For most of our real-life problems when represented/realized using mathematical models, . . solutions are often analytically intractable. What do we do?

Thus, we resort to numerical approximation for system solution, and we use computers to perform the tasks for us.

⇓

This seems fairly okay as long as the system is modeled with accurate physics of the system of our interest.

However,

A primary problem/enemy in this approach is that—

We, generally, believe that, as it is calculated by computer, so it must be right.

Let's consider a few cases —

① Find roots of $x^2 - 5x + 6 = 0$

② Sort a sequence of numbers $[7, 5, 3, 11, 2]$
- → ascending order
- → descending order

We are cautious about the outcome, but for integer valued problems/outcomes, our belief that a calculation by the computer is right, holds.

But

We can no-longer belief our computational models, programs producing real numbers/values.

~~Because,~~

Let's see an example —

On Feb 25, 1991, during the Gulf war, American missile (patriot missile) battery in Dharan, Saudi Arab, failed to interept an Iraqi Scud Missile.

The scud attacked an American Army barracks, and killed 28 soldiers.

Why did the patriot missile fail?

Answer is —

Because of computer arithmetic error

What was the error?

* Calculation used 24 bit fixed point register.
* So, $1/10$ binary representation chopped recurring bits after the register limit is over.
* $(1/10)_2 = 0.0\underbrace{0011\,0011\,0011\,0011\,0011\,0011}_{24}0011\ldots$

↓ chopped after 24 bits

loose information!

error introduced

↳ 0.00000000000000000....11001100

↓ decimal equivalent

0.000000095

So, less accurate time-calculation.

- The patriot battery was on for about 100 hours.

So, simple calculation shows

$$0.000000095 \times 100 \times 60 \times 60 = 0.034$$

As the measurement was in $1/10^{th}$, it multiplies by a factor 10.

So, time calculation error was 0.34 sec.

- Scud travels at a velocity around 1.6 Km per second.

So, in 0.34 second the scud could travel about half a Kilometer

↓ As a result,

The scud was outside the range of the patriot.

And in the investigation report, it was written that

... consequently, performing the conversion after the patriot has been running continuously for extended periods cause the range gate to shift away from the center of the target.

Another Example :

Explosion of Ariane 5

* European Space agency launched an unmanned rocket on June 4, 1996, namely the Ariane 5.

* It was the rocket's first voyage, and it took a decade and about $7 billion to develop the rocket.

Why did the explosion happen ?

It was a case of software failure.

Precisely, a 64 bit floating point number relating to horizontal velocity of the rocket was converted to a 16 bit signed integer.

And the number was larger than the maximum storeable (32,768) integer in a 16 bit signed integer.

The conversion failed — this led to a software failure.

And the rocket of $7 billion crashed

Overall, the take away message is —

- We need to be very cautious for scenario where algoriths involve floating point calculation

- Just because that computation is done by computers, we should not be very obvious or take it as granted that —

  the calculation is the correct answer

Therefore, we should have some idea on

- HOW COMPUTERS REPRESENT AND CALCULATE WITH REAL NUMBER VALUES.

For instance,

With our general knowledge on algebra, we always have things such as

$$2+2=4, \quad 5\times 4\cdot = 2\cdot, \quad (\sqrt{3})^2 = 3$$

But, with computer arithmetic, we expect exact results for $2+2=4$, $5\times 4=20$

However, $(\sqrt{3})^2 = 3$, we will not get.

We need to have some idea related to the finite-digit arithmetic and how the computer arithmatic is done when the calculation generates real numbers.

Number system

$\mathbb{R} = \{-3, -2 \ldots 0, 1, \sqrt{2}, 5 \ldots\}$
all numbers

Let's consider a binary number 111.11.
It's valve in Decimal is

Binary Representation:

① Positive integers
  * Unsigned Binary

② Negative integers
  * Sign - Magnitude of the numbers

What about Fractions?

There are two specific representation —

① Fixed point

* Binary digits are fixed.

For example

1	1	1	0	.	0	1	0
2^3	2^2	2^1	$2^0 \times 0$	.	$(2^{-1} \times 0)$	$(2^{-2} \cdot 1)$	$(2^{-3} \cdot 0)$
8	4	2	0	.	0	$\frac{1}{4}$	0

So, Decimal value

$8 + 4 + 2 + 0 \cdot \frac{1}{4}$

$= 14.25$

Class work:

$(111.11)_2 \equiv (\quad)_{10}$

KEY: Each position is twice the value of the position to the right

What is the binary equivalent of 137

Division by 2	Quotient	Remainder	Bit No.
13/2	6	1	0
6/2	3	0	1
3/2	1	1	2
1/2	0	1	3

Scan ↑ (Bit No. 0 = Least significant; Bit No. 3 = Most significant)

So, $(13)_{10} \equiv \cancel{(1011)_2}\ (1101)_2$

TRY ANOTHER

$(0.125)_{10} \equiv (\quad)_2$

Multiply by 2	Integer Part	Fraction Part	Bit No. (after decimal point)
0.125 × 2	0	0.250	1
0.250 × 2	0	0.500	2
0.500 × 2	1	0.000	3

Scan ↓

So, $(0.125)_{10} \equiv (0.001)_2$

$2^{-1} \times 0 + 2^{-2} \times 0 + 2^{-3} \times 1$

$= 0 + 0 + \frac{1}{8} =$ 0.125 verified

Convert $(10.235)_{10}$ to $(\quad)_2$

First the $(10)_{10}$ to $(10)_2$

Div. by 2	Quotient	Remainder	Bit No.
$10/2$	5	0	0
$5/2$	2	1	1
$2/2$	1	0	2
$1/2$	0	1	3

So, $(10)_{10} \equiv (1010)_2$

Now, the fraction Part

0.235×2	0	0.470
0.470×2	0	0.940
0.940×2	1	0.880
0.880×2	1	0.760
0.760×2	1	0.520
0.520×2	1	0.040
0.040×2	0	0.080
0.080×2	0	0.160
⋮	0	
	0	
	⋮	

$\therefore (10.235)_{10} \equiv$

$(1010.\ 001111\ 0000101000\ 1111)$

Fixed Point Numbers

Let's say,

Fixed point representation using 4 integer bits & 3 fraction bits

Then, Given 0110110

↓ interpreted as

0110.110

↓ Decimal equivalent

$2^3\times0 + 2^2\times1 + 2^1\times1 + 2^0\times0 \cdot 2^{-1}\times1 + 2^{-2}\times1 + 2^{-3}\times0$

$\equiv 4+2 \cdot \frac{1}{2}+\frac{1}{4} = \frac{3}{4}$

$\equiv 6 \cdot 75$

That is,

Binary point is not the part of representation, but is implied.

Signed Fixed Point Number

- Signed-Magnitude Notation
- Two's complement
  - ↳ Subtraction & addition of +ve and -ve numbers can be done using same circuit in CPU

Represent -6 in two's complement?

	32	16	8	4	2	1
+6			0	1	1	0
complement			1	0	0	1
add 1 to LSB						1
			1	0	1	0

↳ -6

+13	0	1	1	0	1
	1	0	0	1	0
					1
	1	0	0	1	1

↳ -13

Real numbers and floating point

$\mathbb{R}$ → denotes set of all possible real numbers

$\mathbb{R} \equiv \{-5, -4, -\sqrt{2}, \ldots 0, 1, 5, \sqrt{35}, \ldots .\}$

Floating point:

- There's no fixed number of digits before or after the decimal point.

  That is, the decimal point can float

  ↳ symbol used to separate integer and fractional part.

- Radix point moves left or right based on the exponent value.

FLOATING POINT NUMBERS.

FLOATING POINT NUMBER

⊞ Given a number represented as $\beta^e \times d_0 . d_1 \ldots d_{p-1}$ we call β the base (or, radix) and p bits are dedicated for precision.

√ Most modern machines use $\beta = 2$ representations; this is also known as binary representation.

* However, floating point representation can use both binary or decimal system. $\beta = 2$ (binary)

$\beta = 10$ (Decimal)

Floating Point Representation

IEEE 754 standard

Binary standard

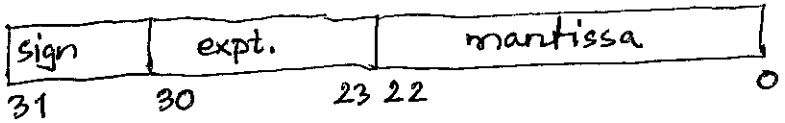
Single precision

$B=2$, $p=24$

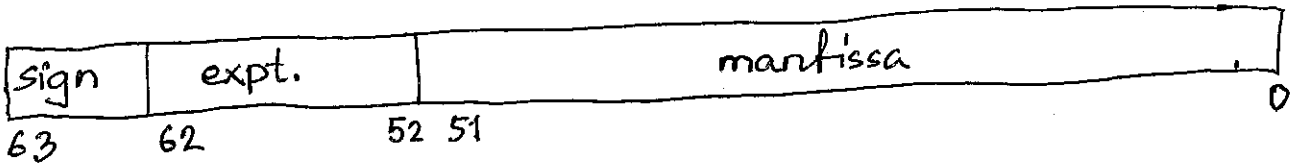
Double precision

$B=2$, $p=53$

Single Precision:



Double Precision:



Floating - Point Number System

A generalized Floating Point System $F(\beta, K, m, M)$, is a subset of the Real Number System, where

β : Base
K : number of digits in the base β expansion
m : minimum exponent
M : maximum exponent

→ Allows minimum/maximum nonzero elements.

- underflow → if number is too small to be presented → We use zero instead
- overflow → If number is too large to be presented → Halt execution.

consider a floating system

$F(10, 2, 0, 2)$

- 10 → Base
- 2 → No. of digit in base expansion
- 0 → minimum exponent
- 2 → maximum exponent.

Say, $\sqrt{7.1} \approx 2.66458$

Here, 2.66458 takes more than 2-digits and hence, it is not the member of

$F(10, 2, 0, 2)$

Represent $(-58.25)_{10}$ using IEEE 754 32 bit format

Div. by 2	Quotient	Remainder	Bit No
$58/2$	29	0	0
$29/2$	14	1	1
$14/2$	7	0	2
$7/2$	3	1	3
$3/2$	1	1	4
$1/2$	0	1	5

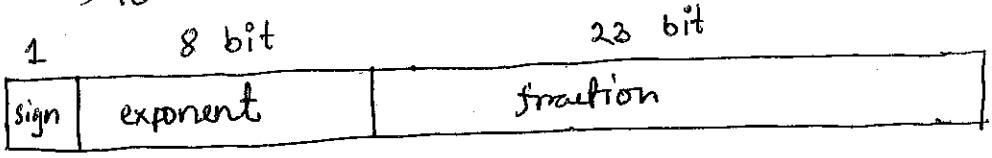
$(58)_{10} = (111010)_2$

Mult. by 2	Integar Part	Fraction Part	Bit no.
0.25×2	0	0.5	1
0.5×2	1	0.00	2

$(0.25)_{10} = (01)_2$

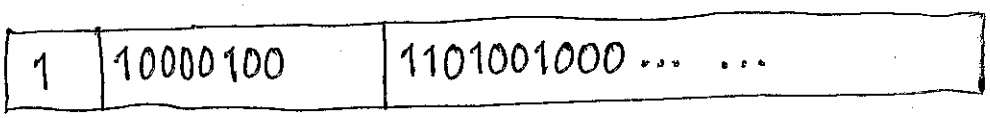
So, $(58.25)_{10} = (111010.01)_2 \equiv 1.1101001 \times 2^5$

IEE754, 32 bit :



here, sign bit 1 (Negative)
exponent $(5+127) = 132$
fraction 1101001000....

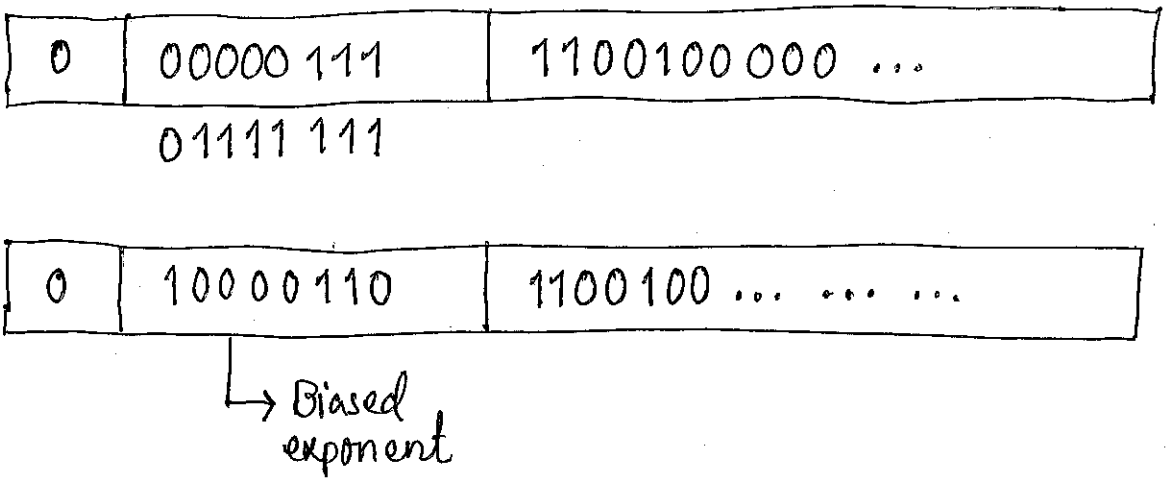
Finally,



IEEE standard

* IEEE 754 floating point

$(127)_{10} = (01111111)_2$ ⟶ Used for exponent bias.



so, exponent "7" was stored as $127 + 7 = 134$

Q. Why do we use biased exponent?

comparison of numbers become harder with two's complement

- Exponents have to be positive and negative to represent very high and small values
  With 2's complement, it is harder.

- Exponent is stored as an unsigned value and the bias is ~~slot~~ subtracted when the conversion to an exponent is done.

Single precision: Bias 127. exponent is stored 1 to 254.
So, $254 - 127 = +127$ and $-$ ⇒ -126

Double precision: 11 bit exponent

Quad precision: 15 bit exponent

Subtraction
$2^{n-1} - 1$
$= 2^{8-1} - 1 = 127$
$= 2^{11-1} - 1 = 1023$

So, $\sqrt{7.1} \approx 2.66458$ and as maximum 2 digit base expansion is allowed, we must discard extra digits to fit in $F(10, 2, 02)$.

if $\sqrt{7.1} \approx 2.6$ chopping the number

$\sqrt{7.1} \approx 2.7$ rounding off the number

- Regardless of the fact that a number is chopped/rounded for floating point conversion, it introduces error.

  ↓

  Roundoff Error

- Definition: Assume p : True value
  and p^* : Approximated value

So, Absolute error, $|p^* - p|$

Relative error, $\dfrac{|p^* - p|}{|p|}$

-

$|2.6 - 2.66458| = 6.458 \times 10^{-2}$. Absolute error

$\dfrac{|2.6 - 2.66458|}{2.66458} = 0.0242 = 2.42\%$ relative error.

Accumulation of Roundoff Error

Floating point arithmetic:

$$x \,@_{fl}\, y = fl\big(fl(x) \,@\, fl(y)\big)$$

Steps:

1. Each operand is replaced by floating point equivalent.
2. Exact arithmetic is performed
3. Result is replaced by its floating point equivalent.

$$x \,@_{fl}\, y - x @ y = fl\big(fl(x) @ fl(y)\big) - x @ y$$

$$= \underbrace{\Big[fl\big(fl(x) @ fl(y)\big) - fl(x) @ fl(y)\Big]}_{\text{introduced error}} + \underbrace{\Big[fl(x) @ fl(y) - x @ y\Big]}_{\text{Propagated error}}$$

- Introduced error is generally small.
- The ... error can be large.

Example:

$\delta_x = fl(x) - x \longrightarrow$ Absolute deviation

$\epsilon_x = \dfrac{fl(x) - x}{x} = \dfrac{\delta_x}{x}$ relative deviation

So, $fl(x) = x + \epsilon_x \cdot x$

$fl(y) = y + \epsilon_y \cdot y$

$fl(x) \times fl(y)$

$= (x + x.\epsilon_x)(y + y.\epsilon_y)$

$= xy + xy\epsilon_x + xy\epsilon_y + xy\epsilon_x \cdot \epsilon_y$

here, $\epsilon_{xy} = \epsilon_x + \epsilon_y + \epsilon_x \cdot \epsilon_y$

$\approx \epsilon_x + \epsilon_y$

$= xy + xy\epsilon_x\epsilon_y + xy(\epsilon_x + \epsilon_y)$

so, $= xy(1 + \epsilon_{xy})$

$$fl(x) + fl(y) = [x+\delta_x] + [y+\delta_x]$$
$$= (x+y) + (\delta_x + \delta_y)$$
$$= (x+y)\left(1 + \frac{x\epsilon_x + y\epsilon_y}{x+y}\right)$$

For instance,

Adding two almost equal number with opposite sign. Then, relative error will be large

↓ Propagation error

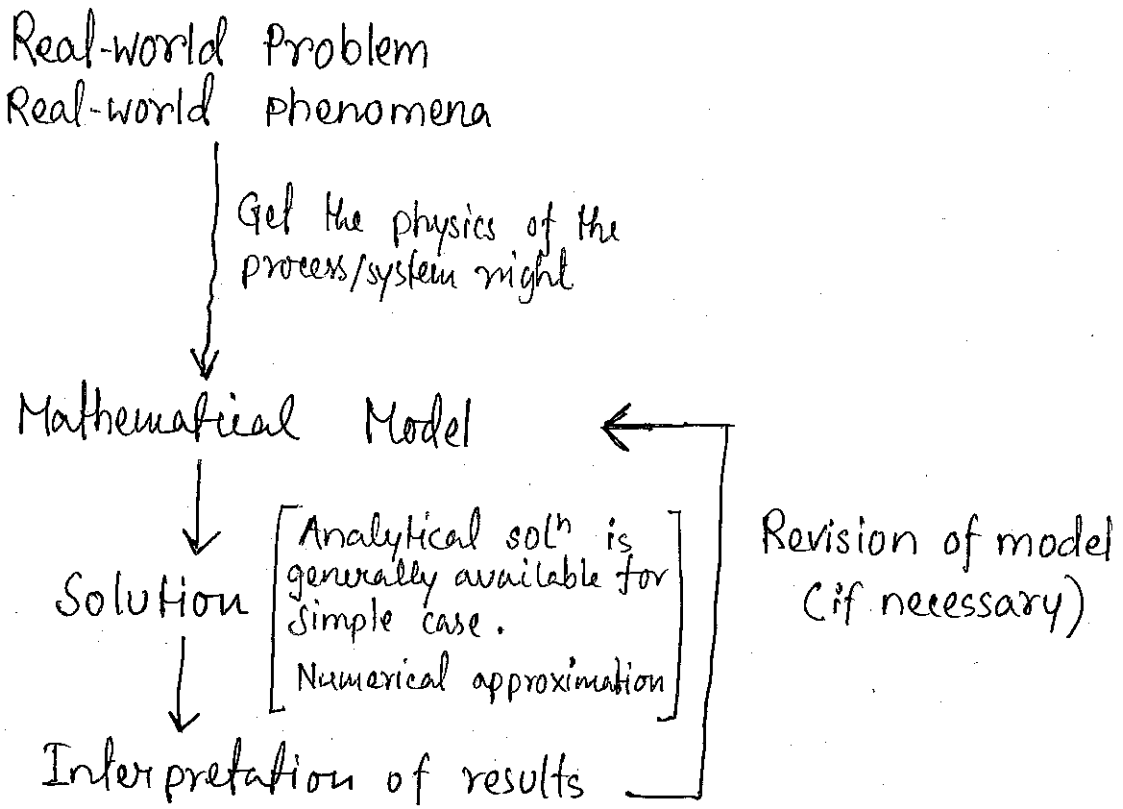
Drastic propagation error

↓ May result to

Loss of significant digits in calculation.

Thus, Algorithms should avoid subtraction of nearly equal numbers whenever possible.

Mathematical Model



ALGORITHMS

An algorithm is a precisely defined sequence of steps for performing a specified task.

Generally, all algorithms have three basic components :

1. List of input parameters
2. Specific operations to be performed and the order needs to be followed to perform
3. Component to report the outcome

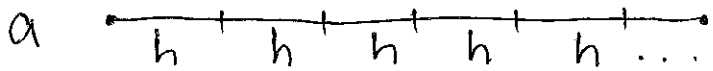
Example : Single step Algorithm

Let's consider the definite integral. $\int_a^b f(x)\,dx$

* We partion the range [a b]

$$a = x_0 < x_1 < x_2 \ldots < x_{n-1} < x_n = b$$

here, $x_i = a + i.h \quad \forall i.$ → for all



* We now use Trapezoidal rule approximation, and is given by

$$\int_a^b f(x)\,dx = \frac{h}{2}\left[f(a) + 2\sum_{i=1}^{n-1} f(x_i) + f(b) \right]$$

Algorithm :

Given	a, b, f, n
STEP 1	Compute $h = \frac{(b-a)}{n}$; sum = 0
STEP 2	for i from 1 to n-1 compute $f(x_i)$ and add to sum
STEP 3	multiply sum by 2
STEP 4	add f(a) and f(b) to sum
OUTPUT	(h/2) sum

Only single approximation is done !

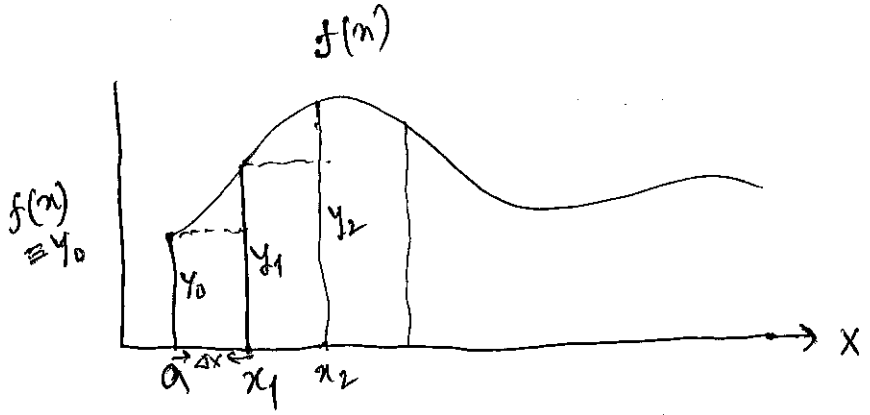
ITERATIVE Algorithm

- It generates a sequence of approximations which converges towards the true value.

There are a few issues that need attention.

1. Each algorithm must have a stopping criteria
2. If convergence rate is low/slow, an upperlimit of the number of max^m iteration should be set.
3. Avoid saving sequence from every iteration.

Trapezoidal Rule : Single iteration



$$\text{Area} = \text{Rectangle} + \text{triangle}$$

$$= y_0 \cdot \Delta x + \frac{1}{2}(y_1 - y_0) \cdot \Delta x$$

$$= \frac{2y_0\Delta x + y_1\Delta x - y_0\Delta x}{2}$$

$$= \frac{y_1\Delta x + y_0\Delta x}{2} = \frac{\Delta x}{2}(y_1 + y_0)$$

Example: Square Root Algorithm

We wish to approximate the value of $\sqrt{2}$

Given
- $a = 2$
- Start approximation x_0
- Convergence parameter ϵ
- Maximum iteration N_{max}

STEP 1 → for iter from 1 to N_{max}

STEP 2 | compute $x_1 = \left(x_0 + \frac{a}{x_0}\right)/2$

STEP 3 | if $|x_1 - x_0| < \epsilon$, OUTPUT x_1

STEP 4 | copy x_1 to x_0

→ end

OUTPUT "maximum number of iterations has been exceeded"

For any positive real number x_0, the sequence generated by the rule

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$$

for $n = 0, 1, 2, \ldots$ converges to $\sqrt{a}$

This algorithm generates a sequence of approximations which hopefully

CONVERGE

towards the desired solution.

Scientific Notation

Let us consider a number

$$100,000 = 0.100000 \times 10^{6}$$ (6 → exponent)

$$111,111 = \underbrace{0.111111}_{\text{mantissa}} \times 10^{6}$$ (6 → Base)

That is, in general, scientific notation of any number,

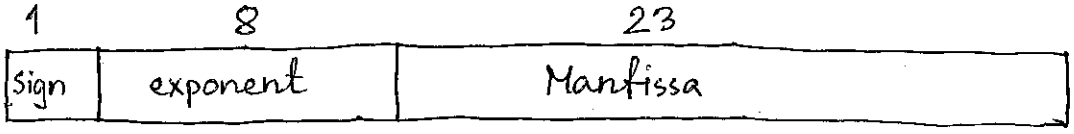
$$\pm M \times B^{E}$$

M → Mantissa
B → Base
E → Exponent

So,

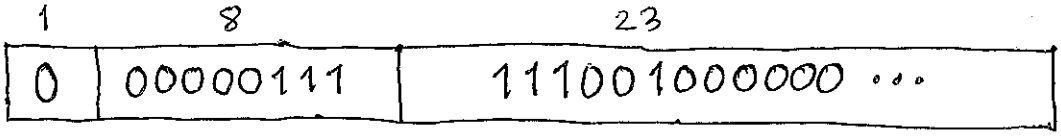
$$232 = 2.34 \times 10^{2}, \text{ where } M = 2.34,\ B = 10 \text{ and } E = 2.$$

Structure of Floating-point number



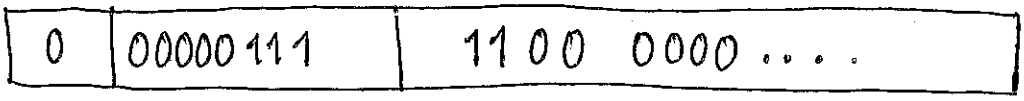
Consider $(228)_{10} = (11100100)_{2} = 1.1100100 \times 2^{7}$

$\equiv$ Exponent 7
Base 2



Another representation:

Consider First bit of mantiss is always "1"
So, storing the most significant is redundant.



Convergence

What does the word "converge" mean?

Many algorithms are iterative in nature

↳ Proceed towards the target soln through sequence of evaluations and reinforcement.

Generally, all iterative algorithms generate a sequence of approximations

↓ this eventually CONVERGE to

desired solution.

- Convergence rate & order are the two important performance criteria of any ~~numerical~~ algorithm.
- Generally, we choose a technique/algorithm whose sequence converges as rapidly as possible.

~~Rate of convergence~~

Let's assume $\{x_n\}$ is the sequence that converges to the value "L". So,

$$\lim_{n\to\infty} \{x_n\} = L, \text{ or } \lim_{n\to\infty} |x_n - L| = 0$$

Value at which sequence $\{x_n\}$ converges, "L", is called the limit.

A sequence for which $\lim_{n\to\infty} x_n$ does not exist is said to diverge

Rate of convergence :

Let $\{P_n\}$ be a sequence that converges to a true value "P".

If there exists a sequence β_n that converges to zero, and

a positive constant λ

↳ independent of "n" | implies

↓

At every computation step, λ remains unchanged.

Such that

$$|P_n - P| \leq |\beta_n| \lambda$$

for all sufficiently large values of "n"

Then $\{P_n\}$ is said to converge to "P"

with rate of convergence $O(\beta_n)$

↳ This is big-O notation

↳ Reads as, big-O of β_n

If $\{P_n\}$ converges to "P", you can write

$$P_n = P + O(\beta_n)$$

Important: Big-O provides a measure/reference for

How quickly the error approaches to zero.

Question : $O(1/n^2)$ & $O(1/n^{10})$ ✓✓ Which one is converging fast ?

- ~~In general, Expression $O(\beta_n)$~~

- Sequence $\{\beta_n\}$, typically, takes a form $\frac{1}{n^a}$ or $\frac{1}{a^n}$, where a is some positive constant.

Let's assume $a = 2$

So, we obtain $\frac{1}{n^2}$ or $\frac{1}{2^n}$

- Question $O\left(\frac{1}{n^2}\right)$ and $O\left(\frac{1}{2^n}\right)$

Identify the faster rate of convergence.

Answer: $O\left(\frac{1}{2^n}\right)$

↓

Example: Sequence (1) $\left\{\frac{n+3}{n+7}\right\}$ Sequence (2) $\left\{\frac{2^n+3}{2^n+7}\right\}$

What is the limit "L" here?

$\lim_{n\to\infty} \frac{n+3}{n+7} = 1$ $\lim_{n\to\infty} \frac{2^n+3}{2^n+7} = 1$ $L = 1$

So, both the sequences converge to the limit 1

But, which one is faster convergence?

Let's calculate ——

n	$(n+3)/(n+7)$	$(2^n+3)/(2^n+7)$
1	0·5000000000	0·5555555556
2	0·6555555556	0·6363636364
3	0·6000000000	0·7333333333
⋮	⋮	⋮
9	0·7500000000	0·9922928709

Compare this two

∴ It is evident that sequence $\left\{\frac{2^n+3}{2^n+7}\right\}$ converges fast

HW: Represent the individual rate of convergence of sequences $\left\{\frac{n+3}{n+7}\right\}$ and $\left\{\frac{2^n+3}{2^n+7}\right\}$ using Big-O notation [?]

Individual Rate of convergence:

Sequence $\underbrace{\left\{\frac{n+3}{n+7}\right\}}_{x_n}$:

~~$\left|\frac{n+3}{n+7}\right|$~~

We had $|x_n - L| \equiv \left|\frac{n+3}{n+7} - 1\right| = \left|\frac{n+3-n-7}{n+7}\right|$

$$= \left|\frac{-4}{n+7}\right| = \frac{4}{n+7}$$

$$\Rightarrow \frac{4}{n+7} < \frac{4}{n} = 4 \cdot \frac{1}{n}$$

Now, consider $\lambda |\beta_n| \equiv 4 \cdot \frac{1}{n}$

So, Rate of convergence $O(1/n)$

For the sequence $\left\{\frac{2^n+3}{2^n+7}\right\}$

So, $\left|\frac{2^n+3}{2^n+7}-1\right| = \left|\frac{2^n+3-2^n-7}{2^n+7}\right| = \left|\frac{-4}{2^n+7}\right|$

$$= \frac{4}{2^n+7} < \frac{4}{2^n}$$

So, $\frac{4}{2^n} = 4 \cdot \frac{1}{2^n} \equiv \lambda \left| \cdot \frac{1}{2^n} \right|$

Thus, Rate of convergence $O\left(\frac{1}{2^n}\right)$

* These results confirm the numerical evidence that $\frac{1}{2^n}$ approaches zero faster as $n \to \infty$

RATE OF CONVERGENCE

Let us consider f be a fn^c defined on the interval (a, b) that contains $x = 0$, and suppose $\lim_{x \to 0} f(x) = L$,

↳ limit

Now, if there exists a fn^c $g(x)$, for which $\lim_{x \to 0} g(x) = 0$, and a positive constant K such that

$$|f(x) - L| \leq K\, g(x)$$

for all sufficiently small values of x,

Then,

$f(x)$ is said to converge to L with "Rate of convergence" $O(g(x))$

↳ O-notation for functions

↳ $g(x)$

Example :

$$f(x) = \frac{\cos x - 1 + \frac{1}{2}x^2}{x^4}$$

What is the Limit of "f" as $x \to 0$?

What is the rate of convergence "f" to limit L?

Solution: From Taylor's expansion, we obtain

$$\cos x = 1 - \frac{1}{2}x^2 + \frac{1}{24}x^4 - \frac{1}{720}x^6 \cos\xi$$

for some ξ between 0 and x.

$$\Rightarrow \frac{\cos x - 1 + \frac{1}{2}x^2}{x^4} = \frac{1}{24} - \frac{1}{720}x^2 \cos\xi$$

$$\Rightarrow \frac{\cos x - 1 + \frac{1}{2}x^2}{x^4} - \frac{1}{24} = -\frac{1}{720}x^2 \cos\xi$$

$$\Rightarrow \left| \frac{\cos x - 1 + \frac{1}{2}x^2}{x^4} - \frac{1}{24} \right| = \frac{1}{720} \left| x^2 \cos\xi \right|$$

by taking absolute values

$$\Rightarrow \left| \frac{\cos x - 1 + \frac{1}{2}x^2}{x^4} - \frac{1}{24} \right| \leq \frac{1}{720} |x^2|$$

$\downarrow$ follows that

$\lim_{x \to 0} f(x) = \frac{1}{24}$ and Rate of convergence is $O(x^2)$