AMS526: Numerical Analysis I
(Numerical Linear Algebra for Computational and Data Sciences)

Lecture 6: Accuracy and Stability;
Triangular Systems;
Backward Stability of Back Substitution

Xiangmin Jiao

Stony Brook University
Outline

1. Accuracy and Stability (NLA§14-15)

2. Triangular Systems (MC§3.1)

3. Backward Stability of Back Substitution (NLA§17)
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 $\exists C \text{ s.t. } |\varphi(t)| \leq C|\psi(t)|$ 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 $\exists C \text{ s.t. } |\varphi(s, t)| \leq C|\psi(t)|$ 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 $x$)
  \[
  \|\tilde{f}(x) - f(\tilde{x})\| / \|f(\tilde{x})\| = O(\epsilon_{\text{machine}})
  \]
  for some $\tilde{x}$ with $\|\tilde{x} - x\| / \|x\| = O(\epsilon_{\text{machine}})$

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

▶ Backward stability is stronger.

▶ 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 $f(x)$?
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. \( \forall x \in \mathbb{R}, \exists \epsilon, |\epsilon| \leq \epsilon_{\text{machine}} \) s.t. \( \text{fl}(x) = x(1 + \epsilon) \)
  2. For floating-point numbers \( x, y \), \( \exists \epsilon, |\epsilon| \leq \epsilon_{\text{machine}} \) s.t. 

     \( x \odot y = (x \ast y)(1 + \epsilon) \)

- 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 \( |\epsilon_4|, |\epsilon_5| \leq 2\epsilon_{\text{machine}} + O(\epsilon_{\text{machine}}^2) \)
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 $\tilde{f}(x) = fl(x) \oplus 1$ is not backward stable
- Example: $f(x, y) = x + y$ computed as $\tilde{f}(x, y) = fl(x) \oplus fl(y)$ 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

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

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

\[
\| \tilde{x} - x \| / \| x \| = O(\epsilon_{\text{machine}})
\]

Definition of condition number gives

\[
\| f(\tilde{x}) - f(x) \| / \| f(x) \| \leq (\kappa(x) + o(1)) \| \tilde{x} - x \| / \| x \| \to 0 \text{ as } \epsilon_{\text{machine}} \to 0.
\]

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

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

Definition of condition number gives

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

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

Combining the two gives desired result.
Outline

1. Accuracy and Stability (NLA§14-15)

2. Triangular Systems (MC§3.1)

3. Backward Stability of Back Substitution (NLA§17)
A matrix $G = (g_{ij})$ is lower triangular if $g_{ij} = 0$ whenever $i < j$.

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

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 $G \in \mathbb{R}^{n \times n}$ is nonsingular if and only if $g_{ii} \neq 0$ for $i = 1, \ldots, n$. 

Xiangmin Jiao
Numerical Analysis I
Lower-Triangular Systems

- Consider system
  \[ Gy = b, \]
  where \( G \) is nonsingular, lower-triangular matrix.
- We can solve the system by
  \[
  y_1 = \frac{b_1}{g_{11}} \\
  y_2 = \frac{(b_2 - g_{21}y_1)}{g_{22}} \\
  \vdots \\
  y_i = \frac{(b_i - g_{i1}y_1 - g_{i2}y_2 - \cdots - g_{i,i-1}y_{i-1})}{g_{ii}} \\
  = \left( b_i - \sum_{j=1}^{i-1} g_{ij}y_j \right) / g_{ii}.
  \]
Forward Substitution

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

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

- 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)
  
  ```
  for j = 1 : n
      bj ← bj / gjj
  for i = j + 1 : n
      bi ← bi - gij bj
  ```

- The number of operations is again ≈ \(n^2\)
- 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

  \[ y_n = b_n / g_{nn} \]
  \[ y_{n-1} = \left( b_{n-1} - g_{n-1,n}y_n \right) / g_{n-1,n-1} \]
  \[ \vdots \]
  \[ y_i = \left( b_i - g_{in}y_n - g_{i,n-1}y_{n-1} - \cdots - g_{i,i+1}y_{i+1} \right) / g_{ii} \]
  \[ = \left( b_i - \sum_{j=i+1}^{n} g_{ij}y_j \right) / g_{ii}. \]

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

1. Accuracy and Stability (NLA§14-15)

2. Triangular Systems (MC§3.1)

3. Backward Stability of Back Substitution (NLA§17)
Backward Stability of Back Substitution

- Solve $Rx = b$ using back substitution
  
  $\begin{bmatrix} r_{11} & \cdots & r_{1n} \\ \vdots & \ddots & \vdots \\ r_{nn} & & r_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$

- for $j = n$ downto 1
  
  $x_j = (b_j - \sum_{k=j+1}^{n} x_k r_{jk})/r_{jj}$

- Back substitute is backward stable
  
  $(R + \delta R)\tilde{x} = b, \quad \|\delta R\|/\|R\| = O(\epsilon_{\text{machine}})$

  Furthermore, each component of $\delta R$ satisfies
  
  $|\delta r_{ij}|/|r_{ij}| \leq n\epsilon_{\text{machine}} + O(\epsilon_{\text{machine}}^2)$

- We will show in full detail for $n = 1, 2, 3$ as well as general $n$
Proof of Backward Stability \((n = 1)\)

- For \(n = 1\), the algorithm is simply one floating point division. Using floating-point axiom, we get

\[
\tilde{x}_1 = b_1 \odot r_{11} = \frac{b_1}{r_{11}} (1 + \epsilon_1) = \frac{b_1}{r_{11}(1 + \epsilon'_1)}
\]

where \(|\epsilon_1| \leq \epsilon_{\text{machine}}\) and \(|\epsilon'_1| \leq \epsilon_{\text{machine}} + O(\epsilon_{\text{machine}}^2)\)

- Therefore, we solved a perturbed problem exactly:

\[
(r_{11} + \delta r_{11})\tilde{x}_1 = b_1 \quad \text{with} \quad \frac{|\delta r_{11}|}{|r_{11}|} \leq \epsilon_{\text{machine}} + O(\epsilon_{\text{machine}}^2)
\]
Proof of Backward Stability \((n = 2)\)

- For \(n = 2\), we first solve for \(\tilde{x}_2\) as before. Then, we compute \(\tilde{x}_1\):

\[
\tilde{x}_1 = (b_1 \ominus (\tilde{x}_2 \otimes r_{12})) \otimes r_{11} = \frac{(b_1 - \tilde{x}_2 r_{12}(1 + \epsilon_2))(1 + \epsilon_3)}{r_{11}} (1 + \epsilon_4)
\]

\[
= \frac{b_1 - \tilde{x}_2 r_{12}(1 + \epsilon_2)}{r_{11}(1 + \epsilon'_3)(1 + \epsilon'_4)} = \frac{b_1 - \tilde{x}_2 r_{12}(1 + \epsilon_2)}{r_{11}(1 + 2\epsilon_5)}
\]

where \(|\epsilon_2|, |\epsilon_3|, |\epsilon_4| \leq \epsilon_{\text{machine}}\) and \(|\epsilon'_3|, |\epsilon'_4|, |\epsilon_5| \leq \epsilon_{\text{machine}} + O(\epsilon^2_{\text{machine}})\)

- Again, this is an exact solution to \((R + \delta R)\tilde{x} = b\) with

\[
\begin{bmatrix}
|\delta r_{11}|/|r_{11}| & |\delta r_{12}|/|r_{12}| \\
|\delta r_{21}|/|r_{21}| & |\delta r_{22}|/|r_{22}|
\end{bmatrix} = \begin{bmatrix}
2|\epsilon_5| & |\epsilon_2| \\
|\epsilon_1|
\end{bmatrix} \leq \begin{bmatrix}
2 & 1 \\
1 & 1
\end{bmatrix} \epsilon_{\text{machine}} + O(\epsilon^2_{\text{machine}})
\]
Proof for B-S of B-S ($n = 3$)

For $n = 3$, we solve for $\tilde{x}_3$ and $\tilde{x}_2$ as before. Then, we compute $\tilde{x}_1$:

$$
\tilde{x}_1 = [b_1 \ominus (\tilde{x}_2 \otimes r_{12}) \ominus (\tilde{x}_3 \otimes r_{13})] \otimes r_{11}
$$

$$
= \frac{[(b_1 - \tilde{x}_2 r_{12}(1 + \epsilon_4))(1 + \epsilon_6) - \tilde{x}_3 r_{13}(1 + \epsilon_5)](1 + \epsilon_7)}{r_{11}(1 + \epsilon'_8)}
$$

$$
= \frac{b_1 - \tilde{x}_2 r_{12}(1 + \epsilon_4) - \tilde{x}_3 r_{13}(1 + \epsilon_5)(1 + \epsilon'_6)}{r_{11}(1 + \epsilon'_6)(1 + \epsilon'_7)(1 + \epsilon'_8)}
$$

That is, $(R + \Delta R)\tilde{x} = b)$ with

$$
\begin{align*}
\left| \begin{array}{c|c|c}
\delta r_{11} & \delta r_{12} & \delta r_{13} \\
\hline
r_{11} & r_{12} & r_{13} \\
\delta r_{22} & \delta r_{23} & \delta r_{33} \\
r_{22} & r_{23} & r_{33}
\end{array} \right| & \leq \left| \begin{array}{ccc}
3 & 1 & 2 \\
2 & 1 & 1
\end{array} \right| \epsilon_{\text{machine}} + O(\epsilon^2_{\text{machine}})
\end{align*}
$$
Proof for B-S of B-S (general $n$)

- Similar analysis for general $n$ gives pattern

$$\frac{|\delta R|}{|R|} \leq W\epsilon + O(\epsilon^2_{\text{machine}})$$

where $W$ is (for $n = 5$)

$$\begin{bmatrix}
0 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 & 0
\end{bmatrix} \times \begin{bmatrix}
1 & 1 \\
1 & 1 \\
1 & 1 \\
0 & 0
\end{bmatrix} \oplus \begin{bmatrix}
4 & 0 & 1 & 2 & 3 \\
3 & 0 & 1 & 2 \\
2 & 0 & 1 \\
1 & 1 & 0 & 0
\end{bmatrix}$$

or

$$\begin{bmatrix}
5 & 1 & 2 & 3 & 4 \\
4 & 1 & 2 & 3 \\
3 & 1 & 2 \\
2 & 1 & 1 \\
1 & 0 & 0
\end{bmatrix}$$