)x$.

6. Let x, y and z be real numbers whose floating point approximations in a computing device coincide with x, y and z respectively. Show that the relative error in computing $x(y + z)$ equals $\epsilon_1 + \epsilon_2 - \epsilon_1\epsilon_2$, where $\epsilon_1 = E_r(\text{fl}(y + z))$ and $\epsilon_2 = E_r(\text{fl}(x \times \text{fl}(y + z)))$.

7. Let $\epsilon = E_r(\text{fl}(x))$. Show that
   i) $|\epsilon| \leq 10^{-n+1}$ if the computing device uses n-digit (decimal) chopping.
   ii) $|\epsilon| \leq \frac{1}{2}10^{-n+1}$ if the computing device uses n-digit (decimal) rounding.
   iii) Can the equality hold in the above inequalities?

8. Let the approximation $\sin x \approx x$ be used on the interval $[-\delta, \delta]$ where $\delta > 0$. Show that for all $x \in [-\delta, \delta]$,

$$|\sin x - x| < \frac{x^3}{6}.$$

Find a $\delta > 0$ such that the inequality

$$|\sin x - x| < \frac{1}{2}10^{-6},$$

holds for all $x \in [-\delta, \delta]$.

9. Let $x_A = 3.14$ and $y_A = 2.651$ be obtained from x_T and y_T using 4-digit rounding. Find the smallest interval that contains
(i) x_T (ii) y_T (iii) $x_T + y_T$ (iv) $x_T - y_T$ (v) $x_T \times y_T$ (vi) x_T/y_T.
10. The ideal gas law is given by $PV = nRT$ where R is the gas constant. We are interested in knowing the value of T for which $P = V = n = 1$. If R is known only approximately as $R_A = 8.3143$ with an absolute error at most 0.12×10^{-2}. What is the relative error in the computation of T that results in using R_A instead of R?

Loss of Significance and Propagation of Error

11. Obtain the number of significant digits in x_A when compared to x for the following cases:
(i) $x = 451.01$ and $x_A = 451.023$ (ii) $x = -0.04518$ and $x_A = -0.045113$
(iii) $x = 23.4604$ and $x_A = 23.4213$.
12. Instead of using the true values $x_T = 0.71456371$ and $y_T = 0.71456238$ in calculating $z_T = x_T - y_T (= 0.133 \times 10^{-5})$, if we use the approximate values $x_A = 0.71456414$ and $y_A = 0.71456103$, and calculate $z_A = x_A - y_A (= 0.311 \times 10^{-5})$, then find the loss of significant digits in the process of calculating z_A when compared to the significant digits in x_A.
13. Let $x < 0 < y$ be such that the approximate numbers x_A and y_A has seven and nine significant digits with x and y respectively. Show that $z_A := x_A - y_A$ has at least six significant digits when compared to $z := x - y$.

Propagation of Relative Error

14. Let $x = 0.65385$ and $y = 0.93263$. Obtain the total error in obtaining the product of these two numbers using 3-digit rounding.

15. Find the condition number at a point $x = c$ for the following functions
(i) $f(x) = x^2$, (ii) $g(x) = \pi^x$, (iii) $h(x) = b^x$.

16. Let x_T be a real number. Let $x_A = 2.5$ be an approximate value of x_T with an absolute error at most 0.01. The function $f(x) = x^3$ is evaluated at $x = x_A$ instead of $x = x_T$. Estimate the resulting absolute error.

17. Is the process of computing the function $f(x) = (e^x - 1)/x$ stable or unstable for $x \approx 0$? Justify your answer.

18. Show that the process of evaluating the function

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

for $x \approx 0$ is unstable. Suggest an alternate formula for evaluating f for $x \approx 0$, and check if the computation process is stable using this new formula.

19. Check for stability of computing the function

$$f(x) = \left(x + \frac{1}{3}\right) - \left(x - \frac{1}{3}\right)$$

for large values of x.

20. Check for stability of computing the function

$$g(x) = \frac{\left(3 + \frac{x^2}{3}\right) - \left(3 - \frac{x^2}{3}\right)}{x^2}$$

for values of x very close to 0.

21. Check for stability of computing the function

$$h(x) = \frac{\sin^2 x}{1 - \cos^2 x}$$

for values of x very close to 0.

22. Compute the values of the function

$$\frac{\sin x}{\sqrt{1 - \sin^2 x}}$$

using your calculator (without simplifying the given formula) at $x = 89.9, 89.95, 89.99$ degrees. Also compute the values of the function $\tan x$ at these values of x and compare the number of significant digits.