AMS526: Numerical Analysis I (Numerical Linear Algebra)

Lecture 8: Floating Point Arithmetic; Accuracy and Stability

Xiangmin Jiao

Stony Brook University

Outline

Floating Point Representations

- Computers can only use finite number of bits to represent a real number
  - Numbers cannot be arbitrarily large or small (associated risks of *overflow* and *underflow*)
  - There must be gaps between representable numbers (potential round-off errors)
- Commonly used computer-representations are floating point representations, which resemble scientific notation

$$\pm(d_0 + d_1\beta^{-1} + \cdots + d_{p-1}\beta^{-p+1})\beta^e,\ 0 \le d_i < \beta$$

  where β is base, p is digits of precision, and e is exponent between e_{min} and e_{max}
- Normalize if $d_0 \neq 0$ (except for 0)
- Gaps between adjacent numbers scale with size of numbers
- Relative resolution given by *machine epsilon* $\epsilon_{\text{machine}} = 0.5\beta^{1-p}$
- For all x, there exists a floating point x' such that $|x - x'| \le \epsilon_{\text{machine}}|x|$

IEEE Floating Point Representations

- Single precision: 32 bits
  - 1 sign bit (S), 8 exponent bits (E), 23 significant bits (M), $(-1)^S \times 1.M \times 2^{E-127}$
  - $\epsilon_{\text{machine}}$ is $2^{-24} \approx 6e-8$
- Double precision: 64 bits
  - 1 sign bit (S), 11 exponent bits (E), 52 significant bits (M), $(-1)^S \times 1.M \times 2^{E-1023}$
  - $\epsilon_{\text{machine}}$ is $2^{-53} \approx e-16$
- Special quantities
  - $+\infty$ and $-\infty$ when operation overflows; e.g., $x/0$ for nonzero x
  - NaN (Not a Number) is returned when an operation has no well-defined result; e.g., $0/0$, $\sqrt{-1}$, $\arcsin(2)$, NaN

Machine Epsilon

- Define $\text{fl}(x)$ as closest floating point approximation to x
- By definition of $\epsilon_{\text{machine}}$, we have:

$$\text{For all } x \in \mathbb{R}, \text{ there exists } \epsilon \text{ with } |\epsilon| \leq \epsilon_{\text{machine}}$$
$$\text{such that } \text{fl}(x) = x(1+\epsilon)$$

- Given operation $+$, $-$, $\times$, and $/$ (denoted by $*$), floating point numbers x and y, and corresponding floating point arithmetic (denoted by $\circledast$), we require that $x \circledast y = \text{fl}(x * y)$
- This is guaranteed by IEEE floating point arithmetic
- Fundamental axiom of floating point arithmetic:

$$\text{For all } x, y \in \mathbb{F}, \text{ there exists } \epsilon \text{ with } |\epsilon| \leq \epsilon_{\text{machine}}$$
$$\text{such that } x \circledast y = (x * y)(1+\epsilon)$$

- These properties will be the basis of error analysis with rounding errors

Potential "Catastrophic" Error 1: Cancellation

Catastrophic cancellation: If $x \approx y$ and $|x - y| \ll |x| + |y|$, then the computed result $\hat{x} - \hat{y}$ is in general inaccurate.

- This is an issue if intermediate result involves cancellation errors
- This *sometimes* can be avoided by using alternative equations
- Example:
  - Quadratic formula for quadratic roots of $ax^2 + bx + c = 0$ can be solved as
    $$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$
    or
    $$x = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}}$$
    depending on the sign of b

Potential "Catastrophic" Error 2: Swamping

Swamping: If $|x| \ll |y|$ (more precisely $|x| \lesssim \epsilon_{\text{machine}}|y|$), then $x + y \approx y$ (or $fl(x) \oplus fl(y) = fl(y)$). In other words, the effect of smaller value may be lost

- This is an issue if there are intermediate large values in the computation
- It *sometimes* can be avoided by reordering computations
- Examples:
  - Approximating e with $\sum_{n=0}^{N} \frac{1}{n!}$ versus $\sum_{n=0}^{N} \frac{1}{(N-n)!}$
  - in Gaussian elimination, using small values as pivoting can lead to large intermediate values

Potential "Catastrophic" Error 3: Overflow

Overflow: When multiplying two very large numbers x and y, then $|xy|$ can cause unnecessary overflow

- This is an issue, for example, when computing squares before taking squared root, but it can often be addressed by rescaling
- Analogously, unnecessary underflow can occur when multiplying two small numbers, but underflow is typically less harmful
- Examples:
  - In finding roots of $ax^2 + bx + c = 0$, it is advisable to rescale the problem by dividing it by $\max\{|a|, |b|, |c|\}$ (this avoids unnecessary overflow and underflow)
  - When computing $\|x\|_2$, rescale the problem to compute $\|x\|_\infty \left\| \frac{x}{\|x\|_\infty} \right\|_2$ (i.e., first dividing x by $\max\{|x_i|\}$ and multiplying it back. This can avoid unnecessary overflow and underflow)

Outline

Accuracy

- Roughly speaking, accuracy means that "error" is small in an *asymptotic* sense, say $O(\epsilon_{\text{machine}})$
- Notation $\varphi(t) = O(\psi(t))$ means $\exists C$ s.t. $|\varphi(t)| \leq C|\psi(t)|$ as t approaches 0 (or ∞)
  - Example: $\sin^2 t = O(t^2)$ as $t \to 0$
- If φ depends on s and t, then $\varphi(s,t) = O(\psi(t))$ means $\exists C$ s.t. $|\varphi(s,t)| \leq C|\psi(t)|$ for any s as t approaches 0 (or ∞)
  - Example: $\sin^2 t \ \sin^2 s = O(t^2)$ as $t \to 0$
- When we say $O(\epsilon_{\text{machine}})$, we are thinking of a series of idealized machines for which $\epsilon_{\text{machine}} \to 0$

More on Accuracy

- An algorithm $\tilde{f}$ is *accurate* if **relative** error is in the order of machine precision, i.e.,

$$\|\tilde{f}(x) - f(x)\| / \|f(x)\| = O(\epsilon_{\text{machine}}),$$

  i.e., $\leq C_1 \epsilon_{\text{machine}}$ as $\epsilon_{\text{machine}} \to 0$, where constant C_1 may depend on the condition number and the algorithm itself
- In most cases, we expect

$$\|\tilde{f}(x) - f(x)\| / \|f(x)\| = O(\kappa \epsilon_{\text{machine}}),$$

  i.e., $\leq C \kappa \epsilon_{\text{machine}}$ as $\epsilon_{\text{machine}} \to 0$, where constant C should be independent of κ and value of x (although it may depends on the dimension of x)
- How do we determine whether an algorithm is accurate or not?
  - It turns out to be an extremely subtle question
  - A forward error analysis (operation by operation) is often too difficult and impractical, and cannot capture dependence on condition number
  - An effective solution is *backward error analysis*

Outline

Stability

- We say an algorithm is *stable* if it gives "nearly the right answer to nearly the right question"
- More formally, an algorithm $\tilde{f}$ for problem f is *stable* if (for all x)

$$\|\tilde{f}(x) - f(\tilde{x})\| / \|f(\tilde{x})\| = O(\epsilon_{\text{machine}})$$

  for some $\tilde{x}$ with $\|\tilde{x} - x\| / \|x\| = O(\epsilon_{\text{machine}})$

Stability

- We say an algorithm is *stable* if it gives "nearly the right answer to nearly the right question"
- More formally, an algorithm $\tilde{f}$ for problem f is *stable* if (for all x)

$$\|\tilde{f}(x) - f(\tilde{x})\| / \|f(\tilde{x})\| = O(\epsilon_{\text{machine}})$$

  for some $\tilde{x}$ with $\|\tilde{x} - x\| / \|x\| = O(\epsilon_{\text{machine}})$
- We say an algorithm is *backward stable* if it gives "exactly the right answer to nearly the right question"
- More formally, an algorithm $\tilde{f}$ for problem f is *backward stable* if (for all x)

$$\tilde{f}(x) = f(\tilde{x})$$

  for some $\tilde{x}$ with $\|\tilde{x} - x\| / \|x\| = O(\epsilon_{\text{machine}})$

Stability

- We say an algorithm is *stable* if it gives "nearly the right answer to nearly the right question"
- More formally, an algorithm $\tilde{f}$ for problem f is *stable* if (for all x)

$$\|\tilde{f}(x) - f(\tilde{x})\| / \|f(\tilde{x})\| = O(\epsilon_{\text{machine}})$$

  for some $\tilde{x}$ with $\|\tilde{x} - x\| / \|x\| = O(\epsilon_{\text{machine}})$
- We say an algorithm is *backward stable* if it gives "exactly the right answer to nearly the right question"
- More formally, an algorithm $\tilde{f}$ for problem f is *backward stable* if (for all x)

$$\tilde{f}(x) = f(\tilde{x})$$

  for some $\tilde{x}$ with $\|\tilde{x} - x\| / \|x\| = O(\epsilon_{\text{machine}})$
- Is stability or backward stability stronger?

Stability

- We say an algorithm is *stable* if it gives "nearly the right answer to nearly the right question"
- More formally, an algorithm $\tilde{f}$ for problem f is *stable* if (for all x)
$$\|\tilde{f}(x) - f(\tilde{x})\| / \|f(\tilde{x})\| = O(\epsilon_{\text{machine}})$$
for some $\tilde{x}$ with $\|\tilde{x} - x\| / \|x\| = O(\epsilon_{\text{machine}})$
- We say an algorithm is *backward stable* if it gives "exactly the right answer to nearly the right question"
- More formally, an algorithm $\tilde{f}$ for problem f is *backward stable* if (for all x)
$$\tilde{f}(x) = f(\tilde{x})$$
for some $\tilde{x}$ with $\|\tilde{x} - x\| / \|x\| = O(\epsilon_{\text{machine}})$
- Is stability or backward stability stronger?
  - Backward stability is stronger.
- Does (backward) stability depend on condition number of $f(x)$?

Stability

- We say an algorithm is *stable* if it gives "nearly the right answer to nearly the right question"
- More formally, an algorithm $\tilde{f}$ for problem f is *stable* if (for all x)

$$\|\tilde{f}(x) - f(\tilde{x})\| / \|f(\tilde{x})\| = O(\epsilon_{\text{machine}})$$

  for some $\tilde{x}$ with $\|\tilde{x} - x\| / \|x\| = O(\epsilon_{\text{machine}})$
- We say an algorithm is *backward stable* if it gives "exactly the right answer to nearly the right question"
- More formally, an algorithm $\tilde{f}$ for problem f is *backward stable* if (for all x)

$$\tilde{f}(x) = f(\tilde{x})$$

  for some $\tilde{x}$ with $\|\tilde{x} - x\| / \|x\| = O(\epsilon_{\text{machine}})$
- Is stability or backward stability stronger?
  - Backward stability is stronger.
- Does (backward) stability depend on condition number of $f(x)$?
  - No.

Stability of Floating Point Arithmetic

- Backward stability of floating point operations is implied by these two floating point axioms:
  1. $\forall x \in \mathbb{R}$, $\exists \epsilon$, $|\epsilon| \le \epsilon_{\text{machine}}$ s.t. $\text{fl}(x) = x(1+\epsilon)$
  2. For floating-point numbers x, y, $\exists \epsilon$, $|\epsilon| \le \epsilon_{\text{machine}}$ s.t. $x \circledast y = (x * y)(1+\epsilon)$
- Example: Subtraction $f(x_1, x_2) = x_1 - x_2$ with floating-point operation

$$\tilde{f}(x_1, x_2) = \text{fl}(x_1) \ominus \text{fl}(x_2)$$

  - Axiom 1 implies $\text{fl}(x_1) = x_1(1+\epsilon_1)$, $\text{fl}(x_2) = x_2(1+\epsilon_2)$, for some $|\epsilon_1|, |\epsilon_2| \le \epsilon_{\text{machine}}$
  - Axiom 2 implies $\text{fl}(x_1) \ominus \text{fl}(x_2) = (\text{fl}(x_1) - \text{fl}(x_2))(1+\epsilon_3)$ for some $|\epsilon_3| \le \epsilon_{\text{machine}}$
  - Therefore,

$$\begin{aligned} \text{fl}(x_1) \ominus \text{fl}(x_2) &= (x_1(1+\epsilon_1) - x_2(1+\epsilon_2))(1+\epsilon_3) \\ &= x_1(1+\epsilon_1)(1+\epsilon_3) - x_2(1+\epsilon_2)(1+\epsilon_3) \\ &= x_1(1+\epsilon_4) - x_2(1+\epsilon_5) \end{aligned}$$

  where $|\epsilon_4|, |\epsilon_5| \le 2\epsilon_{\text{machine}} + O(\epsilon_{\text{machine}}^2)$

Stability of Floating Point Arithmetic Cont'd

- Example: Inner product $f(x, y) = x^T y$ using floating-point operations $\otimes$ and $\oplus$ is backward stable
- Example: Outer product $f(x, y) = xy^T$ using $\otimes$ and $\oplus$ is not backward stable
- Example: $f(x) = x + 1$ computed as $\tilde{f}(x) = \mathrm{fl}(x) \oplus 1$ is not backward stable
- Example: $f(x, y) = x + y$ computed as $\tilde{f}(x, y) = \mathrm{fl}(x) \oplus \mathrm{fl}(y)$ is backward stable

Accuracy of Backward Stable Algorithm

Theorem

If a backward stable algorithm $\tilde{f}$ is used to solve a problem f with condition number κ using floating-point numbers satisfying the two axioms, then

$$\|\tilde{f}(x) - f(x)\| / \|f(x)\| = O(\kappa(x)\epsilon_{machine})$$

Accuracy of Backward Stable Algorithm

Theorem

If a backward stable algorithm $\tilde{f}$ is used to solve a problem f with condition number κ using floating-point numbers satisfying the two axioms, then

$$\|\tilde{f}(x) - f(x)\| / \|f(x)\| = O(\kappa(x)\epsilon_{machine})$$

Proof: Backward stability means $\tilde{f}(x) = f(\tilde{x})$ for $\tilde{x}$ such that $\|\tilde{x} - x\| / \|x\| = O(\epsilon_{\text{machine}})$
Definition of condition number gives

$$\|f(\tilde{x}) - f(x)\| / \|f(x)\| \leq (\kappa(x) + o(1)) \|\tilde{x} - x\| / \|x\|$$

where $o(1) \to 0$ as $\epsilon_{\text{machine}} \to 0$.
Combining the two gives desired result.