

# Approximation Errors in Computer Arithmetic (Chapters 3 and 4)

## Outline:

- Positional notation – binary representation of numbers
  - Computer representation of integers
  - Floating point representation
    - IEEE standard for floating point representation
- Truncation errors in floating point representation
  - Chopping and rounding
  - Absolute error and relative error
  - Machine precision
  - Significant digits
- Approximating a function — Taylor series



Positional notation: different position represents different magnitude.

**Base-2:** sequence of 0 and 1

Primary logic units of digital computers are ON/OFF components.

Bit: each binary digit is referred to as a bit.

For example:  $(1011)_2 = 1 \times 2^3 + 1 \times 2^1 + 1 = (11)_{10}$

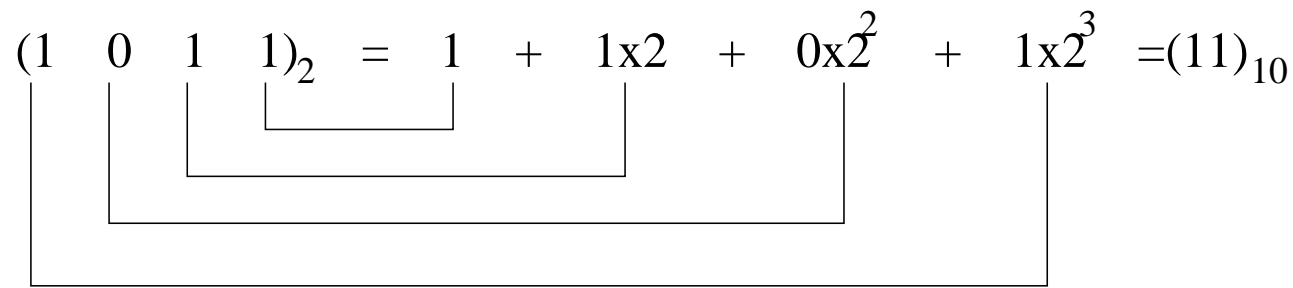
$$(1 \ 0 \ 1 \ 1)_2 = 1 + 1 \times 2 + 0 \times 2^2 + 1 \times 2^3 = (11)_{10}$$


Figure 2: Positional notation of a base-2 number

## 1.2 Binary representation of integers

### Signed magnitude method

The sign bit is used to represent positive as well as negative numbers:

sign bit = 0  $\rightarrow$  positive number

sign bit = 1  $\rightarrow$  negative number

Examples: 8-bit representation

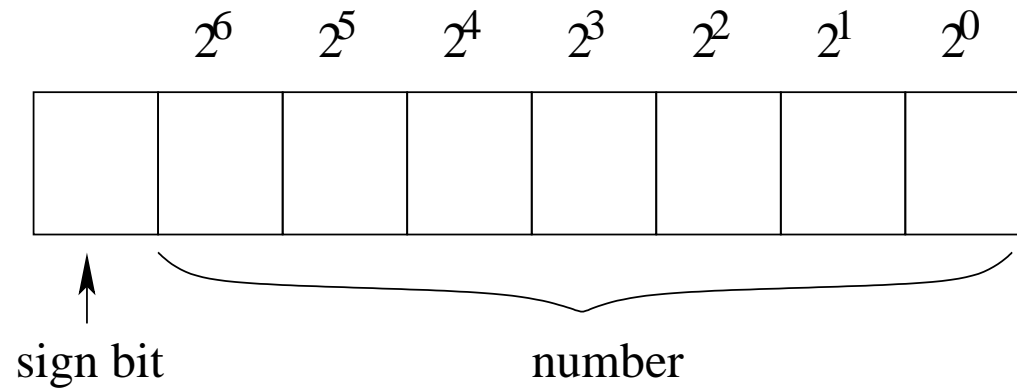


Figure 3: 8-bit representation of an integer with a sign bit

$$(00011000)_2 = (2^4 + 2^3) = (24)_{10}$$

$$(10011000)_2 = -(2^4 + 2^3) = (-24)_{10}$$

	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
24:	0	0	0	1	1	0	0

	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
-24:	1	0	0	1	1	0	0

Figure 4: Signed magnitude representation of  $(24)_{10}$  and  $-24_{10}$

**Maximum number in 8-bit representation:**  $(01111111)_2 = \sum_{i=0}^6 1 \times 2^i = 127$ .

Minimum number in 8-bit representation:  $(11111111)_2 = -\sum_{i=0}^6 1 \times 2^i = -127$ .

The range of representable numbers in 8-bit signed representation is from -127 to 127.

In general, with  $n$  bits (including one sign bit), the range of representable numbers is  $-(2^{n-1} - 1)$  to  $2^{n-1} - 1$ .

## 2's complement representation

A computer stores 2's complement of a number.

How to find 2's complement of a number:

i) The 2's complement of a positive integer is the same:

$$(24)_{10} = (00011000)_2$$

ii) The 2's complement of a negative integer: Negative 2's complement numbers are represented as the binary number that when added to a positive number of the same magnitude equals zero.

– toggle the bits of the positive integer:  $(00011000)_2 \rightarrow (11100111)_2$

– add 1  $(11100111 + 1)_2 = (11101000)_2$

$$(24)_{10} = (0\ 0\ 0\ 1\ 1\ 0\ 0\ 0)_2$$

Toggle bits:      1   1   1   0   0   1   1   1

$$\begin{array}{r} \text{Add 1: } \quad +) \qquad \qquad \qquad 1 \\ \hline \qquad \qquad \qquad 1\ 1\ 1\ 0\ 1\ 0\ 0\ 0 \end{array}$$

Figure 5: 2's complement representation of -24

$$(128)_{10} = (1\ 0\ 0\ 0\ 0\ 0\ 0\ 0)_2$$

Toggle bits:      0   1   1   1   1   1   1   1

$$\begin{array}{r} \text{Add 1: } \quad +) \qquad \qquad \qquad 1 \\ \hline \qquad \qquad \qquad 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \end{array}$$

Figure 6: 2's complement representation of -128

With 8-bits, representable range: from -128 to 127.

In 2's complement representation,  $(10000000)_2 = -128$ . The representation 10000000 is not used in signed notation.

With 8 bits, the signed magnitude method can represent all numbers from -127 to 127, while the 2's complement method can represent all numbers from -128 to 127.

2's complement is preferred because of the way arithmetic operations are performed in the computer<sup>1</sup>. In addition, the range of representable numbers is -128 to 127.

### 1.3 Binary representation of floating point numbers

Consider a decimal floating point number:

$$(37.71)_{10} = 3 \times 10^1 + 7 \times 10^0 + 7 \times 10^{-1} + 1 \times 10^{-2}$$

$n$ -th digit right to “.” represents  $0 \sim 9 \times 10^{-n}$

Similarly, a binary floating point number:

$$(10.11)_2 = 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$$

$n$ -th digit right to “.” represents  $0 \sim 1 \times 2^{-n}$

---

<sup>1</sup>Interested student can visit [http://en.wikipedia.org/wiki/Two's\\_complement](http://en.wikipedia.org/wiki/Two's_complement) for further explanation.

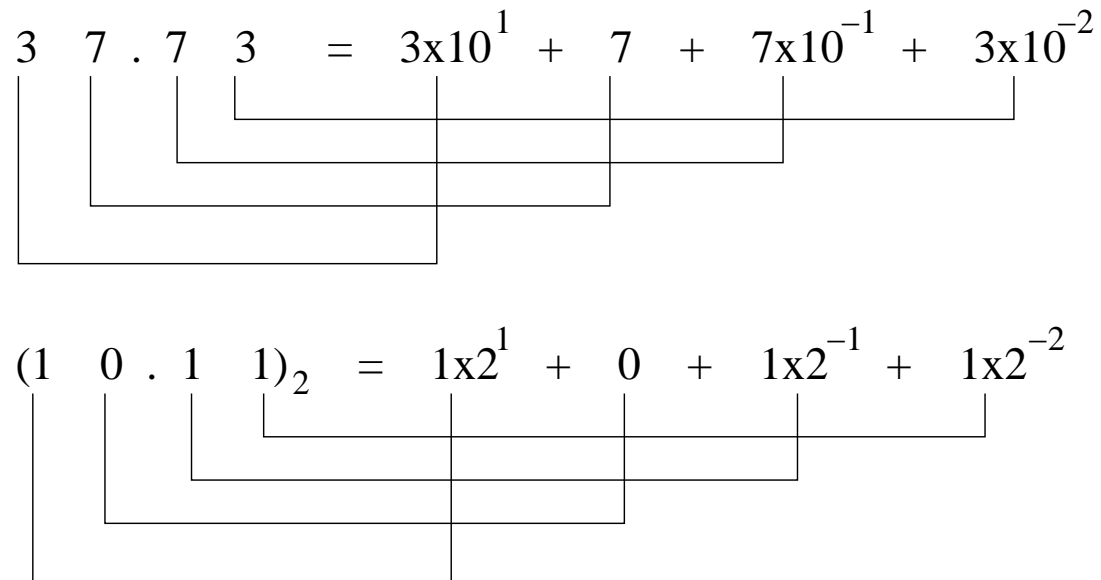


Figure 7: Positional notations of floating point numbers

## Normalized representation:

Decimal:

$$37.71 = 3.771 \times 10^1$$

$$0.3771 = 3.771 \times 10^{-1}$$

Idea: move the decimal point to the left or right until there is only one non-zero digit to the left of the point (.) and then compensate for it in the exponent.

Binary:

$$(10.11)_2 = (1.011)_2 \times 2^1$$

$(\times 2^1) \leftrightarrow$  move decimal point one position right

$(\times 2^{-1}) \leftrightarrow$  move decimal point one position left



In general, a real number  $x$  can be written as

$$x = (-1)^s \cdot m \cdot b^e$$

where

$s$  is the sign bit ( $s = 0$  represents positive numbers, and  $s = 1$  negative numbers),  
 $m$  is the mantissa (the normalized value) ( $m = 1.f$  for  $x \neq 0$  binary),  
 $b$  is the base ( $b = 2$  for binary),  
and  $e$  is the exponent.

In computers, we store  $s$ ,  $m$  and  $e$ . An example of 8-bit representation of floating point numbers is shown in Fig. 8.

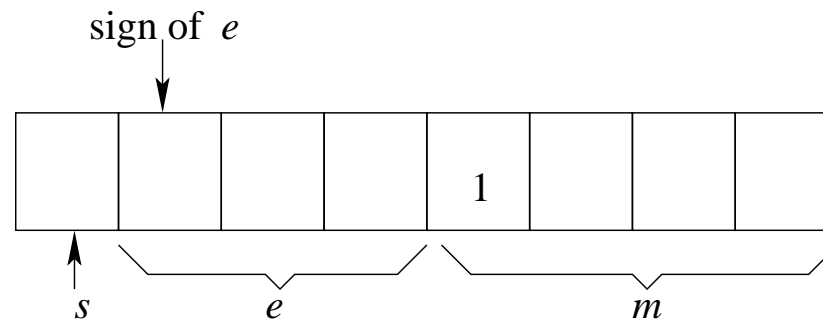


Figure 8: 8-bit floating point normalized representation

Example:  $(2.75)_{10} = (10.11)_2 = (1.011)_2 \times 2^1$

The MSB bit of  $m$  is always 1, except when  $x = 0$ . (Why?) Therefore,  $m$  can be

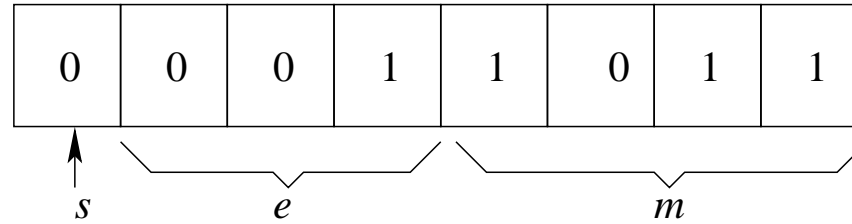


Figure 9: 8-bit floating point normalized representation of 2.75

rewritten as

$$m = (1.f)_2$$

Since only the last three bits carry information, we may exclude the MSB of  $m$  and use the last four bits to represent  $f$ . Then

$$x = (-1)^s \cdot m \cdot b^e = (-1)^s \cdot (1.f)_2 \cdot b^e$$

For example,  $(2.75)_{10} = (1.011)_2 \times 2^1$  can be represented in floating point format as Fig. 10.

In the improved floating point representation, we store  $s$ ,  $f$  and  $e$  in computers.

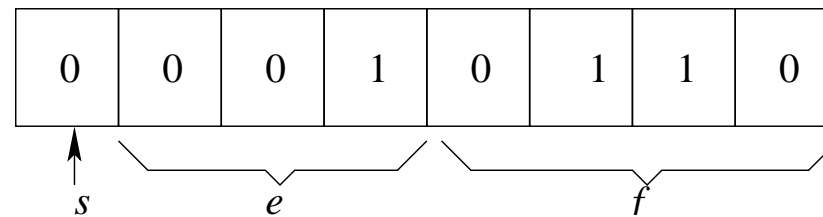


Figure 10: Improved 8-bit floating point normalized representation of 2.75

Special case,  $x = 0$ .

## IEEE standard for floating point representation<sup>2</sup>

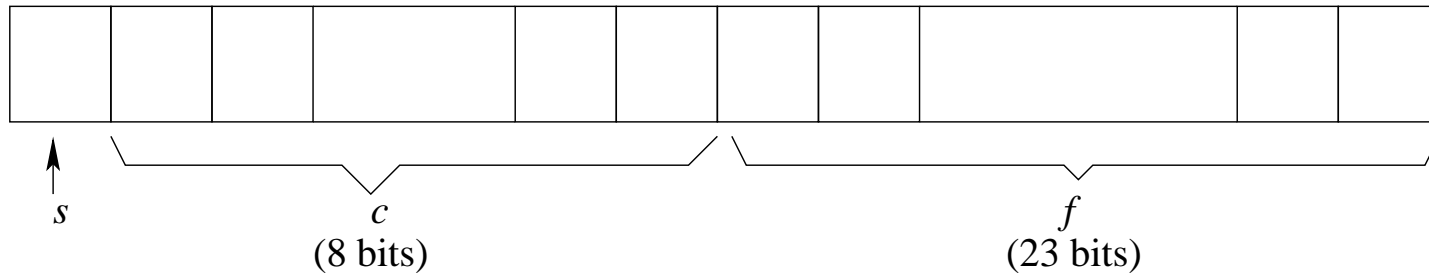


Figure 11: IEEE 32-bit floating point format

In IEEE 32-bit floating point format,

- $s$  (1 bit): sign bit;
- $c$  (8 bits): exponent  $e$  with offset,  $e = c - 127$ , and  $c = e + 127$ ;  
The exponent is not stored directly. The reason for the offset is to make the aligning of the radix point easier during arithmetic operation.  
The range of  $c$  is from 0 to 255.  
Special case:  $c = 0$  when  $x = 0$ , and  $c = 255$  when  $x$  is infinity or not a number (Inf/Nan).  
The valid range of  $e$  is from -126 to 127 ( $x \neq 0$ , Inf, and Nan).
- $f$  (23 bits):  $m = (1.f)_2$ , ( $x \neq 0$ , Inf, and Nan),  $0 \leq f < 1$ .

Example:  $2.75 = (10.11)_2 = (1.011)_2 \times 2^1$ , where  $s = 0$ ,  $m = (1.011)_2$ ,  $e = 1$ , and  $c = e + 127 = 128 = (10000000)_2$ .

<sup>2</sup>More details about the IEEE Standard can be found from <http://steve.hollasch.net/cgindex/coding/ieeefloat.html> and <http://grouper.ieee.org/groups/754/>.

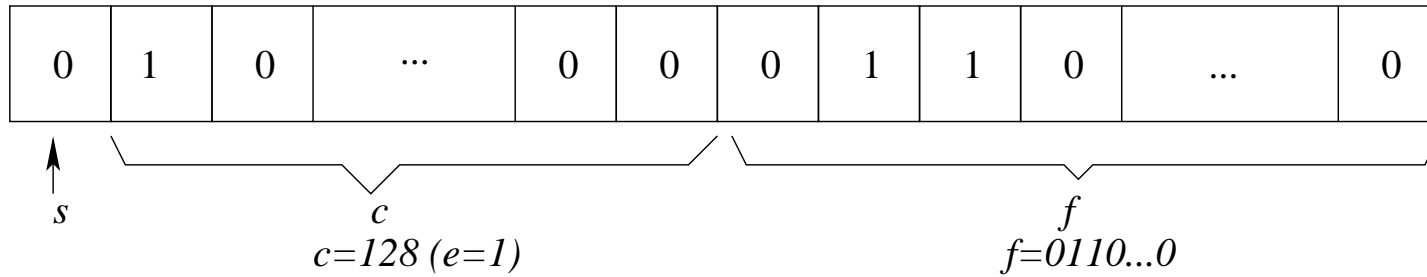


Figure 12: IEEE 32-bit floating point representation of 2.75

## 1.4 Floating point errors

Using a finite number of bits, for example, as in the IEEE format, only a finite number of real values can be represented exactly. That is, only certain numbers between a minimum and a maximum can be represented.

With the IEEE 32-bit format

- The upper bound  $U$  on the representable value of  $x$ :

When  $s = 0$ , and both  $c$  and  $f$  are at their maximum values, i.e.,  $c = 254$  (or  $e = 127$ ), and  $f = (11 \dots 1)_2$  (or  $m = (1.11 \dots 1)_2$ ),

$$U = m \cdot b^e = (2 - 2^{-23}) \times 2^{127} \approx 3.4028 \times 10^{38}$$

- The lower bound  $L$  on the positive representable value of  $x$ :

When  $s = 0$ , and both  $c$  ( $c \neq 0$ ) and  $f$  are at their minimum values, i.e.,  $c = 1$  ( $e = c - 127 = -126$ ), and  $f = (00 \dots 0)_2$  (when  $m = (1.00 \dots 0)_2$ ). Then

$$L = m \cdot b^e = 1 \times 2^{-126} = 1.1755 \times 10^{-38}$$

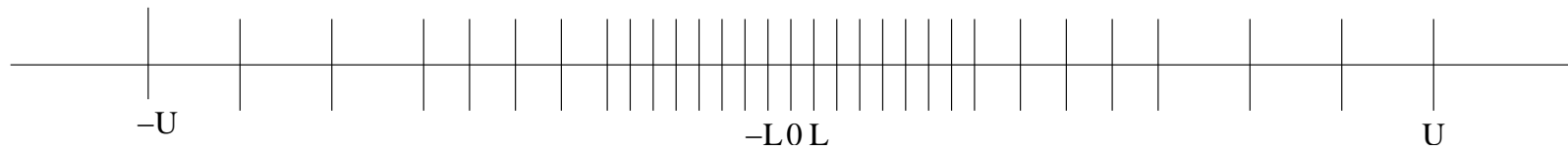


Figure 13: Range of exactly representable numbers

- Only the numbers between  $-U$  and  $-L$ , 0 and between  $L$  and  $U$  can be represented:  $-U \leq x \leq -L$ ,  $x = 0$ , or  $L \leq x \leq U$ .

During a calculation,

- when  $|x| > U$ , the result is in an overflow;
- when  $|x| < L$ , the result is in an underflow;
- if  $x$  falls between two exactly representable numbers, its floating point representation has to be approximated, leading to truncation errors.
- When  $|x|$  increases, truncation error increases in general.

## 2 Truncation errors in floating point representation

When a real value  $x$  is stored using its floating point representation  $fl(x)$ , truncation error occurs. This is because of the need to represent an infinite number of real values using a finite number of bits.

Example: the floating point representation of  $(0.1)_{10}$

$$(0.1)_{10} = (0.0001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ \dots)_2$$

### 2.1 Truncation errors

Assume: we have  $t$  number of bits to represent  $m$  (or equivalently  $t - 1$  bits for  $f$ ). In IEEE 32-bit format,  $t - 1 = 23$ , or  $t = 24$ .

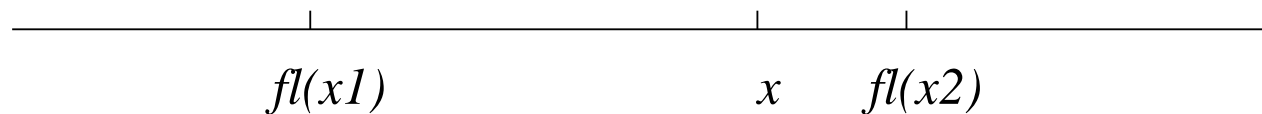


Figure 14: Truncation error

With a finite number of bits, there are two ways to approximate a number that cannot be represented exactly in floating point format.

Consider real value  $x$  which cannot be represented exactly and falls between two floating point representations  $fl(x_1)$  and  $fl(x_2)$ .

## Chopping:

$$fl(x) = fl(x_1)$$

— Ignore all bits beyond the  $(t - 1)$ th one in  $f$  (or  $t$ th one in  $m$ ).

## Rounding:

$$fl(x) = \begin{cases} fl(x_1), & \text{if } x - fl(x_1) \leq fl(x_2) - x \\ fl(x_2), & \text{otherwise.} \end{cases}$$

— Select the closest representable floating point number.

Example:  $x > 0$ ,

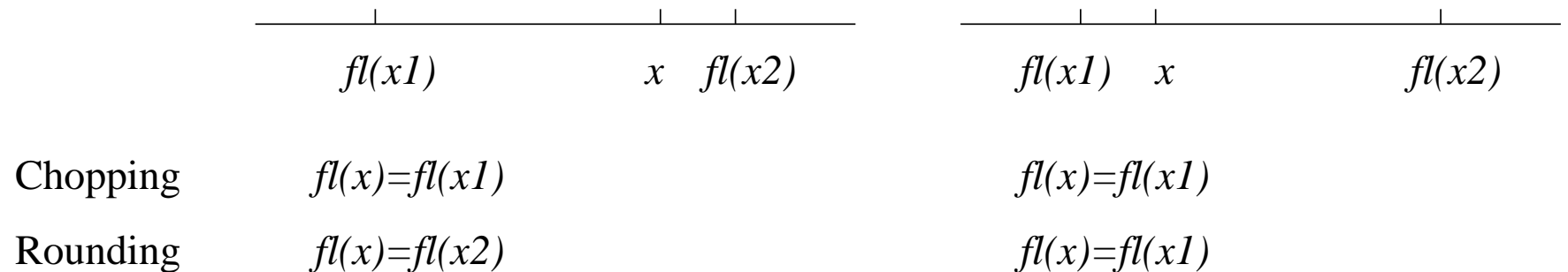


Figure 15: Example of using chopping and rounding

As  $x$  increases,  $fl(x_2) - fl(x_1)$  increases, then the truncation error increases.

## Absolute error:

$$\text{Absolute error} \triangleq |x - fl(x)|$$

is the difference between the actual value and its floating point representation.

**Relative error:**

$$\text{Relative error} \triangleq \frac{|x - fl(x)|}{|x|}$$

is the error with respect to the value of the number to be represented.

## 2.2 Bound on the errors ( $b = 2$ )

Consider chopping,  $x > 0$ .

The real value  $x$  can be represented exactly with an infinite number of bits as

$$x = 1. \underbrace{XX \dots XX}_{t-1 \text{ bits}} \underbrace{XX \dots XX}_{\infty \text{ bits}} \times b^e$$

The floating point representation  $fl(x)$  is

$$fl(x) = 1. \underbrace{XX \dots XX}_{t-1 \text{ bits}} \times b^e$$

The absolute error is maximum when

$$x = 1. \underbrace{XX \dots XX}_{t-1 \text{ bits}} \underbrace{11 \dots 11}_{\infty \text{ bits}} \times b^e$$



Then, the absolute error is bounded by

$$|x - fl(x)| < \sum_{i=t}^{\infty} b^{-i} \times b^e < b^{-t+1} \times b^e$$

and the relative error is bounded by

$$\frac{|x - fl(x)|}{|x|} < \frac{b^{1-t} \times b^e}{|x|}$$

Because  $|x| > 1.00 \dots 00 \times b^e = b^e$ ,

$$\frac{|x - fl(x)|}{|x|} < \frac{b^{1-t} \times b^e}{b^e} = b^{1-t}$$

For binary representation,  $b = 2$ ,

$$\frac{|x - fl(x)|}{|x|} < 2^{1-t}$$

which is the bound on relative errors for chopping.

For chopping,

$$|x - fl(x)| \leq [fl(x_2) - fl(x_1)]$$

For rounding: the maximum truncation error occurs when  $x$  is in the middle of the interval between  $fl(x_1)$  and  $fl(x_2)$ . Then

$$|x - fl(x)| \leq \frac{1}{2}[fl(x_2) - fl(x_1)]$$

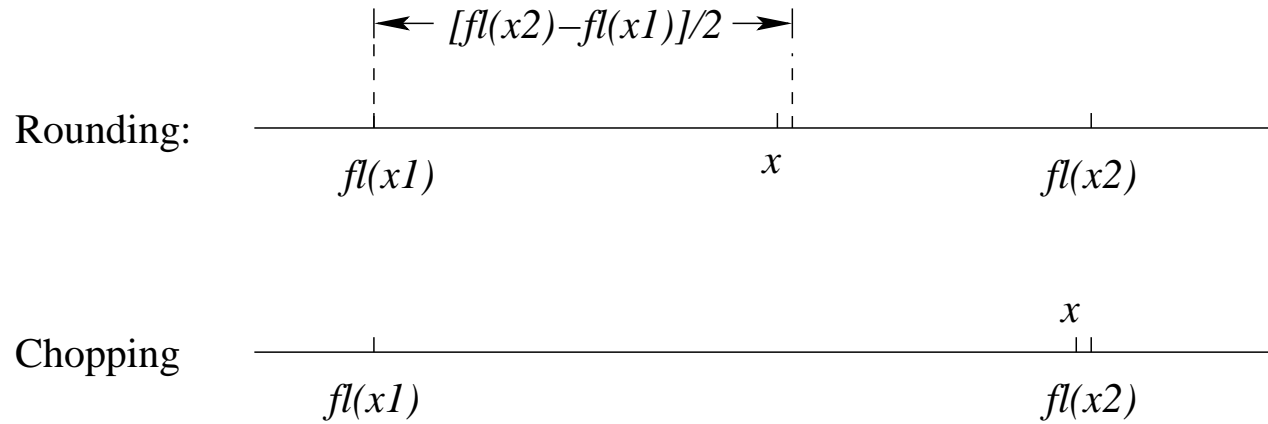


Figure 16: Illustration of error bounds

The bound on absolute errors for rounding is

$$|x - fl(x)| \leq \frac{2^{-t+1} \cdot 2^e}{2} = 2^{-t} \cdot 2^e$$

The bound on relative errors for rounding is

$$\frac{|x - fl(x)|}{|x|} \leq \frac{2^{-t+1}}{2} = 2^{-t}$$

**Machine precision:** Define machine precision,  $\epsilon_{mach}$ , as the maximum relative error. Then

$$\epsilon_{mach} = \begin{cases} 2^{1-t}, & \text{for chopping} \\ \frac{2^{1-t}}{2} = 2^{-t}, & \text{for rounding} \end{cases}$$

For IEEE standard,  $t - 1 = 23$  or  $t = 24$ ,  $\epsilon_{mach} = 2^{-24} \approx 10^{-7}$  for rounding.

The machine precision  $\epsilon_{mach}$  is also defined as the smallest number  $\epsilon$  such that  $fl(1 + \epsilon) > 1$ , i.e., if  $\epsilon < \epsilon_{mach}$ , then  $fl(1 + \epsilon) = fl(1)$ .

(Prove that the two definitions are equivalent.)

Note: Difference between the machine precision,  $\epsilon_{mach}$ , and the lower bound on the representable floating point numbers,  $L$ :

- $\epsilon_{mach}$  is determined by the number of bits in  $m$
- $L$  is determined by the number of bits in the exponent  $e$ .

### 2.3 Effects of truncation and machine precision

Example 1: Evaluate  $y = (1 + \delta) - (1 - \delta)$ , where  $L < \delta < \epsilon_{mach}$ . With rounding,

$$\begin{aligned} y &= fl(1 + \delta) - fl(1 - \delta) \\ &= 1 - 1 = 0. \end{aligned}$$

The correct answer should be  $2\delta$ .

When  $\delta = 1.0 \times 2^{-25}$ ,

$$\begin{aligned} 1 + \delta &= 1.0 \times 2^0 + 1.0 \times 2^{-25} \\ &= (1.\underbrace{00 \cdots 0}_{24 \text{ 0's}}1)_2 \times 2^0 \\ fl(1 + \delta) &= 1 \end{aligned}$$

Example 2: Consider the infinite summation  $y = \sum_{n=1}^{\infty} \frac{1}{n}$ . In theory, the sum diverges as  $n \rightarrow \infty$ . But, using floating point operations (with finite number of bits), we get a finite output.

As  $n$  increases, the addition due to another  $\frac{1}{n}$  does not change the output! That is,

$$\begin{aligned} y &= \sum_{n=1}^{\infty} \frac{1}{n} = \sum_{n=1}^{k-1} \frac{1}{n} + \sum_{n=k}^{\infty} \frac{1}{n} = \sum_{n=1}^{k-1} \frac{1}{n} \\ \frac{1}{k-1} &\geq \epsilon_{mach} \quad \text{and} \quad \frac{1}{k} < \epsilon_{mach} \end{aligned}$$

Example 3: Consider the evaluation of  $e^{-x}$ ,  $x > 0$ , using

$$e^{-x} = \sum_{n=0}^{\infty} \frac{(-x)^n}{n!} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \dots$$

This may result in erroneous output due to cancellation. Alternative expressions are needed for evaluation with negative exponents.

## 2.4 Significant figures (digits)



Figure 17: Significant figures

The significant digits of a number are those that can be used with confidence. For example:

$$0.00453 = 4.53 \times 10^{-3} \text{ 3 significant digits}$$

$$0.004530 = 4.530 \times 10^{-3} \text{ 4 significant digits}$$

Example:  $\pi = 3.1415926 \dots$

With 2 significant digits,  $\pi \approx 3.1$

With 3 significant digits,  $\pi \approx 3.14$

With 4 significant digits,  $\pi \approx 3.142$

The number of significant digits is the number of certain digits plus one estimated digit.

### 3 Approximating a function using a polynomial

#### 3.1 McLaurin series

Assume that  $f(x)$  is a continuous function of  $x$ , then

$$f(x) = \sum_{i=0}^{\infty} a_i x^i = \sum_{i=0}^{\infty} \frac{f^{(i)}(0)}{i!} x^i$$

is known as the McLaurin Series, where  $a_i$ 's are the coefficients in the polynomial expansion given by

$$a_i = \frac{f^{(i)}(x)}{i!} \Big|_{x=0}$$

and  $f^{(i)}(x)$  is the  $i$ -th derivative of  $f(x)$ .

The McLaurin series is used to predict the value  $f(x_1)$  (for any  $x = x_1$ ) using the function's value  $f(0)$  at the “reference point” ( $x = 0$ ). The series is approximated by summing up to a suitably high value of  $i$ , which can lead to approximation or truncation errors.

**Problem:** When function  $f(x)$  varies significantly over the interval from  $x = 0$  to  $x = x_1$ , the approximation may not work well.

A better solution is to move the reference point closer to  $x_1$ , at which the function's polynomial expansion is needed. Then,  $f(x_1)$  can be represented in terms of  $f(x_r)$ ,  $f^{(1)}(x_r)$ , etc.

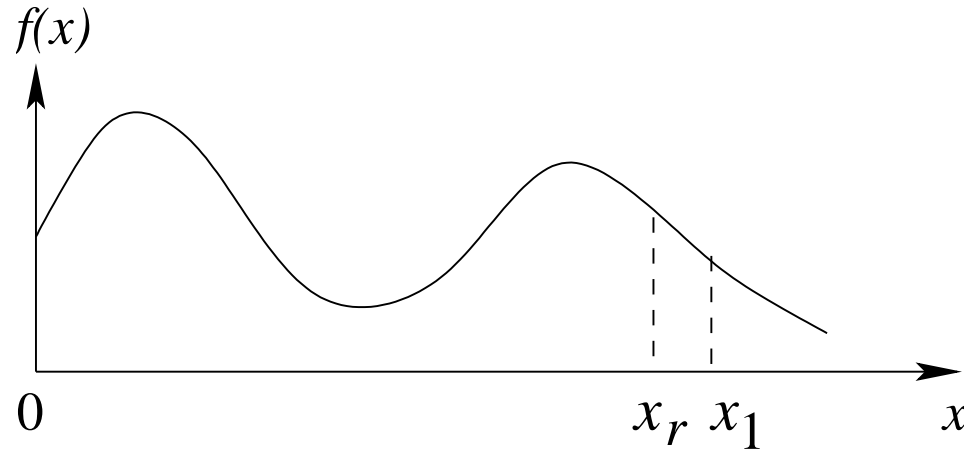


Figure 18: Taylor series

### 3.2 Taylor series

Assume that  $f(x)$  is a continuous function of  $x$ , then

$$f(x) = \sum_{i=0}^{\infty} a_i (x - x_r)^i$$



where  $a_i = \frac{f^{(i)}(x)}{i!} \Big|_{x=x_r}$ . Define  $h = x - x_r$ . Then,

$$f(x) = \sum_{i=0}^{\infty} a_i h^i = \sum_{i=0}^{\infty} \frac{f^{(i)}(x_r)}{i!} h^i$$

which is known as the Taylor Series.

If  $x_r$  is sufficiently close to  $x$ , we can approximate  $f(x)$  with a small number of coefficients since  $(x - x_r)^i \rightarrow 0$  as  $i$  increases.

Question: What is the error when approximating function  $f(x)$  at  $x$  by  $f(x) = \sum_{i=0}^n a_i h^i$ , where  $n$  is a finite number (the order of the Taylor series)?

### **Taylor theorem:**

A function  $f(x)$  can be represented exactly as

$$f(x) = \sum_{i=0}^n \frac{f^{(i)}(x_r)}{i!} h^i + R_n$$

where  $R_n$  is the remainder (error) term, and can be calculated as

$$R_n = \frac{f^{(n+1)}(\alpha)}{(n+1)!} h^{n+1}$$

and  $\alpha$  is an unknown value between  $x_r$  and  $x$ .

- Although  $\alpha$  is unknown, Taylor's theorem tells us that the error  $R_n$  is proportional to  $h^{n+1}$ , which is denoted by

$$R_n = O(h^{n+1})$$

which reads “ $R_n$  is order  $h$  to the power of  $n + 1$ ”.

- With  $n$ -th order Taylor series approximation, the error is proportional to step size  $h$  to the power  $n + 1$ . Or equivalently, the truncation error goes to zero no slower than  $h^{n+1}$  does.
- With  $h \ll 1$ , an algorithm or numerical method with  $O(h^2)$  is better than one with  $O(h)$ . If you half the step size  $h$ , the error is quartered in the former but is only halved in the latter.

Question: How to find  $R_n$ ?

$$\begin{aligned} R_n &= \sum_{i=0}^{\infty} a_i h^i - \sum_{i=0}^n a_i h^i \\ &= \sum_{i=n+1}^{\infty} a_i h^i = \sum_{i=n+1}^{\infty} \frac{f^{(i)}(x_r)}{i!} h^i \end{aligned}$$

For small  $h$  ( $h \ll 1$ ),

$$R_n \approx \frac{f^{(n+1)}(x_r)}{(n+1)!} h^{n+1}$$

The above expression can be used to evaluate the dominant error terms in the  $n$ -th order approximation.

For different values of  $n$  (the order of the Taylor series), a different trajectory is fitted (or approximated) to the function. For example:

- $n = 0$  (zero order approximation)  $\rightarrow$  straight line with zero slope
- $n = 1$  (first order approximation)  $\rightarrow$  straight line with some slope
- $n = 2$  (second order approximation)  $\rightarrow$  quadratic function

**Example 1:** Expand  $f(x) = e^x$  as a McLaurin series.

Solution:

$$\begin{aligned} a_0 &= f(0) = e^0 = 1, \\ a_1 &= \frac{f'(x)}{1!} \Big|_{x=0} = \frac{e^0}{1} = 1 \\ a_i &= \frac{f^{(i)}(x)}{i!} \Big|_{x=0} = \frac{e^0}{i!} = \frac{1}{i!} \end{aligned}$$

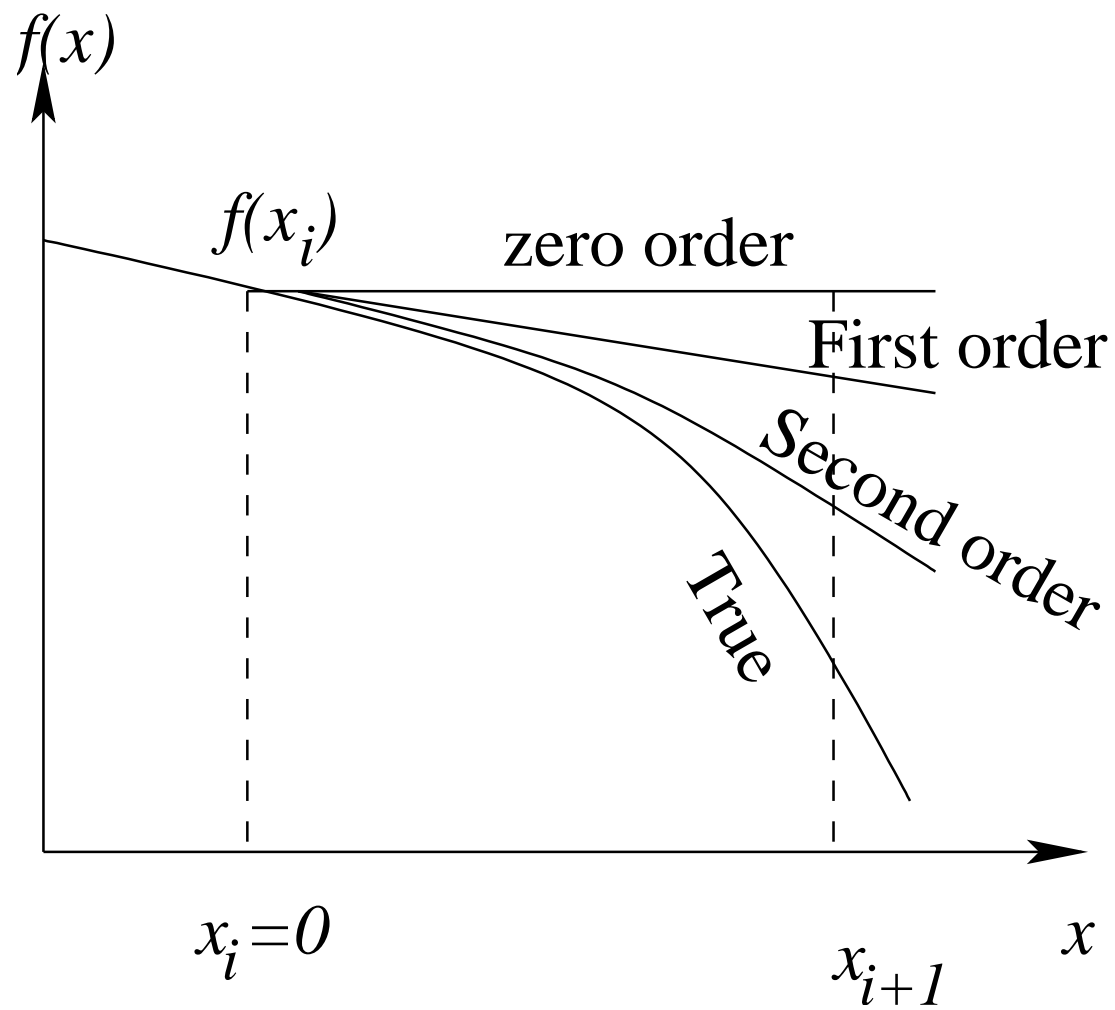


Figure 19: Taylor series

Then  $f(x) = e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!}$ .

**Example 2:** Find the McLaurin series up to order 4, Taylor series (around  $x = 1$ ) up to order 4 of function  $f(x) = x^3 - 2x^2 + 0.25x + 0.75$ .

Solution:

$$\begin{array}{lll} f(x) = x^3 - 2x^2 + 0.25x + 0.75 & f(0) = 0.75 & f(1) = 0 \\ f'(x) = 3x^2 - 4x + 0.25 & f'(0) = 0.25 & f'(1) = -0.75 \\ f''(x) = 6x - 4 & f''(0) = -4 & f''(1) = 2 \\ f^{(3)}(x) = 6 & f^{(3)}(0) = 6 & f^{(3)}(1) = 6 \\ f^{(4)}(x) = 0 & f^{(4)}(0) = 0 & f^{(4)}(1) = 0 \end{array}$$

The McLaurin series of  $f(x) = x^3 - 2x^2 + 0.25x + 0.75$  can be written as

$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(0)}{i!} x^i = \sum_{i=0}^3 \frac{f^{(i)}(0)}{i!} x^i$$

Then the third order McLaurin series expansion is

$$\begin{aligned} f_{M3}(x) &= f(0) + f'(0)x + \frac{1}{2}f''(0)x^2 + \frac{1}{3!}f^{(3)}(0)x^3 \\ &= 0.75 + 0.25x - 2x^2 + x^3 \end{aligned}$$

which is the same as the original polynomial function.

The lower order McLaurin series expansion may be written as

$$\begin{aligned}f_{M2}(x) &= f(0) + \frac{1}{2}f''(0)x^2 \\&= 0.75 + 0.25x - 2x^2 \\f_{M1}(x) &= 0.75 + 0.25x \\f_{M0}(x) &= 0.75\end{aligned}$$

The Taylor series can be written as

$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(x_r)}{i!} (x - x_r)^i = \sum_{i=0}^3 \frac{f^{(i)}(1)}{i!} (x - 1)^i$$

Then the third order Taylor series of  $f(x)$  is

$$\begin{aligned}f_{T3}(x) &= f(1) + f'(1)(x - 1) + \frac{1}{2}f''(1)(x - 1)^2 + \frac{1}{3!}f^{(3)}(1)(x - 1)^3 \\&= 0.75 + 0.25x - 2x^2 + x^3\end{aligned}$$

which is the same as the original function.

The lower order Taylor series expansion may be written as

$$\begin{aligned}f_{T2}(x) &= f(1) + \frac{1}{2}f''(1)(x-1)^2 \\&= -0.75(x-1) + (x-1)^2 \\&= 1.75 - 2.75x + x^2 \\f_{T1}(x) &= f(1) + f'(x-1) \\&= -0.75(x-1) = 0.75 - 0.75x \\f_{T0}(x) &= f(1) = 0\end{aligned}$$

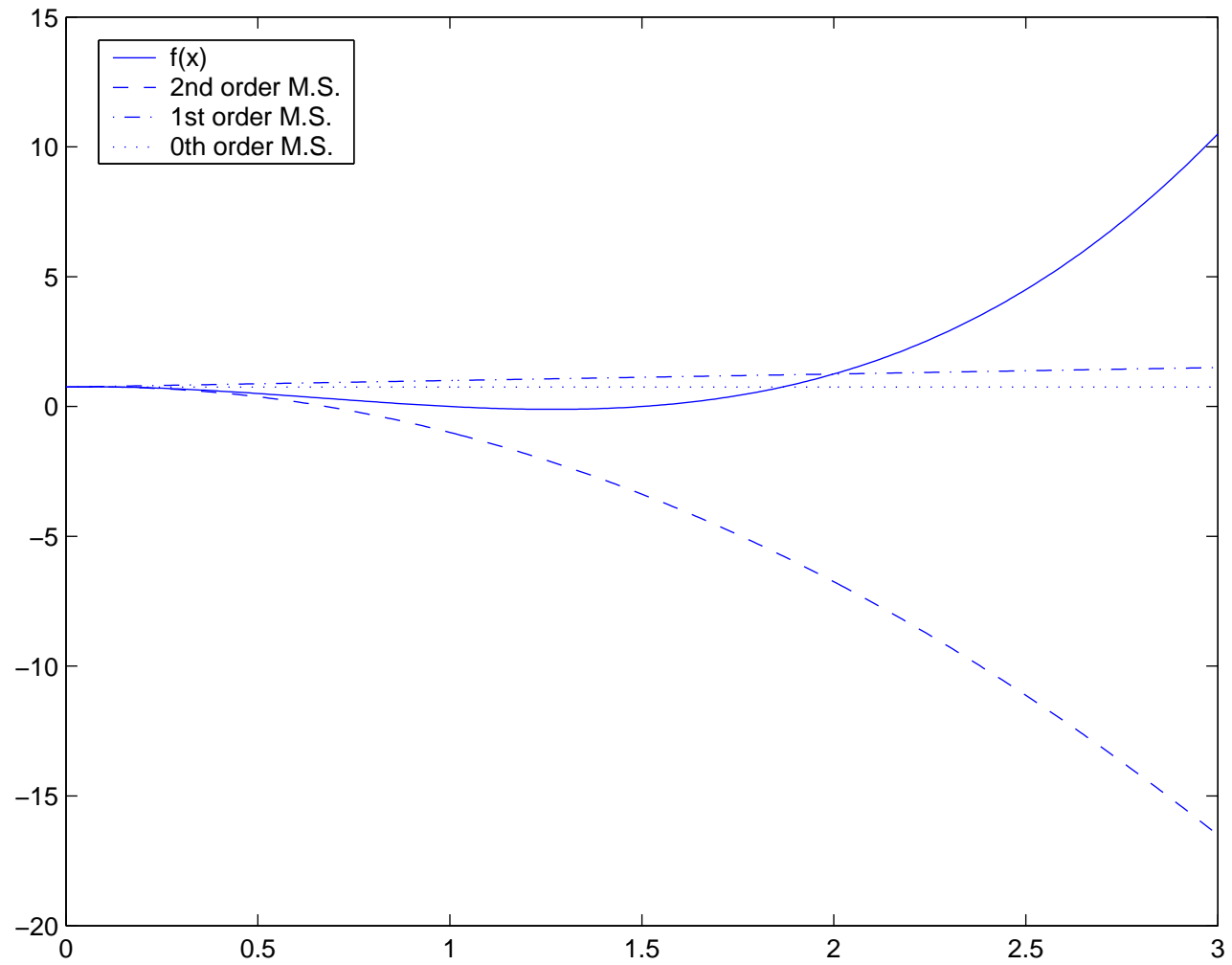


Figure 20: Example 1



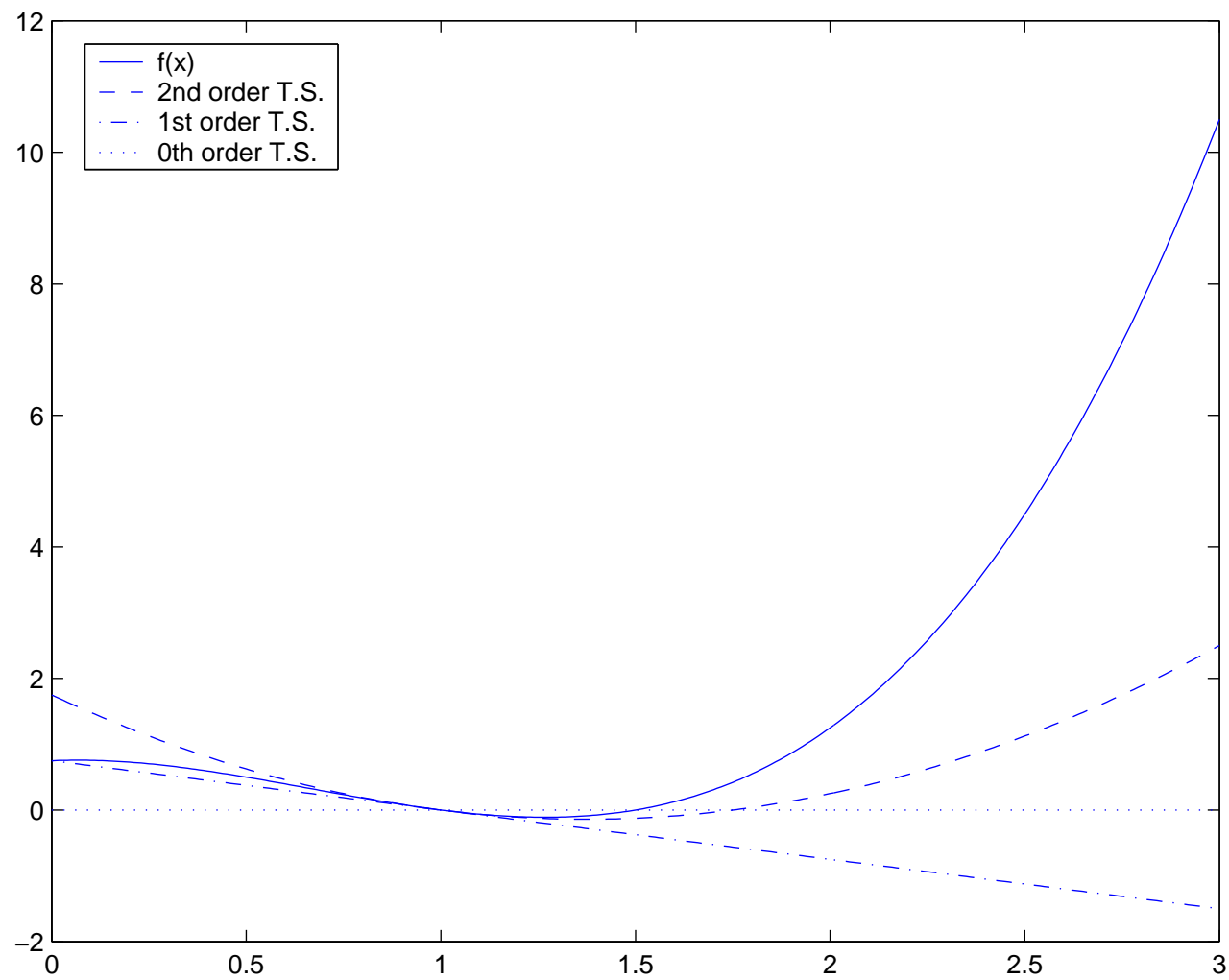


Figure 21: Example 2