Welcome to Ma 221!

Floating Point Error Analysis

Last time: evaluated \( p(x) = (x-2)^3 \) using Horner's rule: \( O(10^{-8}) \) errors, \( \gg |p(x)| \) near \( x = 2 \)

Floating Point - How to represent real numbers

Long ago (\( \leq 1985 \)) all computers were different \( \Rightarrow \) hard to write portable code

Prof. Kahan led IEEE Standard Committee.
First Standard: 1985 then 2008, 2019

Scientific Notation: \( \pm d.dd\ldots d \cdot \text{radix}^e \)

Usually \( \text{radix} = 2 \) (or 10, for finance)

Store sign bit (\( \pm \))
exponent (\( e \))
mantissa \( (d.dd\ldots d) \)
\( \rho = \# \text{ digits in mantissa} \)
Both $p$ and $\#\text{bit in } e$ are limited to fit in 16, 32, 64, 128 bits.

Historically, only hardware support for 32 (single) and 64 (double). Lately, 16 (half) popular for machine learning. Google, Nvidia, Intel... building accelerators for 16 bits.

Some use bfloat16, differs from IEEE 16 bit format in using more $e$ bits, smaller $p$. Current research on how to use bfloat16 for linear algebra.

For simplicity, initially ignore limits on size of $e$, so assume no overflow or underflow.

Normalization: use $3.100e0$, not $0.0031e3$; i.e., leading digit nonzero.
Normalization ⇒ unique representations
and ⇒ in binary, leading digit=1
⇒ don't need to store it
⇒ free extra bit of precision
"hidden bit"

Def: \( \text{rnd}(x) \equiv \text{nearest floating point number to } x \) (Note: Default IEEE rounding rule: for breaking ties: round to "nearest even"
⇒ last bit (digit) is even, i.e. 0 in binary

Def: Relative Representation Error (RRE)
\[
\text{RRE}(x) = \frac{|x - \text{rnd}(x)|}{|\text{rnd}(x)|}
\]

Def: Maximum RRE = \( \max_{x \neq 0} \text{RRE}(x) \)
aka machine epsilon, macheps, \( \varepsilon \)

Max RRE = half distance from 1 to next largest FP number = \( 1 + (\text{radix})^{1-p} \)
= \( .5 \cdot \text{radix}^{1-p} = 2^{p} \) in binary
Round-off model (no over/underflow)

\( f(a \text{ op } b) \equiv \text{rnd}(a \text{ op } b) \)

- true result rounded to nearest even
- \( = (a \text{ op } b)(1+\delta) \), \(|\delta| \leq \varepsilon\)
- \text{ op can be } +, -, *, /

(*) also true for complex arithmetic but with larger \( \varepsilon \) (Q 1.12 for details)

Existing IEEE binary formats
- single(S), double(D), quad(Q), half(H)

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

\[ p = 23 + 1 = 24 \Rightarrow \varepsilon = 2^{-24} \approx 6 \cdot 10^{-8} \]

-127 \leq e \leq 127 \Rightarrow OV = \text{overflow threshold} = 2^{128} \approx 10^{38}

-1022 \leq e \leq 1023 \Rightarrow UN = \text{underflow threshold} = 2^{-1022} \approx 10^{-308}

D: 64 = 1 + 11 + 52 \Rightarrow p = 53, \varepsilon = 2^{-53} \approx 10^{-16}

Q: 128 = 1 + 15 + 112 \Rightarrow \varepsilon = 10^{-34}

\[ OV = 10^{9932} \approx \frac{1}{UN} \]
\[ H: \ 16 = 1 + 5 + 10 \Rightarrow \varepsilon \approx 5 \cdot 10^{-4} \]
\[ 0V \approx 10^4, \ U_N \approx 10^{-4} \]

Bfloat16: \[ 16 = 1 + 8 + 7 \Rightarrow \varepsilon \approx 4 \cdot 10^{-3} \]
same as single

Lecture 1: “guaranteed correct” except in
“rare cases”: common approach is
to do most work in low precision
(fast) and few more steps in high
precision (e.g., a few steps of Newton)
to get “usual precision” when done,
as though all work done in single
(refs on class web page)

Higher precision than 128 (Quad)
available (ARPREC, GMP, links on
web page)

“Extended precision” (80 bit format)
in original IEEE floating point
standard, implemented in Intel x86,
now deprecated
Error Analysis:

\[ f(a \text{ op } b) = \text{round}(a \text{ op } b) = (a \text{ op } b)(1+\delta) \quad |\delta| \leq \epsilon \]

Horner's rule: to evaluate \( p(x) = \sum_{i=0}^{d} a_i x^i \)

Algorithm:

\[
p = a_d
\]

\[
\text{for } i = d-1 \Downarrow -1 \Downarrow 0 \\
\quad p = x \cdot p + a_i
\]

label intermediate terms:

\[
p_d = a_d \\
\text{for } i = d-1 \Downarrow -1 \Downarrow 0 \\
\quad p_i = x \cdot p_{i+1} + a_i
\]

introduce roundoff

\[
p_d = a_d \\
\text{for } i = d-1 \Downarrow -1 \Downarrow 0 \\
\quad p_i = \left[ x \cdot p_{i+1} (1+\delta_i) + a_i \right] (1+\delta'_i) \\
\quad |\delta_i| \leq \epsilon, \quad |\delta'_i| \leq \epsilon
\]

Simplify:

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

**In Words:** Horner's Rule is backward stable; returns exact value of a polynomial at \(x\) with slightly different coefficients \(a_i'\).

Simplify to get error bound

\[
\prod_{i=1}^{n} (1 + \delta_i) \leq \prod_{i=1}^{n} (1 + \varepsilon) = (1 + \varepsilon)^n \\
= 1 + n \varepsilon + O(\varepsilon^2) \\
\ldots \text{usually ignore } O(\varepsilon^2) \\
\leq 1 + \frac{n \varepsilon}{1 - n \varepsilon} \quad \text{if } n \varepsilon < 1 \\
\]

... proof left to students

\[
\prod_{i=1}^{n} (1 + \delta_i) \geq (1 - \varepsilon)^n = 1 - n \varepsilon + O(\varepsilon^2) \\
\geq 1 - \frac{n \varepsilon}{1 - n \varepsilon}, \quad n \varepsilon < 1 \\
\]

\[ \Rightarrow \left| \prod_{i=1}^{n} (1 + \delta_i) - 1 \right| \leq n \varepsilon \]
\[ |\text{computed } p_a - p(x)\| \leq \sum_{i=0}^{2}\epsilon(2i+1) \cdot |1 \cdot \|x^i\| + 2d \cdot |a_2x^d| \]

\[
\text{relerr} = \frac{|\text{computed } p_a - p(x)|}{|p(x)|}
\leq \sum_{i=0}^{d} |a_i| \cdot |x| \cdot 2d\epsilon
\leq \frac{\text{condition number}}{|p(x)|} \Rightarrow \text{backward error}
\]

How many correct digits can we guarantee?

\[ k \text{ correct digits} \iff \text{relative error bound} \leq 10^{-k} \]
\[ \iff -\log_{10} (\text{relative error}) \geq k \]

Modify Horner's Rule to get error bound:

\[ p = a_d \]

for \( i = d-1 \rightarrow 0 \)

\[ p = p + a_i \cdot x \cdot \text{ebnd}^i = 1 \cdot x \cdot \text{ebnd}^i + a_i \cdot x \cdot \text{ebnd}^i \]

\[ \text{ebnd} = \text{ebnd} \cdot 2 \cdot d \cdot 2 \ldots \text{absolute error bound} \]

(Fig 1.3 in text)
Foreshadows linear algebra

Horizontal axis still problem
to solve usually $n^2$ dimensions

Error increases the closer you get to “hardest possible problem”

for Horner's rule: computing $p(x)=0$

For matrix inversion: set of singular matrices: $n^2-1$ dimensional set in $\mathbb{R}^{n \times n}$: $\det(A)=0$

Later: compute distance from $A$
to nearest singular matrix (SVD)

For evaluating $p(x)$

relative error bound: $\frac{\frac{1}{2} \left| \lambda_1 \right| 1 \cdot |x|^\lambda}{(p(x))}$ \cdot 2 \text{de}$

\begin{align*}
\text{condition number} & \quad \text{backward error} \\
\text{condition } \# & \rightarrow \infty \quad \text{as } x \rightarrow 2 \\
\text{because } p(x) & = 0
\end{align*}
Same idea for linear algebra:
for $A^{-1}$: condition number
proportional to $\frac{1}{\text{distance to nearest singular matrix}}$

for $Ax=b$ what is backward error:
exact answer to $(A+E)x=b$ where $E$ "small" compared to $A$
(need matrix norms)
$\hat{x} - x = (A+E)^{-1}b - A^{-1}b$

Taylor expansion of $(A+E)^{-1}$

Homework: Extend error analysis of Horner to linear algebra:

Horner: $p=ad$, for $i=d-1:-1:0$, $p = x_{i}p + a_{i}$

product of $f_{i}$ and $y_{i}$, for $i=1:d$, $s = x_{i}y_{i} + s$

So error analysis similar

HW 1.10, 1.11
Details of Floating Point

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

1) Exception Handling

IEEE Standard has rules for

Underflow: Tiny/Big = 0, or
  "subnormal": special numbers with smallest exponent, with zero leading bits in mantissa
  0.00101 \cdot 2^{\text{min-exp}}

Overflow or Divide by 0
  \pm 1/0 = \pm \text{Inf} = "\text{Infinity}"
Natural rules:
\[ \frac{\text{Big}}{\text{Big}} = \text{Inf} \]
\[ 3 - \text{Inf} = -\text{Inf} \text{ etc} \]

Invalid
\[ 0/0 = \text{NaN} = "\text{Not a number}" \]
Rules: \[ \text{Inf} - \text{Inf} = \text{NaN}, \ 0 - 1 = \text{NaN} \]
\[ 3 + \text{NaN} = \text{NaN} \]

Flags available to check if an
\text{Inf} or \text{NaN} created

Impact of Exceptions on Software:

Reliability:
\[ \text{Compute: } S = \| x_1 \|_2 = \sqrt{x_1 \cdot x_1} \]
What could go wrong with
\[ S = 0, \text{ for } i = 1:n, \ S = S + x_i^2, \ S = \sqrt{S} \]
Overflow or underflow, could
give wrong answer, even if
true \[ S \text{ "ok", in BLAS} \]
Worst case examples: (see links on web page)

Crash of Ariane 5
Robotic car crash

Current work to make BLAS and LAPACK more reliable (class projects)

Error Analysis:
Before: \( \text{rnd}(a \op b) = (a \op b) \cdot (1 + \delta) \) 
\[ |\delta| \leq \varepsilon \]

With underflow
\[ \text{rnd}(a \op b) = (a \op b)(1 + \delta) + q \]
where \( |\delta| \leq \text{tiny number related to UN} \)

Speed:
Run “reckless” code, fast but ignores possible exceptions.

Check flags to see if exception occurred.

In rare case of exceptions, redo slowly/carefully.
(2) **High Precision:** Various packages see web page. Some special tricks for fast high precision addition. Optional HW Q 1.18

(3) **Lower precision (16 bit or less)**

arithmetic

how low can precision go and still be trustworthy?

lots of variations being considered

New (IEEE Standards committee on “Floating Point for ML”)

Is 8 bits enough? 4, 2, 1?

Variable precision

(5) **Reproducibility?**

getting same answer for each run, for debugging + reliability
Not guaranteed on modern architectures.

Parallel computers may perform sums in different orders:

\[
(1 - 1) + 10^{-20} = 10^{-20} 
eq 0 = 1 + (1 + 10^{-20})
\]

Lots of recent work on reproducible summation! (see class web page)

Latest IEEE standard has new instruction for this (class projects!)