2.1 Bisection steps

(1). Select x_l and x_u such that the function changes signs, i.e.,

$$f(x_l) \cdot f(x_u) < 0$$

(2). Estimate the root as x_r given by

$$x_r = \frac{x_l + x_r}{2}$$

(3). Determine the next interval:

- If $f(x_l) \cdot f(x_r) < 0$, then the root lies in (x_l, x_r), set $x_u = x_r$ and return to Step (2).
- If $f(x_u) \cdot f(x_r) < 0$, then the root lies in (x_r, x_u), set $x_l = x_r$ and return to Step (2).
- If $f(x_u) \cdot f(x_r) = 0$, then the root has been found. Set the solution $x = x_r$ and terminate the computation.

Define *approximate percentage relative error* as

$$\epsilon_a = \left| \frac{x_r^{\text{new}} - x_r^{\text{old}}}{x_r^{\text{new}}} \right| \times 100\%$$

where x_r^{new} is the estimated root for the present iteration, and x_r^{old} is the estimated root from the previous iteration.

Define *true percentage relative error* as

$$\epsilon_t = \left| \frac{x_t - x_r}{x_t} \right| \times 100\%$$

or

$$\epsilon_a = \frac{x_u - x_l}{x_u + x_l} \times 100\%$$

That is, the relative error can be found before starting the iteration.

2.3 Finding the number of iterations

$$\frac{\Delta_x^0}{2^n} \le E_{a,d} \quad \Rightarrow \quad n \ge \log_2 \frac{\Delta_x^0}{E_{a,d}} = \frac{\log_{10}\left(\frac{\Delta_x^0}{E_{a,d}}\right)}{\log_{10} 2}$$

3 Newton-Raphson method

Then x_{i+1} can be solved as

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

Relative error: $\epsilon_a = \left| \frac{x_{i+1} - x_i}{x_{i+1}} \right| \times 100\%.$

Iterations stop if $\epsilon_a \le \epsilon_{threshold}$.

3.3 Error analysis of Newton-Raphson method using Taylor series

$$|e_{i+1}| \propto |e_i|^2$$

The error in the current iteration is proportional to the square of the previous error. That is, we have quadratic convergence with the Newton-Raphson method. The number of correct decimal places in a Newton-Raphson solution doubles after each iteration.

4 Secant method

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

Derivative Rule $\frac{d}{dx}\left[\frac{f(x)}{g(x)}\right] = \frac{g(x)f'(x) - f(x)g'(x)}{(g(x))^2}$

chain rule $\frac{d}{dx} f[g(x)] = f'[g(x)] \cdot g'(x)$

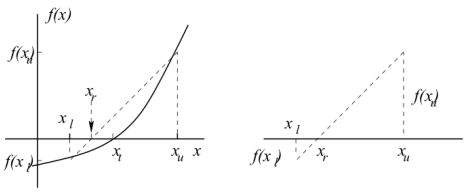
Derivative Rule $\frac{d}{dx} \ln(x) = \frac{1}{x}$

$\frac{d}{dx} \ln[f(x)] = \frac{1}{f(x)} f'(x)$

$\frac{d}{dx} \log_a(x) = \frac{1}{x \ln a}$

$\frac{d}{dx} \log_a[f(x)] = \frac{1}{f(x) \ln a} f'(x)$

5 False position method



False position steps:

(1). Find x_l, x_u, $x_l < x_u$, $f(x_l)f(x_u) < 0$

(2). Estimate x_r using similar triangles:

$$\frac{-f(x_l)}{f(x_u)} = \frac{x_r - x_l}{x_u - x_r}$$

$$x_r = x_u - \frac{f(x_u)(x_u - x_l)}{f(x_u) - f(x_l)}$$

(3). Determine next iteration interval

- If $f(x_l) \cdot f(x_r) < 0$, then the root lies in (x_l, x_r), set $x_u = x_r$ and return to Step (2).
- If $f(x_u) \cdot f(x_r) < 0$, then the root lies in (x_r, x_u), set $x_l = x_r$ and return to Step (2).
- If $f(x_u) \cdot f(x_r) = 0$, then the root has been found. Set the solution $x = x_r$ and terminate the computation.

(4). If $\epsilon_a < \epsilon_{threshold}$, $x = x_r$; else, back to (2).

False position method is one of the incremental search methods.

In general, false position method performs better than bisection method. However, special case exists (see fig. 5.14 in textbook).

6.1 Modified Newton-Raphson method

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

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

6 Handling repeated (multiple) roots

- $f(x) = (x-1)^3(x-3)$ has 3 repeated roots at $x = 1$
- $f(x) = (x-1)^4(x-3)$ has 4 repeated roots at $x = 1$

x_t is an unrepeated root of $u(x) = 0$

$f(x)$ has n repeated roots, $n \ge 2$

$$f(x) = g(x) \cdot (x - x_t)^n$$

where $g(x_t) \ne 0$. Then

$$\begin{aligned} f'(x) &= g'(x)(x - x_t)^n + g(x)n(x - x_t)^{n-1} \\ &= (x - x_t)^{n-1}\left[g'(x)(x - x_t) + ng(x)\right] \end{aligned}$$

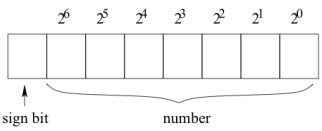
and

$$\begin{aligned} u(x) &= \frac{f(x)}{f'(x)} \\ &= \frac{g(x) \cdot (x - x_t)^n}{(x - x_t)^{n-1}\left[g'(x)(x - x_t) + ng(x)\right]} \\ &= \frac{g(x) \cdot (x - x_t)}{g'(x)(x - x_t) + ng(x)} \end{aligned}$$

Therefore, x_t is an unrepeated root of $u(x) = 0$.

Signed magnitude method

Examples: 8-bit representation



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 $2^{n-1}-1$.

Normalized representation:

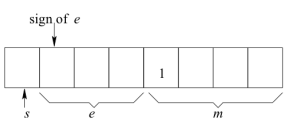
In general, a real number x can be written as

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

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

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.

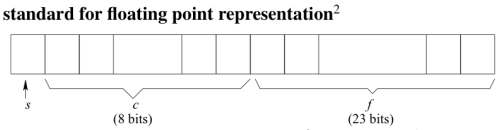


$s = 0$
$m = (1.011)_2$
$e = 1$
$c = e + 127$
$c = 128$

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

2^1 2^0 2^{-1} 2^{-2}

IEEE standard for floating point representation[2]



In IEEE 32-bit floating point format, $2 \times (2^8 - 2) \times 2^{23} + 1$

- 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), $0 \leq f < 1$.

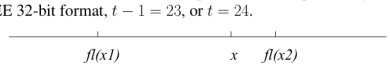
1.4 Floating point errors

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 $f = (11\ldots1)_2$ (or $m = (1.11\ldots1)_2$),
  $$U = m \cdot b^e = (2 - 2^{-23}) \times 2^{127} \approx 3.4028 \times 10^{38}$$
- 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 $f = (00\ldots0)_2$ (when $m = (1.00\ldots0)_2$). Then
  $$L = m \cdot b^e = 1 \times 2^{-126} = 1.1755 \times 10^{-38}$$

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 $t = 24$.



Machine precision: Define machine precision, ϵ_{mach}, as the maximum relative error. Then

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

floating → abs error $2^{-t} \cdot 2^e$, relative error 2^{-t}

For IEEE standard, $t - 1 = 23$ or $t = 24$, $\epsilon_{mach} = 2^{-24} \approx 10^{-7}$ for rounding.
The machine precision ϵ_{mach} is also defined as the smallest number ϵ such that $fl(1+\epsilon) > 1$, i.e., if $\epsilon < \epsilon_{mach}$, then $fl(1+\epsilon) = fl(1)$.

$$L < \delta < \epsilon_{mach}$$

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

$$\begin{aligned} 1 + \delta &= 1.0 \times 2^0 + 1.0 \times 2^{-25} \\ &= (1.\underbrace{00\cdots0}_{24\ 0's}1)_2 \times 2^0 \end{aligned}$$

$$fl(1+\delta) = 1$$

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

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

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.

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

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

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

Problem 3.
(8 Points) Consider a computer that uses 8 bits to represent floating-point numbers, 1 bit for s, 4 bits for c ($c = e + 7$), and 3 bits for f. In terms of s, e, and f, the base 10 numbers are given by $x = (-1)^s 2^e (1 + f)$, c is non-negative, and $0 \leq f < 1$.

(a) What are the smallest and largest positive numbers that can be represented accurately on this computer?
(b) What is the machine precision for this computer?
(c) How many base-10 numbers can be accurately represented on this computer?

$m = t + 1 + f$

(a) s | c | f — error happens

$x = (-1)^s 2^e (1+f)$

$s = 0$, $c = 0000$, $\because c = 0$, $c = e + 7$, $\therefore e = -7$, $f = 000$
$x = (-1)^s 2^e (1+f) = (-1)^0 2^{-7} (1)$

$s = 0$, $c = 1110$, $\because c = 14$, $c = e + 7$, $\therefore e = 7$, $f = 111$
$x = (-1)^s 2^e (1+f) = (-1)^0 2^7 (1 + 0.875)$

(b) $\epsilon_{machine} = \begin{cases} \text{rounding} & 2^{-4} \\ \text{chopping} & 2^{1-4} \sim 2^{-3} \end{cases}$

(c) $2 \times (2^4 - 2) \times 2^3 + 1$

$2^{-1}\ 2^{-2}\ 2^{-3}\ 2^{-4}\ 2^{-5}$

If error in 2^{-5}, $\frac{1}{32}$ (in binary)
$.x_1 x_2$ (in decimal) $\frac{1}{100}$
$\frac{1}{32} > \frac{1}{100}$ → 能保证第一位精确
precision in decimal 第二位不一定精确

滿足二進制小數不循環, fraction $\left(\frac{1}{2^n}\right)$

$(101.01)_{10}$ → 循環 $(101.0625)_{10}$ → 不循環