AMS526: Numerical Analysis I
(Numerical Linear Algebra)
Lecture 6: Floating Point Arithmetic; Accuracy and Stability; Triangular Systems

Xiangmin Jiao

Stony Brook University
Outline

1. Floating Point Arithmetic (NLA§13)

2. Accuracy and Stability (NLA§14-15)

3. Triangular Systems (MC§3.1)
Floating Point Representations

- Computers use finite number of bits to represent real numbers
  - 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
  \[ \pm (d_0 + d_1 \beta^{-1} + \cdots + d_{p-1} \beta^{-p+1}) \beta^e, \ 0 \leq d_i < \beta \]
  where \( \beta \) is base, \( p \) is digits of precision, and \( e \) is exponent between \( e_{\text{min}} \) and \( e_{\text{max}} \)

- 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 \( |x - x'| \leq \epsilon_{\text{machine}} |x| \)
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 \(2^{-24} \approx 6 \times 10^{-8}\)

- **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 \(2^{-53} \approx 1.11 \times 10^{-16}\)

- **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., \(0/0\), \(\sqrt{-1}\), \(\text{arcsin}(2)\), NaN
Floating Point Arithmetic

- Define $\text{fl}(x)$ as closest floating point approximation to $x$
- By definition of $\epsilon_{\text{machine}}$, we have:
  
  For all $x \in \mathbb{R}$, there exists $\epsilon$ with $|\epsilon| \leq \epsilon_{\text{machine}}$ such that $\text{fl}(x) = x(1 + \epsilon)$

- Given operation $+, -, \times, \text{ and } / \ (\text{denoted by } \ast)$, floating point numbers $x$ and $y$, and corresponding floating point arithmetic (denoted by $\ast$), we require that $x \ast y = \text{fl}(x \ast y)$
- This is guaranteed by IEEE floating point arithmetic
- Fundamental axiom of floating point arithmetic:
  
  For all $x, y \in \mathbb{F}$, there exists $\epsilon$ with $|\epsilon| \leq \epsilon_{\text{machine}}$ such that $x \ast y = (x \ast y)(1 + \epsilon)$

- These properties will be the basis of error analysis with rounding errors
- Note that floating point arithmetic is not associative
Outline

1. Floating Point Arithmetic (NLA§13)

2. Accuracy and Stability (NLA§14-15)

3. Triangular Systems (MC§3.1)
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 $\infty$).
  - Example: $\sin^2 t = O(t^2)$ as $t \to 0$.
- If $\varphi$ 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 $\infty$).
  - 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.,

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

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

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

i.e., $\leq C\kappa\epsilon_{\text{machine}}$ as $\epsilon_{\text{machine}} \to 0$, where constant $C$ should be independent of $\kappa$ and value of $x$ (although it may depend 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*.
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 $x$)
  \[
  \frac{\| \tilde{f}(x) - f(\tilde{x}) \|}{\| f(\tilde{x}) \|} = O(\epsilon_{\text{machine}})
  \]
  for some $\tilde{x}$ with $\| \tilde{x} - x \| / \| x \| = O(\epsilon_{\text{machine}})$

- Backward stability is stronger.
- Does (backward) stability depend on condition number of $f(x)$?
  \[ \text{No.} \]
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 \( x \))
  \[
  \frac{\| \tilde{f}(x) - f(\tilde{x}) \|}{\| f(\tilde{x}) \|} = O(\epsilon_{\text{machine}})
  \]
  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

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

Backward stability is stronger.
Stability

- We say an algorithm is \textit{stable} if it gives “nearly the right answer to nearly the right question”
- More formally, an algorithm $\tilde{f}$ for problem $f$ is \textit{stable} if (for all $x$)
  \[ \frac{\|\tilde{f}(x) - f(\tilde{x})\|}{\|f(\tilde{x})\|} = O(\epsilon_{\text{machine}}) \]
  for some $\tilde{x}$ with \[ \frac{\|\tilde{x} - x\|}{\|x\|} = O(\epsilon_{\text{machine}}) \]
- We say an algorithm is \textit{backward stable} if it gives “exactly the right answer to nearly the right question”
- More formally, an algorithm $\tilde{f}$ for problem $f$ is \textit{backward stable} if (for all $x$)
  \[ \tilde{f}(x) = f(\tilde{x}) \]
  for some $\tilde{x}$ with \[ \frac{\|\tilde{x} - x\|}{\|x\|} = O(\epsilon_{\text{machine}}) \]
- 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 $x$)
  \[
  \frac{\|\tilde{f}(x) - f(\tilde{x})\|}{\|f(\tilde{x})\|} = O(\epsilon_{\text{machine}})
  \]
  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. For all $x \in \mathbb{R}$, there exists $\epsilon$, $|\epsilon| \leq \epsilon_{\text{machine}}$ such that $\text{fl}(x) = x(1 + \epsilon)$
  2. For floating-point numbers $x$, $y$, there exists $\epsilon$, $|\epsilon| \leq \epsilon_{\text{machine}}$ such that $x \odot y = (x \ast y)(1 + \epsilon)$

- 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 $|\epsilon_1|, |\epsilon_2| \leq \epsilon_{\text{machine}}$
  - 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 $|\epsilon_3| \leq \epsilon_{\text{machine}}$
  - Therefore,
    \[
    \text{fl}(x_1) \ominus \text{fl}(x_2) = (x_1(1 + \epsilon_1) - x_2(1 + \epsilon_2))(1 + \epsilon_3) \\
    = x_1(1 + \epsilon_1)(1 + \epsilon_3) - x_2(1 + \epsilon_2)(1 + \epsilon_3) \\
    = x_1(1 + \epsilon_4) - x_2(1 + \epsilon_5)
    \]
    where $|\epsilon_4|, |\epsilon_5| \leq 2\epsilon_{\text{machine}} + O(\epsilon_{\text{machine}}^2)$
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 $\tilde{f}(x) = \text{fl}(x) \oplus 1$ is not backward stable

Example: $f(x, y) = x + y$ computed as $\tilde{f}(x, y) = \text{fl}(x) \oplus \text{fl}(y)$ 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 \( \kappa \) using floating-point numbers satisfying the two axioms, then

\[
\frac{\| \tilde{f}(x) - f(x) \|}{\| f(x) \|} = O(\kappa(x) \varepsilon_{\text{machine}})
\]

Proof: Backward stability means \( \tilde{f}(x) = f(\tilde{x}) \) for \( \tilde{x} \) such that

\[
\frac{\| \tilde{x} - x \|}{\| x \|} = O(\varepsilon_{\text{machine}})
\]

Definition of condition number gives

\[
\frac{\| f(\tilde{x}) - f(x) \|}{\| f(x) \|} \leq (\kappa(x) + o(1)) \frac{\| \tilde{x} - x \|}{\| x \|}
\]

where \( o(1) \to 0 \) as \( \varepsilon_{\text{machine}} \to 0 \).

Combining the two gives desired result.
Accuracy of Backward Stable Algorithm

Theorem

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

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

Proof: Backward stability means $\tilde{f}(x) = f(\tilde{x})$ for $\tilde{x}$ such that

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

Definition of condition number gives

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

where $o(1) \to 0$ as $\epsilon_{\text{machine}} \to 0$.

Combining the two gives desired result.
Outline

1. Floating Point Arithmetic (NLA§13)
2. Accuracy and Stability (NLA§14-15)
3. Triangular Systems (MC§3.1)
Triangular Systems

- A matrix $G = (g_{ij})$ is lower triangular if $g_{ij} = 0$ whenever $i < j$.

$$G = \begin{bmatrix}
g_{11} & 0 & 0 & \cdots & 0 \\
g_{21} & g_{22} & 0 & \cdots & 0 \\
g_{31} & g_{32} & g_{33} & \cdots & \vdots \\
\vdots & \vdots & \vdots & \ddots & 0 \\
g_{n1} & g_{n2} & g_{n3} & \cdots & g_{nn}
\end{bmatrix}$$

- Similarly, an upper triangular matrix is one for which $g_{ij} = 0$ whenever $i > j$.

- A triangular matrix is one that is either upper or lower triangular.

- A triangular matrix $G \in R^{n \times n}$ is nonsingular if and only if $g_{ii} \neq 0$ for $i = 1, \ldots, n$. 
Lower-Triangular Systems

- Consider system $Gy = b$, where $G$ is nonsingular, lower-triangular matrix.
- We can solve the system by

$$
\begin{align*}
y_1 &= b_1 / g_{11} \\
y_2 &= (b_2 - g_{21}y_1) / g_{22} \\
& \vdots \\
y_i &= (b_i - g_{i1}y_1 - g_{i2}y_2 - \cdots - g_{i,i-1}y_{i-1}) / g_{ii} \\
& = \left( b_i - \sum_{j=1}^{i-1} g_{ij}y_j \right) / g_{ii}.
\end{align*}
$$
Forward Substitution

Pseudo-code forward substitution ($y$ overwrites $b$)

```
for i = 1 : n
    for j = 1 : i - 1
        $b_i \leftarrow b_i - g_{ij} b_j$
    $b_i \leftarrow b_i / g_{ii}$
```

- This algorithm is *row-oriented*, as it access $G$ by rows.
- It may raise an exception if $g_{ii} = 0$
- The number of operations is
  \[
  \sum_{i=1}^{n} \sum_{j=1}^{i-1} 2 = 2 \sum_{i=1}^{n} (i - 1) = n(n - 1) \approx n^2
  \]
Column-Oriented Forward Substitution

- We can reorder loops and obtain a column-oriented algorithm

Pseudo-code forward substitution ($y$ overwrites $b$)

\[
\text{for } j = 1 : n \\
\quad b_j \leftarrow b_j / g_{jj} \\
\text{for } i = j + 1 : n \\
\quad b_i \leftarrow b_i - g_{ij} b_j
\]

- The number of operations is

\[
\sum_{i=1}^{n} \sum_{j=1}^{i-1} 2 = 2 \sum_{i=1}^{n} (i - 1) = n(n - 1) \approx n^2
\]

- In practice, column-oriented algorithm is faster if $G$ is stored in a column-oriented fashion

- Like matrix-matrix multiplication, performance can be improved using block-matrix operators
Exploiting Leading Zeros

- If $b_1 = b_2 = \cdots = b_{k-1} = 0$, how to change algorithm to reduce operations?
- In row-oriented algorithm: let $i = k, \ldots, n$ and $j = k, \ldots, i - 1$
- In column-oriented algorithm: let $j = k, \ldots, n$
- This saves flops, especially if $k$ is large
  - Example: compute inverse of a lower-triangular matrix
Upper-Triangular Systems

- Consider the system

\[ Uy = b, \]

where \( U \in \mathbb{R}^{n \times n} \) is nonsingular and upper triangular.

- We solve the system from bottom to top

\[
y_n = b_n / g_{nn}
\]

\[
y_{n-1} = (b_{n-1} - g_{n-1,n} y_n) / g_{n-1,n-1}
\]

\[ \vdots \]

\[
y_i = (b_i - g_{in} y_n - g_{i,n-1} y_{n-1} - \cdots - g_{i,i+1} y_{i+1}) / g_{ii}
\]

\[
= \left( b_i - \sum_{j=i+1}^{n} g_{ij} y_j \right) / g_{ii}.
\]

- This is back substitution, and it has the same cost as forward substitution.