Welcome to Ma221! Lecture 3 Fall 2024

Floating Point + Error Analysis

FP: how to represent real numbers

Long ago (< 1985) many computers did arithmetic differently $\Rightarrow$ hard to write portable code $\Rightarrow$ IEEE 754 FP Standard, Led by Prof Kahan

First standard 1985, then 2008, 2019, 2029...

Prizes: Turing Award 1989
IEEE Milestone 2023

Scientific Notation $\pm d.d\cdots d \cdot \text{radix}^e$

Usually radix = 2 (or 10 for finance)

Store: signbit ($\pm$)
exponent (e integer)
mantissa: ($d.d \ldots d$)
p = # digits in mantissa

9 decimal digits $\rightarrow$ $9 \cdot 4 = 36$ bits $\rightarrow$ $\left(\frac{10}{16}\right)^9 = .015$ memory usage

9 decimal digits $\rightarrow$ 3 groups of 3 digits
$\rightarrow$ 3 groups of 10 bits
$\rightarrow$ 30 bits $\rightarrow$ $\left(\frac{1000}{1024}\right)^3 = .93$ memory usage

Both p and #bits in e are limited
to fit 64 bits, 32 bits, 16 bits, 128 bits
Historically hardware support for 32, 64 bits
128 bit usually in software
Now 16 bit popular for ML
Lots of accelerators for matmul using
16 bits
Some use bfloat16 ("brain float")
eg Google TPUs (Tensor processing units)

Error analysis will only depend on knowing
p and #bits in e for each format

For now, ignore "exceptions" ie
numbers too big or too small to
represent using e

Normalization: use
3.100e0 or .0031e3 ?
assume leading digit is nonzero
Normalization ⇒ unique representation
and in binary → leading digit = 1
→ don't need to store it
→ free extra bit!
"hidden bit"

Def: rnd(x) = nearest FP number to x

(need to break ties: default rule: "round to nearest even" → round to nearest number ending in zero

good idea: unbiased, half time round up, half time down, more accurate for long sums

eg google Vancouver Stock Exchange

Def: Relative Representation Error

$$RRE(x) = \frac{|x - rnd(x)|}{|rnd(x)|}$$

Def: Maximum RRE $= \max_{x \neq 0} RRE(x)$

aka machine epsilon, macheps, ε

(Matlab: their eps $= 2 \cdot \varepsilon$)

Max RRE = half distance from 1 to next larger FP number $= 1 + 2^{(1-p)}$

$= 2^{-p}$ in binary

Roundoff model (no over/underflow)

(*) $fl(a \text{ op } b) = rnd(a \text{ op } b)$

= true result rounded to nearest even

$= (a \text{ op } b)(1+\delta), \ |\delta| \le \varepsilon$

$op \in \{+,-,*,/\}$

(*) also true for complex arithmetic with larger ε (see Q1.12 for details)
(C standard has 30+ lines of code for complex multiply)

Existing IEEE binary Formats

single (S) = 32 bits, double (D) = 64 bits
quad (Q) = 128 bits, half (H) = 16 bits

S: 32 bits = 1 sign
+ 8 exponent
+ 23 mantissa

$\Rightarrow p = 1+23 = 24 \Rightarrow \varepsilon = 2^{-24} \sim 6 \cdot 10^{-8}$

OV = overflow threshold $\sim 2^{128} \sim 10^{38}$
UN = underflow " $\sim 2^{-126} \sim 10^{-38}$

D: $64 = 1 + 11 + 52 \Rightarrow p = 53$
$\Rightarrow \varepsilon = 2^{-53} \sim 10^{-16}$
$OV \sim 2^{1024} \approx 10^{308}$ $UN \sim 2^{-1022} \sim 10^{-308}$

Q: $128 = 1 + 15 + 112 \Rightarrow \varepsilon \approx 10^{-34}$
$OV \sim 10^{4932} \sim \frac{1}{UN}$

H: $16 = 1 + 5 + 10 \Rightarrow \varepsilon \cdot 5 \cdot 10^{-4}$
$OV \sim 10^{4} \sim \frac{1}{UN}$

Bfloat: $16 = 1 + \underbrace{8}_{\text{same as S}} + 7 \sim \varepsilon \sim 4 \cdot 10^{-3}$

Higher precision packages for arbitrary precision in software, see web page
(ARPREC, GMP, ...)

Error analysis for Horner: $p(x) = \sum_{i=0}^{d} a_i \cdot x^i$

$p = a_d$
for $i = d-1 : -1 : 0$
$\quad p = x \cdot p + a_i$

Label intermediate terms

$p_d = a_d$
for $i = d-1 : -1 : 0$
$\quad p_i = x \cdot p_{i+1} + a_i$

Introduce roundoff

$p_d = a_d$
for $i = d-1 : -1 : 0$
$$p_i = [x \cdot p_{i+1}(1+\delta_i) + a_i](1+\delta_i')$$
$$|\delta_i| \le \varepsilon \quad |\delta_i'| \le \varepsilon$$

Simplify

$$p_0 = \sum_{i=0}^{d-1} a_i \cdot x^i \left[(1+\delta_i') \prod_{j=0}^{i-1} (1+\delta_j)(1+\delta_j')\right]$$
$$+ \; a_d \cdot x^d \left[\prod_{j=0}^{d-1} (1+\delta_j)(1+\delta_j')\right]$$
$$= \sum_{i=0}^{d-1} \left([\text{product of } 2i+1 \text{ terms like } 1+\delta] a_i \right) x^i$$

$+ \left(\left[\text{product of } 2d \text{ terms like } 1+\delta \right] a_d \right) x^d$

$= \sum_{i=0}^{d} a_i' x^i \qquad a_i' = a_i \left[\begin{array}{c} \text{product of} \\ \text{at most } 2d \text{ terms} \\ \text{like } 1+\delta \end{array} \right]$

In words: Horner's rule is backward stable: returns exact value of a polynomial with slightly different coefficients

Simplify to get error bound:

$$\prod_{i=1}^{n} (1+\delta_i) \le \prod_{i=1}^{n} (1+\varepsilon) = (1+\varepsilon)^n$$

$$= 1 + n\varepsilon + O(\varepsilon^2)$$

usually ignore $O(\varepsilon^2)$

$$\le 1 + \frac{n\varepsilon}{1-n\varepsilon} \quad \text{if} \quad n\varepsilon < 1$$

... proof left to you!

$$\prod_{i=1}^{n} (1+\delta_i) \ge \prod_{i=1}^{n} (1-\varepsilon) = 1 - n\varepsilon + O(\varepsilon^2)$$

$$\ge 1 - \frac{n\varepsilon}{1-n\varepsilon}, \quad n\varepsilon < 1$$

$$\Rightarrow \left| \prod_{i=1}^{n} (1+\delta_i) - 1 \right| \le n\varepsilon \quad (+ O(\varepsilon^2))$$

Apply to Horner: $|\text{computed } p_0 - p(x)|$

$$\le \sum_{i=0}^{d-1} (2i+1)\varepsilon |a_i x^i| + 2d\varepsilon |a_d \cdot x^d|$$

$$\text{relative error} = \frac{|\text{computed } p_0 - p(x)|}{|p(x)|}$$

$$\le \underbrace{\frac{\sum_{i=0}^{d} |a_i \cdot x^i|}{|p(x)|}}_{\text{condition number}} \cdot \underbrace{2d\varepsilon}_{\text{backword error}}$$

How many correct digits can I guarantee?

k correct digits $\Longleftrightarrow$ relative error bound $\le 10^{-k}$

$\Longleftrightarrow$ $-\log_{10}(\text{relative error}) \ge k$

Modify Horner to get error bound

$p = a_d$, $ebnd = |a_d|$

for $i = d-1 : -1 : 0$

$p = x \cdot p + a_i$, $ebnd = |x| \cdot ebnd + |a_i|$

$ebnd = ebnd \cdot 2 \cdot d \cdot \varepsilon$ … absolute bound

---

Details of FP:

- Important to understand to write reliable software
- lots of recent HW developments
- Analyzing code reliability hard
  - recent work on automatic tools to detect errors
- see posted notes

① Exception Handling - IEEE 754 rules:

Underflow: Tiny / Big = 0 or
"subnormal": special numbers
with smallest exponent, leading
zeros in mantissa (not normalized)

$\underline{0}.00010 \cdot 2^{minexp}$

no hidden bit

Use case: if $(x \neq y)$ $z = \frac{1}{x-y}$

no division by zero only if
you have subnormal numbers

Overflow or divide by 0

$\pm 1/0 = \pm \mathrm{Inf}$ = "Infinity"
special bit pattern reserved

Natural rules: Big + Big = Inf
$3 - \mathrm{Inf} = -\mathrm{Inf}$
$7/\mathrm{Inf} = 0$

Invalid $0/0$ = "NaN" = Not a number

Rules: Inf − Inf = NaN
$\sqrt{-1}$ = NaN
3 + NaN = NaN

Flags available to check if Inf or NaN appeared