KUTAISI INTERNATIONAL UNIVERSITY

Numerical Linear Algebra

Ramaz Botchorishvili

Kutaisi International University

November 1, 2023

KUTAISI
INTERNATIONAL
UNIVERSITY

Numerical Linear Algebra

Ramaz Botchorishvili

Kutaisi International University

November 1, 2023

Well posed problem, ill conditioned problem, condition Number

- Recap of Previous Lecture
- Round-of errors
- How to avoid cancellation and recursion errors
- Computational template for numerical linear algebra
- Thomas algorithm
- Q & A

Recap of Previous Lecture

- Perturbations in right hand side and coefficients
- Error sources
- Number systems
- Floating point

Round-off errors, 1

Rounding and Chopping

Round-off errors, 1

Rounding and Chopping

- Not all real numbers are machine representable

Round-off errors, 1

Rounding and Chopping

- Not all real numbers are machine representable
- Finite number of real numbers are only representable in computers

Round-off errors, 1

Rounding and Chopping

- Not all real numbers are machine representable
- Finite number of real numbers are only representable in computers
- Consider $0.d_{-1}...d_{-m}d_{-m-1}$

Round-off errors, 1

Rounding and Chopping

- Not all real numbers are machine representable
- Finite number of real numbers are only representable in computers
- Consider $0.d_{-1}...d_{-m}d_{-m-1}$
- **Chopping**:

  in m-digit arithmetics d_{-m-1} and all other further digits are thrown away

Round-off errors, 1

Rounding and Chopping

- Not all real numbers are machine representable
- Finite number of real numbers are only representable in computers
- Consider $0.d_{-1}...d_{-m}d_{-m-1}$
- **Chopping**:

  in m-digit arithmetics d_{-m-1} and all other further digits are thrown away
- **Rounding**:

  in m-digit arithmetics d_{-m} is rounded up or down, d_{-m-1} and all other further digits are thrown away

Round-off errors, 1

Rounding and Chopping

- Not all real numbers are machine representable
- Finite number of real numbers are only representable in computers
- Consider $0.d_{-1}...d_{-m}d_{-m-1}$
- **Chopping**:
  in m-digit arithmetics d_{-m-1} and all other further digits are thrown away
- **Rounding**:
  in m-digit arithmetics d_{-m} is rounded up or down, d_{-m-1} and all other further digits are thrown away
- Rounding down: $d_{-m-1} < \beta/2$

Round-off errors, 1

Rounding and Chopping

- Not all real numbers are machine representable
- Finite number of real numbers are only representable in computers
- Consider $0.d_{-1}...d_{-m}d_{-m-1}$
- **Chopping**:
  in m-digit arithmetics d_{-m-1} and all other further digits are thrown away
- **Rounding**:
  in m-digit arithmetics d_{-m} is rounded up or down, d_{-m-1} and all other further digits are thrown away
- Rounding down: $d_{-m-1} < \beta/2$
- Rounding up: $d_{-m-1} \geq \beta/2$

Example 7.1

$\pi = 3.1415926$

Round-off errors, 1

Rounding and Chopping

- Not all real numbers are machine representable
- Finite number of real numbers are only representable in computers
- Consider $0.d_{-1}...d_{-m}d_{-m-1}$
- **Chopping**:
  in m-digit arithmetics d_{-m-1} and all other further digits are thrown away
- **Rounding**:
  in m-digit arithmetics d_{-m} is rounded up or down, d_{-m-1} and all other further digits are thrown away
- Rounding down: $d_{-m-1} < \beta/2$
- Rounding up: $d_{-m-1} \geq \beta/2$

Example 7.1

$\pi = 3.1415926$

- Two-digit arithmetic $fl(\pi) = 0.31 \cdot 10^{-2}$

Round-off errors, 1

Rounding and Chopping

- Not all real numbers are machine representable
- Finite number of real numbers are only representable in computers
- Consider $0.d_{-1}...d_{-m}d_{-m-1}$
- **Chopping**:
  in m-digit arithmetics d_{-m-1} and all other further digits are thrown away
- **Rounding**:
  in m-digit arithmetics d_{-m} is rounded up or down, d_{-m-1} and all other further digits are thrown away
- Rounding down: $d_{-m-1} < \beta/2$
- Rounding up: $d_{-m-1} \geq \beta/2$

Example 7.1

$\pi = 3.1415926$

- Two-digit arithmetic $fl(\pi) = 0.31 \cdot 10^{-2}$
- Three-digit arithmetic $fl(\pi) = 0.314 \cdot 10^{-2}$

Round-off errors, 1

Rounding and Chopping

- Not all real numbers are machine representable
- Finite number of real numbers are only representable in computers
- Consider $0.d_{-1}...d_{-m}d_{-m-1}$
- **Chopping**:
  in m-digit arithmetics d_{-m-1} and all other further digits are thrown away
- **Rounding**:
  in m-digit arithmetics d_{-m} is rounded up or down, d_{-m-1} and all other further digits are thrown away
- Rounding down: $d_{-m-1} < \beta/2$
- Rounding up: $d_{-m-1} \geq \beta/2$

Example 7.1

$\pi = 3.1415926$

- Two-digit arithmetic $fl(\pi) = 0.31 \cdot 10^{-2}$
- Three-digit arithmetic $fl(\pi) = 0.314 \cdot 10^{-2}$
- Four-digit arithmetic $fl(\pi) = 0.3142 \cdot 10^{-2}$

Round-off errors, 2

Machine precision, significant numbers

Definition 7.2

Machine precision μ is **smallest positive number** such that

$$\mathit{fl}(1+\mu) > 1$$

Round-off errors, 2

Machine precision, significant numbers

Definition 7.2

Machine precision μ is **smallest positive number** such that

$$fl(1+\mu) > 1$$

Definition 7.3

Suppose x is real number and $\tilde{x}$ is its approximation. $\tilde{x}$ approximates x to s significant digits if s is **largest nonnegative integer** for which relative error satisfies the inequality:

$$\frac{|x-\tilde{x}|}{|x|} < 5 \cdot 10^{-s}$$

Round-off errors, 2

Machine precision, significant numbers

Definition 7.2

Machine precision μ is **smallest positive number** such that

$$fl(1+\mu) > 1$$

Definition 7.3

Suppose x is real number and $\tilde{x}$ is its approximation. $\tilde{x}$ approximates x to s significant digits if s is **largest nonnegative integer** for which relative error satisfies the inequality:

$$\frac{|x-\tilde{x}|}{|x|} < 5 \cdot 10^{-s}$$

Example 7.4

- $x = 1.31, \tilde{x} = 1.3, |x-\tilde{x}| = 0.01, \frac{|x-\tilde{x}|}{|x|} = 0.007635$
- $7.635 \cdot 10^{-3} < 5 \cdot 10^{-2}$, agree up to two significant digits

Round-off errors, 3

- Round-off Error in Representation of a Real Number

Theorem 7.5

$$\frac{|x - fl(x)|}{|x|} \leq \mu = \begin{cases} 0.5\beta^{1-m} & \textit{for rounding} \\ \beta^{1-m} & \textit{for chopping} \end{cases}$$

Round-off errors, 3

- Round-off Error in Representation of a Real Number

Theorem 7.5

$$\frac{|x - fl(x)|}{|x|} \leq \mu = \begin{cases} 0.5\beta^{1-m} & \textit{for rounding} \\ \beta^{1-m} & \textit{for chopping} \end{cases}$$

- Proof for round-off case

Proof.

- Consider $x = (0.d_{-1}...d_{-m}d_{-m-1}..)\beta^e$, $d_{-1} \neq 0, 0 \leq d_{-i} < \beta$

Round-off errors, 3

- Round-off Error in Representation of a Real Number

Theorem 7.5

$$\frac{|x-fl(x)|}{|x|} \le \mu = \begin{cases} 0.5\beta^{1-m} & \textit{for rounding} \\ \beta^{1-m} & \textit{for chopping} \end{cases}$$

- Proof for round-off case

Proof.

- Consider $x = (0.d_{-1}...d_{-m}d_{-m-1}..)\beta^e$, $d_{-1} \neq 0, 0 \le d_{-i} < \beta$
- $fl(x)_{\text{rounded down}} = (0.d_{-1}...d_{-m})\beta^e$

Round-off errors, 3

- Round-off Error in Representation of a Real Number

Theorem 7.5

$$\frac{|x-fl(x)|}{|x|} \leq \mu = \begin{cases} 0.5\beta^{1-m} & \textit{for rounding} \\ \beta^{1-m} & \textit{for chopping} \end{cases}$$

- Proof for round-off case

Proof.

- Consider $x = (0.d_{-1}...d_{-m}d_{-m-1}..)\beta^e$, $d_{-1} \neq 0, 0 \leq d_{-i} < \beta$
- $fl(x)_{\text{rounded down}} = (0.d_{-1}...d_{-m})\beta^e$
- $fl(x)_{\text{rounded up}} = (0.d_{-1}...d_{-m} + \beta^{-m})\beta^e$

Round-off errors, 3

- Round-off Error in Representation of a Real Number

Theorem 7.5

$$\frac{|x - fl(x)|}{|x|} \leq \mu = \begin{cases} 0.5\beta^{1-m} & \textit{for rounding} \\ \beta^{1-m} & \textit{for chopping} \end{cases}$$

- Proof for round-off case

Proof.

- Consider $x = (0.d_{-1}...d_{-m}d_{-m-1}..)\beta^e$, $d_{-1} \neq 0, 0 \leq d_{-i} < \beta$
- $fl(x)_{\text{rounded down}} = (0.d_{-1}...d_{-m})\beta^e$
- $fl(x)_{\text{rounded up}} = (0.d_{-1}...d_{-m} + \beta^{-m})\beta^e$
- $x \in [fl(x)_{\text{rounded down}}, fl(x)_{\text{rounded up}}]$

Round-off errors, 3

- Round-off Error in Representation of a Real Number

Theorem 7.5

$$\frac{|x - fl(x)|}{|x|} \le \mu = \begin{cases} 0.5\beta^{1-m} & \textit{for rounding} \\ \beta^{1-m} & \textit{for chopping} \end{cases}$$

- Proof for round-off case

Proof.

- Consider $x = (0.d_{-1}...d_{-m}d_{-m-1}..)\beta^e$, $d_{-1} \neq 0, 0 \le d_{-i} < \beta$
- $fl(x)_{\text{rounded down}} = (0.d_{-1}...d_{-m})\beta^e$
- $fl(x)_{\text{rounded up}} = (0.d_{-1}...d_{-m} + \beta^{-m})\beta^e$
- $x \in [fl(x)_{\text{rounded down}}, fl(x)_{\text{rounded up}}]$
- $|x - fl(x)_{\text{rounded down}}| \le 0.5|fl(x)_{\text{rounded down}} - fl(x)_{\text{rounded up}}| = \beta^{e-m}$

Round-off errors, 3

- Round-off Error in Representation of a Real Number

Theorem 7.5

$$\frac{|x-fl(x)|}{|x|} \leq \mu = \begin{cases} 0.5\beta^{1-m} & \textit{for rounding} \\ \beta^{1-m} & \textit{for chopping} \end{cases}$$

- Proof for round-off case

Proof.

- Consider $x = (0.d_{-1}...d_{-m}d_{-m-1}..)\beta^e$, $d_{-1} \neq 0, 0 \leq d_{-i} < \beta$
- $fl(x)_{\text{rounded down}} = (0.d_{-1}...d_{-m})\beta^e$
- $fl(x)_{\text{rounded up}} = (0.d_{-1}...d_{-m} + \beta^{-m})\beta^e$
- $x \in [fl(x)_{\text{rounded down}}, fl(x)_{\text{rounded up}}]$
- $|x - fl(x)_{\text{rounded down}}| \leq 0.5|fl(x)_{\text{rounded down}} - fl(x)_{\text{rounded up}}| = \beta^{e-m}$
- $|x - fl(x)_{\text{rounded up}}| \leq 0.5|fl(x)_{\text{rounded down}} - fl(x)_{\text{rounded up}}| = \beta^{e-m}$

Round-off errors, 3

- Round-off Error in Representation of a Real Number

Theorem 7.5

$$\frac{|x-fl(x)|}{|x|} \leq \mu = \begin{cases} 0.5\beta^{1-m} & \textit{for rounding} \\ \beta^{1-m} & \textit{for chopping} \end{cases}$$

- Proof for round-off case

Proof.

- Consider $x = (0.d_{-1}...d_{-m}d_{-m-1}..)\beta^e$, $d_{-1} \neq 0, 0 \leq d_{-i} < \beta$
- $fl(x)_{\text{rounded down}} = (0.d_{-1}...d_{-m})\beta^e$
- $fl(x)_{\text{rounded up}} = (0.d_{-1}...d_{-m} + \beta^{-m})\beta^e$
- $x \in [fl(x)_{\text{rounded down}}, fl(x)_{\text{rounded up}}]$
- $|x - fl(x)_{\text{rounded down}}| \leq 0.5|fl(x)_{\text{rounded down}} - fl(x)_{\text{rounded up}}| = \beta^{e-m}$
- $|x - fl(x)_{\text{rounded up}}| \leq 0.5|fl(x)_{\text{rounded down}} - fl(x)_{\text{rounded up}}| = \beta^{e-m}$
- $\frac{|x-fl(x)|}{|x|} \leq \frac{0.5\beta^{-m}}{0.d_{-1}...d_{-m}d_{-m-1}..} \leq \frac{0.5\beta^{-m}}{\beta^{-1}} = 0.5\beta^{1-m}$

Round-off errors, 4

- Round-off Error in Representation of a Real Number

Corollary 7.6

$$\frac{|x - fl(x)|}{|x|} \le \mu = \begin{cases} 0.5\beta^{1-m} & \textit{for rounding} \\ \beta^{1-m} & \textit{for chopping} \end{cases} \Rightarrow fl(x) = x(1+\delta), |\delta| \le \mu$$

Round-off errors, 4

- Round-off Error in Representation of a Real Number

Corollary 7.6

$$\frac{|x - fl(x)|}{|x|} \leq \mu = \begin{cases} 0.5\beta^{1-m} & \textit{for rounding} \\ \beta^{1-m} & \textit{for chopping} \end{cases} \Rightarrow fl(x) = x(1+\delta), |\delta| \leq \mu$$

Proof.

- We set: $\delta \equiv \frac{x - fl(x)}{x}$

Round-off errors, 4

- Round-off Error in Representation of a Real Number

Corollary 7.6

$$\frac{|x-fl(x)|}{|x|} \leq \mu = \begin{cases} 0.5\beta^{1-m} & \textit{for rounding} \\ \beta^{1-m} & \textit{for chopping} \end{cases} \Rightarrow fl(x) = x(1+\delta), |\delta| \leq \mu$$

Proof.

- We set: $\delta \equiv \frac{x-fl(x)}{x}$
- $\Rightarrow |\delta| \leq \mu$

Round-off errors, 4

- Round-off Error in Representation of a Real Number

Corollary 7.6

$$\frac{|x - fl(x)|}{|x|} \leq \mu = \begin{cases} 0.5\beta^{1-m} & \textit{for rounding} \\ \beta^{1-m} & \textit{for chopping} \end{cases} \Rightarrow fl(x) = x(1+\delta), |\delta| \leq \mu$$

Proof.

- We set: $\delta \equiv \frac{x - fl(x)}{x}$
- $\Rightarrow |\delta| \leq \mu$
- $\Rightarrow fl(x) - x = x\delta$

Round-off errors, 4

- Round-off Error in Representation of a Real Number

Corollary 7.6

$$\frac{|x - fl(x)|}{|x|} \leq \mu = \begin{cases} 0.5\beta^{1-m} & \textit{for rounding} \\ \beta^{1-m} & \textit{for chopping} \end{cases} \Rightarrow fl(x) = x(1+\delta), |\delta| \leq \mu$$

Proof.

- We set: $\delta \equiv \frac{x - fl(x)}{x}$
- $\Rightarrow |\delta| \leq \mu$
- $\Rightarrow fl(x) - x = x\delta$
- $\Rightarrow fl(x) = x(1+\delta)$

$\square$

Round-off errors, 5

- Round-off in floating point addition

Example 7.7

- Floating point system $\beta = 10, m = 2, e_{min} = -3, e_{max} = 3$

Round-off errors, 5

- Round-off in floating point addition

Example 7.7

- Floating point system $\beta = 10, m = 2, e_{min} = -3, e_{max} = 3$
- $x_1 = 0.99 \cdot 10^1, x_2 = 0.11 \cdot 10^0$

Round-off errors, 5

- Round-off in floating point addition

Example 7.7

- Floating point system $\beta = 10, m = 2, e_{min} = -3, e_{max} = 3$
- $x_1 = 0.99 \cdot 10^1, x_2 = 0.11 \cdot 10^0$
- $x_1 + x_2 = 9.9 + 0.11 = 10.01 = 0.1001 \cdot 10^2$

Round-off errors, 5

- Round-off in floating point addition

Example 7.7

- Floating point system $\beta = 10, m = 2, e_{min} = -3, e_{max} = 3$
- $x_1 = 0.99 \cdot 10^1, x_2 = 0.11 \cdot 10^0$
- $x_1 + x_2 = 9.9 + 0.11 = 10.01 = 0.1001 \cdot 10^2$
- $fl(x_1 + x_2) = 0.10 \cdot 10^2$

Round-off errors, 5

- Round-off in floating point addition

Example 7.7

- Floating point system $\beta = 10, m = 2, e_{min} = -3, e_{max} = 3$
- $x_1 = 0.99 \cdot 10^1, x_2 = 0.11 \cdot 10^0$
- $x_1 + x_2 = 9.9 + 0.11 = 10.01 = 0.1001 \cdot 10^2$
- $fl(x_1 + x_2) = 0.10 \cdot 10^2$
- $(x_1 + x_2) - fl(x_1 + x_2) = 0.0001 \cdot 10^2 = 0.1 \cdot 10^{-1}$

Round-off errors, 5

- Round-off in floating point addition

Example 7.7

- Floating point system $\beta = 10, m = 2, e_{min} = -3, e_{max} = 3$
- $x_1 = 0.99 \cdot 10^1, x_2 = 0.11 \cdot 10^0$
- $x_1 + x_2 = 9.9 + 0.11 = 10.01 = 0.1001 \cdot 10^2$
- $fl(x_1 + x_2) = 0.10 \cdot 10^2$
- $(x_1 + x_2) - fl(x_1 + x_2) = 0.0001 \cdot 10^2 = 0.1 \cdot 10^{-1}$
- $\delta = \frac{(x_1+x_2)-fl(x_1+x_2)}{x_1+x_2} \approx 0.999 \cdot 10^{-3}$

Round-off errors, 5

- Round-off in floating point addition

Example 7.7

- Floating point system $\beta = 10, m = 2, e_{min} = -3, e_{max} = 3$
- $x_1 = 0.99 \cdot 10^1, x_2 = 0.11 \cdot 10^0$
- $x_1 + x_2 = 9.9 + 0.11 = 10.01 = 0.1001 \cdot 10^2$
- $fl(x_1 + x_2) = 0.10 \cdot 10^2$
- $(x_1 + x_2) - fl(x_1 + x_2) = 0.0001 \cdot 10^2 = 0.1 \cdot 10^{-1}$
- $\delta = \frac{(x_1+x_2)-fl(x_1+x_2)}{x_1+x_2} \approx 0.999 \cdot 10^{-3}$
- $\delta =\approx 0.999000 \cdot 10^{-3} < 0.5 \cdot 10^{-2}$

- Other arithmetic operations: $\times, :, -$

Round-off errors, 5

- Round-off in floating point addition

Example 7.7

- Floating point system $\beta = 10, m = 2, e_{min} = -3, e_{max} = 3$
- $x_1 = 0.99 \cdot 10^1, x_2 = 0.11 \cdot 10^0$
- $x_1 + x_2 = 9.9 + 0.11 = 10.01 = 0.1001 \cdot 10^2$
- $fl(x_1 + x_2) = 0.10 \cdot 10^2$
- $(x_1 + x_2) - fl(x_1 + x_2) = 0.0001 \cdot 10^2 = 0.1 \cdot 10^{-1}$
- $\delta = \frac{(x_1+x_2)-fl(x_1+x_2)}{x_1+x_2} \approx 0.999 \cdot 10^{-3}$
- $\delta = \approx 0.999000 \cdot 10^{-3} < 0.5 \cdot 10^{-2}$

- Other arithmetic operations: $\times, :, -$
- See other examples in the textbook p.34

Round-off errors, 6

- **Guard digits** are extra digits in arithmetic register

Round-off errors, 6

- **Guard digits** are extra digits in arithmetic register
- Can be used for reducing round-off errors

Round-off errors, 6

- **Guard digits** are extra digits in arithmetic register
- Can be used for reducing round-off errors
- In multi-step calculations intermediate results are not rounded

Round-off errors, 6

- **Guard digits** are extra digits in arithmetic register
- Can be used for reducing round-off errors
- In multi-step calculations intermediate results are not rounded
- For computers with guard digits $fl(x_1 + x_2) = (x_1 + x_2)(1 + \delta), |\delta| \leq \mu$

Round-off errors, 6

- **Guard digits** are extra digits in arithmetic register
- Can be used for reducing round-off errors
- In multi-step calculations intermediate results are not rounded
- For computers with guard digits $fl(x_1 + x_2) = (x_1 + x_2)(1 + \delta), |\delta| \leq \mu$
- For computers without guard digits
  $fl(x_1 + x_2) = x_1(1 + \delta_1) + x_2(1 + \delta_2), |\delta_1| \leq \mu, |\delta_2| \leq \mu$

Round-off errors, 6

- **Guard digits** are extra digits in arithmetic register
- Can be used for reducing round-off errors
- In multi-step calculations intermediate results are not rounded
- For computers with guard digits $fl(x_1 + x_2) = (x_1 + x_2)(1 + \delta), |\delta| \le \mu$
- For computers without guard digits
  $fl(x_1 + x_2) = x_1(1 + \delta_1) + x_2(1 + \delta_2), |\delta_1| \le \mu, |\delta_2| \le \mu$
- IEEE standard:

Theorem 7.8

Laws of floating point arithmetic:

- $fl(x_1 + x_2) = (x_1 + x_2)(1 + \delta), |\delta| \le \mu$

Round-off errors, 6

- **Guard digits** are extra digits in arithmetic register
- Can be used for reducing round-off errors
- In multi-step calculations intermediate results are not rounded
- For computers with guard digits $fl(x_1 + x_2) = (x_1 + x_2)(1 + \delta), |\delta| \leq \mu$
- For computers without guard digits
  $fl(x_1 + x_2) = x_1(1 + \delta_1) + x_2(1 + \delta_2), |\delta_1| \leq \mu, |\delta_2| \leq \mu$
- IEEE standard:

Theorem 7.8

Laws of floating point arithmetic:

- $fl(x_1 + x_2) = (x_1 + x_2)(1 + \delta), |\delta| \leq \mu$
- $fl(x_1 \times x_2) = (x_1 \times x_2)(1 + \delta), |\delta| \leq \mu$

Round-off errors, 6

- **Guard digits** are extra digits in arithmetic register
- Can be used for reducing round-off errors
- In multi-step calculations intermediate results are not rounded
- For computers with guard digits $fl(x_1 + x_2) = (x_1 + x_2)(1 + \delta), |\delta| \leq \mu$
- For computers without guard digits
  $fl(x_1 + x_2) = x_1(1 + \delta_1) + x_2(1 + \delta_2), |\delta_1| \leq \mu, |\delta_2| \leq \mu$
- IEEE standard:

Theorem 7.8

Laws of floating point arithmetic:

- $fl(x_1 + x_2) = (x_1 + x_2)(1 + \delta), |\delta| \leq \mu$
- $fl(x_1 \times x_2) = (x_1 \times x_2)(1 + \delta), |\delta| \leq \mu$
- $fl(x_1 / x_2) = (x_1 / x_2)(1 + \delta), |\delta| \leq \mu$

Round-off errors, 6

- **Guard digits** are extra digits in arithmetic register
- Can be used for reducing round-off errors
- In multi-step calculations intermediate results are not rounded
- For computers with guard digits $fl(x_1 + x_2) = (x_1 + x_2)(1 + \delta), |\delta| \leq \mu$
- For computers without guard digits
  $fl(x_1 + x_2) = x_1(1 + \delta_1) + x_2(1 + \delta_2), |\delta_1| \leq \mu, |\delta_2| \leq \mu$
- IEEE standard:

Theorem 7.8

Laws of floating point arithmetic:

- $fl(x_1 + x_2) = (x_1 + x_2)(1 + \delta), |\delta| \leq \mu$
- $fl(x_1 \times x_2) = (x_1 \times x_2)(1 + \delta), |\delta| \leq \mu$
- $fl(x_1 / x_2) = (x_1 / x_2)(1 + \delta), |\delta| \leq \mu$

- What if *n* operands are involved in computation?

Round-off errors, 7

Theorem 7.9

Round-off error in floating point addition

Round-off errors, 7

Theorem 7.9

Round-off error in floating point addition

- $x_i \in \mathbb{R}, 2 \leq i \leq n$

Round-off errors, 7

Theorem 7.9

Round-off error in floating point addition

- $x_i \in \mathbb{R}, 2 \leq i \leq n$
- $$fl(x_1 + x_2 + ... + x_n) - (x_1 + x_2 + ... + x_n) \approx$$
  $$(x_1 + x_2)(\delta_2 + \delta_3 + ... + \delta_n) + x_3(\delta_3 + \delta_4 + ... + \delta_n) + ... + x_n\delta_n$$

Round-off errors, 7

Theorem 7.9

Round-off error in floating point addition

- $x_i \in \mathbb{R}, 2 \leq i \leq n$
- $$fl(x_1 + x_2 + ... + x_n) - (x_1 + x_2 + ... + x_n) \approx$$
  $$(x_1 + x_2)(\delta_2 + \delta_3 + ... + \delta_n) + x_3(\delta_3 + \delta_4 + ... + \delta_n) + ... + x_n\delta_n$$
- $|\delta_i| \leq \mu, 2 \leq i \leq n$

Round-off errors, 7

Theorem 7.9

Round-off error in floating point addition

- $x_i \in \mathbb{R}, 2 \leq i \leq n$
- $$fl(x_1 + x_2 + ... + x_n) - (x_1 + x_2 + ... + x_n) \approx$$
  $$(x_1 + x_2)(\delta_2 + \delta_3 + ... + \delta_n) + x_3(\delta_3 + \delta_4 + ... + \delta_n) + ... + x_n\delta_n$$
- $|\delta_i| \leq \mu, 2 \leq i \leq n$

Proof.

- $s = x_1 + x_2 + ... + x_n$

Round-off errors, 7

Theorem 7.9

Round-off error in floating point addition

- $x_i \in \mathbb{R}, 2 \le i \le n$
- $$fl(x_1 + x_2 + \ldots + x_n) - (x_1 + x_2 + \ldots + x_n) \approx$$
  $$(x_1 + x_2)(\delta_2 + \delta_3 + \ldots + \delta_n) + x_3(\delta_3 + \delta_4 + \ldots + \delta_n) + \ldots + x_n\delta_n$$
- $|\delta_i| \le \mu, 2 \le i \le n$

Proof.

- $s = x_1 + x_2 + \ldots + x_n$
- $s_2 = fl(x_1 + x_2) = (x_1 + x_2)(1 + \delta_2), |\delta_2| \le \mu$

Round-off errors, 7

Theorem 7.9

Round-off error in floating point addition

- $x_i \in \mathbb{R}, 2 \leq i \leq n$
- $$fl(x_1 + x_2 + \ldots + x_n) - (x_1 + x_2 + \ldots + x_n) \approx$$
$$(x_1 + x_2)(\delta_2 + \delta_3 + \ldots + \delta_n) + x_3(\delta_3 + \delta_4 + \ldots + \delta_n) + \ldots + x_n\delta_n$$
- $|\delta_i| \leq \mu, 2 \leq i \leq n$

Proof.

- $s = x_1 + x_2 + \ldots + x_n$
- $s_2 = fl(x_1 + x_2) = (x_1 + x_2)(1 + \delta_2), |\delta_2| \leq \mu$
- $s_i = fl(s_{i-1} + x_i) = (s_{i-1} + x_i)(1 + \delta_i), |\delta_i| \leq \mu, 2 \leq i \leq n$

Round-off errors, 7

Theorem 7.9

Round-off error in floating point addition

- $x_i \in \mathbb{R}, 2 \leq i \leq n$
- $$fl(x_1 + x_2 + ... + x_n) - (x_1 + x_2 + ... + x_n) \approx$$
  $$(x_1 + x_2)(\delta_2 + \delta_3 + ... + \delta_n) + x_3(\delta_3 + \delta_4 + ... + \delta_n) + ... + x_n\delta_n$$
- $|\delta_i| \leq \mu, 2 \leq i \leq n$

Proof.

- $s = x_1 + x_2 + ... + x_n$
- $s_2 = fl(x_1 + x_2) = (x_1 + x_2)(1 + \delta_2), |\delta_2| \leq \mu$
- $s_i = fl(s_{i-1} + x_i) = (s_{i-1} + x_i)(1 + \delta_i), |\delta_i| \leq \mu, 2 \leq i \leq n$
- $s_3 = fl(s_2 + x_3) = (s_2 + x_3)(1 + \delta_3) =$

Round-off errors, 8

Round-off error in floating point addition, cont.

Proof. cont.

- $s_3 = (x_1 + x_2)(1 + \delta_2 + \delta_3) + x_3(1 + \delta_3) + O(\mu^2)$

Round-off errors, 8

Round-off error in floating point addition, cont.

Proof. cont.

- $s_3 = (x_1 + x_2)(1 + \delta_2 + \delta_3) + x_3(1 + \delta_3) + O(\mu^2)$
- $s_3 - (x_1 + x_2 + x_3) = (x_1 + x_2)(\delta_2 + \delta_3) + x_3\delta_3 + O(\mu^2) \approx$
  $(x_1 + x_2)(\delta_2 + \delta_3) + x_3\delta_3 = (x_1 + x_2)\delta_2 + (x_1 + x_2 + x_3)\delta_3$

Round-off errors, 8

Round-off error in floating point addition, cont.

Proof. cont.

- $s_3 = (x_1 + x_2)(1 + \delta_2 + \delta_3) + x_3(1 + \delta_3) + O(\mu^2)$
- $s_3 - (x_1 + x_2 + x_3) = (x_1 + x_2)(\delta_2 + \delta_3) + x_3\delta_3 + O(\mu^2) \approx$
  $(x_1 + x_2)(\delta_2 + \delta_3) + x_3\delta_3 = (x_1 + x_2)\delta_2 + (x_1 + x_2 + x_3)\delta_3$
- $s_i - (x_1 + x_2 + ... + x_i) \approx$
  $(x_1 + x_2)\delta_2 + (x_1 + x_2 + x_3)\delta_3 + ... + (x_1 + x_2 + ... + x_i)\delta_i$

Round-off errors, 8

Round-off error in floating point addition, cont.

Proof. cont.

- $s_3 = (x_1 + x_2)(1 + \delta_2 + \delta_3) + x_3(1 + \delta_3) + O(\mu^2)$
- $s_3 - (x_1 + x_2 + x_3) = (x_1 + x_2)(\delta_2 + \delta_3) + x_3\delta_3 + O(\mu^2) \approx$
  $(x_1 + x_2)(\delta_2 + \delta_3) + x_3\delta_3 = (x_1 + x_2)\delta_2 + (x_1 + x_2 + x_3)\delta_3$
- $s_i - (x_1 + x_2 + \ldots + x_i) \approx$
- $(x_1 + x_2)\delta_2 + (x_1 + x_2 + x_3)\delta_3 + \ldots + (x_1 + x_2 + \ldots + x_i)\delta_i$

$$
\begin{gathered}
s_{i+1} - (x_1 + x_2 + \ldots + x_{i+1}) = fl(s_i + x_{i+1}) - (x_1 + x_2 + \ldots + x_{i+1}) = \\
(s_i + x_{i+1})(1 + \delta_{i+1}) - (x_1 + x_2 + \ldots + x_{i+1}) = \\
(s_i - (x_1 + x_2 + \ldots + x_i)) + s_i\delta_{i+1} + x_{i+1}\delta_{i+1} = \\
\approx (x_1 + x_2)(\delta_2 + \delta_3 + \ldots + \delta_i) + \ldots + x_i\delta_i + s_i\delta_{i+1} + x_{i+1}\delta_{i+1} = \\
(x_1 + x_2)(\delta_2 + \delta_3 + \ldots + \delta_i) + \ldots + x_i\delta_i + \\
(x_1 + x_2 + \ldots + x_i)\delta_{i+1} + O(\mu^2) + +x_{i+1}\delta_{i+1} = \\
\approx (x_1 + x_2)(\delta_2 + \delta_3 + \ldots + \delta_i) + \ldots + x_i\delta_i + \\
(x_1 + x_2 + \ldots + x_i + x_{i+1})\delta_{i+1}
\end{gathered}
$$

Round-off errors, 9

Laws of floating point arithmetic

$$fl(x \odot y) = (x \odot y)(1 + \delta)$$

$$\odot = +, -, *, /$$

$$|\delta| \leq \mu$$

Round-off errors, 9

Laws of floating point arithmetic

$$fl(x \odot y) = (x \odot y)(1 + \delta)$$

$$\odot = +, -, *, /$$

$$|\delta| \leq \mu$$

Example 7.10

Biswa Nath Datta, 2010

$$\begin{aligned} fl(x(y+z)) &= [x fl(y+z)](1+\delta_1) \\ &= x(y+z)(1+\delta_1)(1+\delta_2) = \\ &= x(y+z)(1+\delta_1+\delta_2+\delta_1\delta_2) = \\ &\approx x(y+z)(1+\delta_1+\delta_2) \end{aligned}$$

Round-off errors, 10

Theorem 7.11

Wilkinson, 1965

1. $|M| = (|m_{ij}|)$

Round-off errors, 10

Theorem 7.11

Wilkinson, 1965

1. $|M| = (|m_{ij}|)$
2. $A, B \in \mathcal{R}^{n \times n}$

Round-off errors, 10

Theorem 7.11

Wilkinson, 1965

1. $|M| = (|m_{ij}|)$
2. $A, B \in \mathcal{R}^{n \times n}$
3. $fl(cA) = cA + E, |E| \leq \mu |cA|$

Round-off errors, 10

Theorem 7.11

Wilkinson, 1965

1. $|M| = (|m_{ij}|)$
2. $A, B \in \mathcal{R}^{n\times n}$
3. $fl(cA) = cA + E, |E| \leq \mu|cA|$
4. $fl(A+B) = (A+B) + E, |E| \leq \mu|A+B|$

Round-off errors, 10

Theorem 7.11

Wilkinson, 1965

1. $|M| = (|m_{ij}|)$
2. $A, B \in \mathcal{R}^{n\times n}$
3. $fl(cA) = cA + E, |E| \leq \mu|cA|$
4. $fl(A+B) = (A+B) + E, |E| \leq \mu|A+B|$
5. $fl(AB) = AB + E, |E| \leq n\mu|A|\;|B| + O(\mu^2)$

Round-off errors, 10

Theorem 7.11

Wilkinson, 1965

1. $|M| = (|m_{ij}|)$
2. $A, B \in \mathcal{R}^{n\times n}$
3. $fl(cA) = cA + E, |E| \leq \mu|cA|$
4. $fl(A+B) = (A+B) + E, |E| \leq \mu|A+B|$
5. $fl(AB) = AB + E, |E| \leq n\mu|A|\ |B| + O(\mu^2)$

Floating point errors and matrix operations

Biswa Nath Datta, 2010

1. $\|fl(AB) - AB\|_1 \leq n\mu\|A\|_1\|B\|_1 + O(\mu^2), A, B \in \mathcal{R}^{n\times n}$

Round-off errors, 10

Theorem 7.11

Wilkinson, 1965

1. $|M| = (|m_{ij}|)$
2. $A, B \in \mathcal{R}^{n \times n}$
3. $fl(cA) = cA + E, |E| \leq \mu |cA|$
4. $fl(A + B) = (A + B) + E, |E| \leq \mu |A + B|$
5. $fl(AB) = AB + E, |E| \leq n\mu |A|\ |B| + O(\mu^2)$

Floating point errors and matrix operations

Biswa Nath Datta, 2010

1. $\|fl(AB) - AB\|_1 \leq n\mu \|A\|_1 \|B\|_1 + O(\mu^2), A, B \in \mathcal{R}^{n \times n}$
2. $\|fl(Ab) - Ab\|_1 \leq n\mu \|A\|_1 \|b\|_1, A \in \mathcal{R}^{n \times n}, b \in \mathcal{R}^n$

Round-off errors, 10

Theorem 7.11

Wilkinson, 1965

1. $|M| = (|m_{ij}|)$
2. $A, B \in \mathcal{R}^{n\times n}$
3. $fl(cA) = cA + E, |E| \leq \mu|cA|$
4. $fl(A+B) = (A+B) + E, |E| \leq \mu|A+B|$
5. $fl(AB) = AB + E, |E| \leq n\mu|A|\ |B| + O(\mu^2)$

Floating point errors and matrix operations

Biswa Nath Datta, 2010

1. $\|fl(AB) - AB\|_1 \leq n\mu\|A\|_1\|B\|_1 + O(\mu^2), A, B \in \mathcal{R}^{n\times n}$
2. $\|fl(Ab) - Ab\|_1 \leq n\mu\|A\|_1\|b\|_1, A \in \mathcal{R}^{n\times n}, b \in \mathcal{R}^n$
3. $\|fl(QB) - QB\|_F \leq n\mu\|A\|_F, A, Q \in \mathcal{R}^{n\times n}, Q^TQ = I$

Avoid round-off errors due to recursion and cancellation, 1

Theorem 7.12

Round-off error in floating point addition: $x_i \in \mathbb{R}, 2 \leq i \leq n$

$$fl(x_1 + x_2 + ... + x_n) - (x_1 + x_2 + ... + x_n) \approx$$
$$(x_1 + x_2)(\delta_2 + \delta_3 + ... + \delta_n) + x_3(\delta_3 + \delta_4 + ... + \delta_n) + ... + x_n\delta_n$$

$|\delta_i| \leq \mu, 2 \leq i \leq n$

Avoid round-off errors due to recursion and cancellation, 1

Theorem 7.12

Round-off error in floating point addition: $x_i \in \mathbb{R}, 2 \leq i \leq n$

$$fl(x_1 + x_2 + ... + x_n) - (x_1 + x_2 + ... + x_n) \approx$$
$$(x_1 + x_2)(\delta_2 + \delta_3 + ... + \delta_n) + x_3(\delta_3 + \delta_4 + ... + \delta_n) + ... + x_n\delta_n$$

$|\delta_i| \leq \mu, 2 \leq i \leq n$

Example 7.13

$S_1 = \sum_{i=1}^{n} \frac{1}{i}, \; S_2 = \sum_{i=n}^{1} \frac{1}{i}$

n	SUM from 1 to n	SUM from n to 1	Difference
10	2.8289682539682537	1.928968253968254	0.8999999999999997
100	5.177377517639621	4.1873775176396215	0.9899999999999993
1000	7.484470860550343	6.485470860550341	0.9990000000000023
10000	9.787506036044348	8.787606036044386	0.9998999999999629
100000	12.090136129863335	11.090146129863408	0.9999899999999275
1000000	14.39272572286499	13.392726722865772	0.9999989999992174
10000000	16.69531126585727	15.695311365859965	0.9999998999973059
100000000	18.997896403852554	17.997896413853447	0.9999999899991074

Figure: which sum is more accurate, S_1 or S_2?

Avoid round-off errors due to recursion and cancellation, 2

Theorem 7.14

Round-off error in floating point addition: $x_i \in \mathbb{R}, 2 \leq i \leq n$

$$fl(x_1 + x_2 + ... + x_n) - (x_1 + x_2 + ... + x_n) \approx$$
$$(x_1 + x_2)(\delta_2 + \delta_3 + ... + \delta_n) + x_3(\delta_3 + \delta_4 + ... + \delta_n) + ... + x_n\delta_n$$

$|\delta_i| \leq \mu, 2 \leq i \leq n$

Avoid round-off errors due to recursion and cancellation, 2

Theorem 7.14

Round-off error in floating point addition: $x_i \in \mathbb{R}, 2 \leq i \leq n$

$$fl(x_1 + x_2 + \ldots + x_n) - (x_1 + x_2 + \ldots + x_n) \approx$$
$$(x_1 + x_2)(\delta_2 + \delta_3 + \ldots + \delta_n) + x_3(\delta_3 + \delta_4 + \ldots + \delta_n) + \ldots + x_n\delta_n$$

$|\delta_i| \leq \mu, 2 \leq i \leq n$

Example 7.15

- $S_1 = \sum_{i=1}^{n} \frac{1}{i},\ S_2 = \sum_{i=n}^{1} \frac{1}{i}$
- Recommendation:
  summation from smallest to largest terms is more accurate

Avoid round-off errors due to recursion and cancellation, 3

Example 7.16

- Cancellation error, compute function for small x: $f(x) = \frac{1-cos(x)}{x^2}$

Avoid round-off errors due to recursion and cancellation, 3

Example 7.16

- Cancellation error, compute function for small x: $f(x) = \frac{1-cos(x)}{x^2}$

```
In [14]: x=0.001

In [15]: print((1-math.cos(x))/(x*x))
0.49999995832550326

In [16]: x=0.000001

In [17]: print((1-math.cos(x))/(x*x))
0.5000444502911705

In [18]: x=0.000000000001

In [19]: print((1-math.cos(x))/(x*x))
0.0
```

```
In [18]: x=0.001

In [19]: f=(math.sin(x)**2)/( (1+math.cos(x))*x**2); print(f)
0.4999999583333347

In [20]: x=0.000001

In [21]: f=(math.sin(x)**2)/( (1+math.cos(x))*x**2); print(f)
0.4999999999999583

In [22]: x=0.000000000001

In [23]: f=(math.sin(x)**2)/( (1+math.cos(x))*x**2); print(f)
0.5
```

Figure: Cancellation errors: left=wrong approach, right=correct approach

Avoid round-off errors due to recursion and cancellation, 3

Example 7.16

- Cancellation error, compute function for small x: $f(x) = \frac{1-cos(x)}{x^2}$

```
In [14]: x=0.001

In [15]: print((1-math.cos(x))/(x*x))
0.49999995832550326

In [16]: x=0.000001

In [17]: print((1-math.cos(x))/(x*x))
0.5000444502911705

In [18]: x=0.000000000001

In [19]: print((1-math.cos(x))/(x*x))
0.0
```

```
In [18]: x=0.001

In [19]: f=(math.sin(x)**2)/( (1+math.cos(x))*x**2); print(f)
0.4999999583333347

In [20]: x=0.000001

In [21]: f=(math.sin(x)**2)/( (1+math.cos(x))*x**2); print(f)
0.4999999999999583

In [22]: x=0.000000000001

In [23]: f=(math.sin(x)**2)/( (1+math.cos(x))*x**2); print(f)
0.5
```

Figure: Cancellation errors: left=wrong approach, right=correct approach

- Remedy: rewrite formula equivalently for avoiding cancellation
$1 - cos(x) = \frac{sin^2(x)}{1+cos(x)}$, $f(x) = \frac{sin^2(x)}{(1+cos(x))x^2}$

Avoid round-off errors due to recursion and cancellation, 4

Example 7.17

```
In [19]: a=1; b=1

In [20]: print(1/(1/a+1/b))
0.5

In [21]: print(a*b/(a+b))
0.5
```

```
In [22]: a=1; b=0.

In [23]: print(a*b/(a+b))
0.0

In [24]: print(1/(1/a+1/b))
Traceback (most recent call last):

  File "C:\Users\KiuAdmin\AppData\Local\Temp\ipykernel_3120\599846102.py", line 1, in
<module>
    print(1/(1/a+1/b))

ZeroDivisionError: float division by zero
```

Figure: Same expressions written differently give different results

Avoid round-off errors due to recursion and cancellation, 5

Example 7.18

Biswa Nath Datta, 2010

1. Exact input data:
    1.1 $x = 0.54617, y = 0.54601$

Avoid round-off errors due to recursion and cancellation, 5

Example 7.18

Biswa Nath Datta, 2010

1. Exact input data:
   1.1 $x = 0.54617, y = 0.54601$
   1.2 $d = x - y = 0.00016$

Avoid round-off errors due to recursion and cancellation, 5

Example 7.18

Biswa Nath Datta, 2010

1. Exact input data:
   1.1 $x = 0.54617, y = 0.54601$
   1.2 $d = x - y = 0.00016$
2. 4-digit arithmetic with rounding

Avoid round-off errors due to recursion and cancellation, 5

Example 7.18

Biswa Nath Datta, 2010

1. Exact input data:
   1.1 $x = 0.54617, y = 0.54601$
   1.2 $d = x - y = 0.00016$
2. 4-digit arithmetic with rounding
   2.1 $\tilde{x} = 0.5462, \tilde{y} = 0.5460$

Avoid round-off errors due to recursion and cancellation, 5

Example 7.18

Biswa Nath Datta, 2010

1. Exact input data:
   1.1 $x = 0.54617, y = 0.54601$
   1.2 $d = x - y = 0.00016$
2. 4-digit arithmetic with rounding
   2.1 $\tilde{x} = 0.5462, \tilde{y} = 0.5460$
   2.2 $\tilde{d} = \tilde{x} - \tilde{y} = 0.0002$

Avoid round-off errors due to recursion and cancellation, 5

Example 7.18

Biswa Nath Datta, 2010

1. Exact input data:
   1.1 $x = 0.54617, y = 0.54601$
   1.2 $d = x - y = 0.00016$
2. 4-digit arithmetic with rounding
   2.1 $\tilde{x} = 0.5462, \tilde{y} = 0.5460$
   2.2 $\tilde{d} = \tilde{x} - \tilde{y} = 0.0002$
   2.3 Large relative error $\frac{|d-\tilde{d}|}{|d|} = 0.25$

Avoid round-off errors due to recursion and cancellation, 5

Example 7.18

Biswa Nath Datta, 2010

1. Exact input data:
   1.1 $x = 0.54617, y = 0.54601$
   1.2 $d = x - y = 0.00016$
2. 4-digit arithmetic with rounding
   2.1 $\tilde{x} = 0.5462, \tilde{y} = 0.5460$
   2.2 $\tilde{d} = \tilde{x} - \tilde{y} = 0.0002$
   2.3 Large relative error $\frac{|d-\tilde{d}|}{|d|} = 0.25$
3. Catastrophic cancellation: two numbers of approximately same size substracted

Avoid round-off errors due to recursion and cancellation, 5

Example 7.18

Biswa Nath Datta, 2010

1. Exact input data:
   1.1 $x = 0.54617, y = 0.54601$
   1.2 $d = x - y = 0.00016$
2. 4-digit arithmetic with rounding
   2.1 $\tilde{x} = 0.5462, \tilde{y} = 0.5460$
   2.2 $\tilde{d} = \tilde{x} - \tilde{y} = 0.0002$
   2.3 Large relative error $\frac{|d-\tilde{d}|}{|d|} = 0.25$
3. Catastrophic cancellation: two numbers of approximately same size substracted
4. Notice: substraction reveals errors of previous computations

Avoid round-off errors due to recursion and cancellation, 5

Example 7.18

Biswa Nath Datta, 2010

1. Exact input data:
   1.1 $x = 0.54617, y = 0.54601$
   1.2 $d = x - y = 0.00016$
2. 4-digit arithmetic with rounding
   2.1 $\tilde{x} = 0.5462, \tilde{y} = 0.5460$
   2.2 $\tilde{d} = \tilde{x} - \tilde{y} = 0.0002$
   2.3 Large relative error $\frac{|d-\tilde{d}|}{|d|} = 0.25$
3. Catastrophic cancellation: two numbers of approximately same size substracted
4. Notice: substraction reveals errors of previous computations

Round-off errors due to recursion

Avoid round-off errors due to recursion and cancellation, 5

Example 7.18

Biswa Nath Datta, 2010

1. Exact input data:
   1.1 $x = 0.54617, y = 0.54601$
   1.2 $d = x - y = 0.00016$
2. 4-digit arithmetic with rounding
   2.1 $\tilde{x} = 0.5462, \tilde{y} = 0.5460$
   2.2 $\tilde{d} = \tilde{x} - \tilde{y} = 0.0002$
   2.3 Large relative error $\frac{|d - \tilde{d}|}{|d|} = 0.25$
3. Catastrophic cancellation: two numbers of approximately same size substracted
4. Notice: substraction reveals errors of previous computations

Round-off errors due to recursion

- p.43, Biswa Nath Datta, 2010,
  $E_n = 1 - nE_{n-1}, n = 2, 3.., E_n = \int_o^1 x^n e^{x-1} dx$

Avoid round-off errors due to recursion and cancellation, 5

Example 7.18

Biswa Nath Datta, 2010

1. Exact input data:
   1.1 $x = 0.54617, y = 0.54601$
   1.2 $d = x - y = 0.00016$
2. 4-digit arithmetic with rounding
   2.1 $\tilde{x} = 0.5462, \tilde{y} = 0.5460$
   2.2 $\tilde{d} = \tilde{x} - \tilde{y} = 0.0002$
   2.3 Large relative error $\frac{|d - \tilde{d}|}{|d|} = 0.25$
3. Catastrophic cancellation: two numbers of approximately same size substracted
4. Notice: substraction reveals errors of previous computations

Round-off errors due to recursion

- p.43, Biswa Nath Datta, 2010,
  $E_n = 1 - nE_{n-1}, n = 2, 3.., E_n = \int_o^1 x^n e^{x-1} dx$
- Recommendation: rearrange recursion

Computational template of Linear Algebra, 1

1. Transform the problem to "easier to solve" form, e.g. matrices of the problem into matrices with special structure

Computational template of Linear Algebra, 1

1. Transform the problem to "easier to solve" form, e.g. matrices of the problem into matrices with special structure

Example 7.19

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ .. & .. & \dots & .. \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \longrightarrow \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22} & \dots & a_{2n} \\ .. & .. & \dots & .. \\ 0 & 0 & \dots & a_{nn} \end{pmatrix}$$

Computational template of Linear Algebra, 1

1. Transform the problem to "easier to solve" form, e.g. matrices of the problem into matrices with special structure

Example 7.19

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ .. & .. & \dots & .. \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \longrightarrow \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22} & \dots & a_{2n} \\ .. & .. & \dots & .. \\ 0 & 0 & \dots & a_{nn} \end{pmatrix}$$

2. Exploit special structure of associated matrices and solve transformed problem

Computational template of Linear Algebra, 1

1. Transform the problem to "easier to solve" form, e.g. matrices of the problem into matrices with special structure

Example 7.19

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ .. & .. & \dots & .. \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \longrightarrow \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22} & \dots & a_{2n} \\ .. & .. & \dots & .. \\ 0 & 0 & \dots & a_{nn} \end{pmatrix}$$

2. Exploit special structure of associated matrices and solve transformed problem
3. From the solution of transformed problem recover solution of the original problem

Computational template of Linear Algebra, 2

Definition 7.20

LU Decomposition, LU Factorisation

$$A = LU$$

A, L, U-square matrices, L-lower triangular, U - upper triangular

Computational template of Linear Algebra, 2

Definition 7.20

LU Decomposition, LU Factorisation

$$A = LU$$

A, L, U-square matrices, L-lower triangular, U - upper triangular

Example 7.21

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ .. & .. & \dots & .. \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ .. & .. & \dots & .. \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ .. & .. & \dots & .. \\ 0 & 0 & \dots & u_{nn} \end{pmatrix}$$

Computational template of Linear Algebra, 2

Definition 7.20

LU Decomposition, LU Factorisation

$$A = LU$$

A, L, U-square matrices, L-lower triangular, U - upper triangular

Example 7.21

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ .. & .. & \dots & .. \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ .. & .. & \dots & .. \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ .. & .. & \dots & .. \\ 0 & 0 & \dots & u_{nn} \end{pmatrix}$$

Example 7.22

LU factoriozation for solving linear systems

$$Ax = b \Rightarrow A = LU, LUx = b \Rightarrow\Rightarrow L(Ux) = b, Ly = b \Rightarrow Ux = y$$

Thomas algorithm, 1

- LU Decompositionm of square tridiagonal matrices

Thomas algorithm, 1

- LU Decompositionm of square tridiagonal matrices
- Efficient for solving linear systems for tridiagonal matrices:

Thomas algorithm, 1

- LU Decompositionm of square tridiagonal matrices
- Efficient for solving linear systems for tridiagonal matrices:
  - Cramer's rule $O(n!)$ flops

Thomas algorithm, 1

- LU Decompositionm of square tridiagonal matrices
- Efficient for solving linear systems for tridiagonal matrices:
  - Cramer's rule $O(n!)$ flops
  - Gaussian elimination $O(n^3)$ flops

Thomas algorithm, 1

- LU Decompositionm of square tridiagonal matrices
- Efficient for solving linear systems for tridiagonal matrices:
  - Cramer's rule $O(n!)$ flops
  - Gaussian elimination $O(n^3)$ flops
  - Thomas algorithm $O(n)$ flops

Thomas algorithm, 1

- LU Decompositionm of square tridiagonal matrices
- Efficient for solving linear systems for tridiagonal matrices:
  - Cramer's rule $O(n!)$ flops
  - Gaussian elimination $O(n^3)$ flops
  - Thomas algorithm $O(n)$ flops
  - Fastest Computer today in development $=$ hexapoint(10^{18}) operations per second

Thomas algorithm, 1

- LU Decompositionm of square tridiagonal matrices
- Efficient for solving linear systems for tridiagonal matrices:
    - Cramer's rule $O(n!)$ flops
    - Gaussian elimination $O(n^3)$ flops
    - Thomas algorithm $O(n)$ flops
    - Fastest Computer today in development = hexapoint(10^{18}) operations per second
    - How fast can you solve linear system with $10^3, 10^6, 10^9$ unknowns?

Thomas algorithm, 1

- LU Decompositionm of square tridiagonal matrices
- Efficient for solving linear systems for tridiagonal matrices:
  - Cramer's rule $O(n!)$ flops
  - Gaussian elimination $O(n^3)$ flops
  - Thomas algorithm $O(n)$ flops
  - Fastest Computer today in development = hexapoint(10^{18}) operations per second
  - How fast can you solve linear system with $10^3, 10^6, 10^9$ unknowns?

Example 7.23

$$\begin{pmatrix} a_{11} & a_{12} & . & . & . & . & 0 \\ a_{21} & a_{22} & a_{23} & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & a_{ii-1} & a_{ii} & a_{ii+1} & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 0 & . & . & . & . & a_{nn-1} & a_{nn} \end{pmatrix}$$

Thomas algorithm, 1

- LU Decompositionm of square tridiagonal matrices
- Efficient for solving linear systems for tridiagonal matrices:
  - Cramer's rule $O(n!)$ flops
  - Gaussian elimination $O(n^3)$ flops
  - Thomas algorithm $O(n)$ flops
  - Fastest Computer today in development = hexapoint(10^{18}) operations per second
  - How fast can you solve linear system with $10^3, 10^6, 10^9$ unknowns?

Example 7.23

$$\begin{pmatrix} a_{11} & a_{12} & . & . & . & . & 0 \\ a_{21} & a_{22} & a_{23} & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & a_{ii-1} & a_{ii} & a_{ii+1} & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 0 & . & . & . & . & a_{nn-1} & a_{nn} \end{pmatrix}$$

- Better storage scheme: store three "diagonals" ONLY

Thomas algorithm, 2

- Tgridiagonal system

Example 7.24

$$\begin{pmatrix} a_1 & c_1 & . & . & . & . & 0 \\ b_2 & a_2 & c_2 & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & b_i & a_i & c_i & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 0 & . & . & . & . & b_n & a_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ . \\ x_i \\ . \\ . \\ x_n \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ . \\ f_i \\ . \\ . \\ f_n \end{pmatrix}$$

- Storage scheme: vectors a, b, c, f and x

Theorem 7.25

If LU decomposition of a tridiagonal matrix exists then matrices L and U are bi-diagonal

Thomas algorithm, 3

- LU factorization of tgridiagonal system

$$\begin{pmatrix} a_1 & c_1 & . & . & . & . & 0 \\ b_2 & a_2 & c_2 & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & b_i & a_i & c_i & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 0 & . & . & . & . & b_n & a_n \end{pmatrix} =$$

$$\begin{pmatrix} 1 & . & . & . & . & . & 0 \\ \beta_2 & 1 & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & \beta_i & 1 & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 0 & . & . & . & . & \beta_n & 1 \end{pmatrix} \begin{pmatrix} \alpha_1 & c_1 & . & . & . & . & 0 \\ . & \alpha_2 & c_2 & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & \alpha_i & c_i & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 0 & . & . & . & . & . & \alpha_n \end{pmatrix}$$

- Storage scheme: vectors a, b, c and α, β

Thomas algorithm, 4

LU Decomposition

$$
\begin{pmatrix}
1 & . & . & . & . & . & 0 \\
\beta_2 & 1 & . & . & . & . & . \\
. & . & . & . & . & . & . \\
. & . & \beta_i & 1 & . & . & . \\
. & . & . & . & . & . & . \\
. & . & . & . & . & . & . \\
0 & . & . & . & . & \beta_n & 1
\end{pmatrix}
\begin{pmatrix}
\alpha_1 & c_1 & . & . & . & . & 0 \\
. & \alpha_2 & c_2 & . & . & . & . \\
. & . & . & . & . & . & . \\
. & . & . & \alpha_i & c_i & . & . \\
. & . & . & . & . & . & . \\
. & . & . & . & . & . & . \\
0 & . & . & . & . & . & \alpha_n
\end{pmatrix}
=
$$

$$
\begin{pmatrix}
a_1 & c_1 & . & . & . & . & 0 \\
b_2 & a_2 & c_2 & . & . & . & . \\
. & . & . & . & . & . & . \\
. & . & b_i & a_i & c_i & . & . \\
. & . & . & . & . & . & . \\
. & . & . & . & . & . & . \\
0 & . & . & . & . & b_n & a_n
\end{pmatrix}
\Rightarrow
\begin{cases}
\alpha_1 = a_1 \\
\alpha_1 \beta_2 = b_2 \Rightarrow \beta_2 = b_2/\alpha_1 \\
\beta_2 c_1 + \alpha_2 = a_2 \Rightarrow \alpha_2 = a_2 - \beta_2 c_1 \\
.. \\
\alpha_{i-1}\beta_i = b_i \Rightarrow \beta_i = b_i/\alpha_{i-1} \\
\beta_i c_{i-1} + \alpha_i = a_i \Rightarrow \\
\alpha_i = a_i - \beta_i c_{i-1} \\
2 \le i \le n
\end{cases}
$$

Thomas algorithm, 5

LU Decomposition, solving linear system

$$\begin{pmatrix} 1 & . & . & . & . & . & 0 \\ \beta_2 & 1 & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & \beta_i & 1 & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 0 & . & . & . & . & \beta_n & 1 \end{pmatrix} \begin{pmatrix} \alpha_1 & c_1 & . & . & . & . & 0 \\ . & \alpha_2 & c_2 & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & \alpha_i & c_i & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 0 & . & . & . & . & . & \alpha_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ . \\ x_i \\ . \\ . \\ x_n \end{pmatrix} =$$

$$\begin{pmatrix} f_1 \\ f_2 \\ . \\ f_i \\ . \\ . \\ f_n \end{pmatrix} \Rightarrow \begin{cases} Ax = b \\ LUx = f \\ L(Ux) = f \\ Ux = y \\ Ly = f \end{cases}$$

Thomas algorithm, 6

solving linear system, forward substituition

$$\begin{pmatrix} 1 & . & . & . & . & . & 0 \\ \beta_2 & 1 & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & \beta_i & 1 & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 0 & . & . & . & . & \beta_n & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ . \\ y_i \\ . \\ . \\ y_n \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ . \\ f_i \\ . \\ . \\ f_n \end{pmatrix} \Rightarrow \begin{cases} y_1 = f_1 \\ \beta_2 y_1 + y_2 = f_2 \\ y_2 = f_2 - \beta_2 y_1 \\ \ldots \\ \beta_i y_{i-1} + y_i = f_i \\ y_i = f_i - \beta_i y_{i-1} \\ 2 \le i \le n \end{cases}$$

Thomas algorithm, 7

Solving linear system, backward substituition

$$\begin{pmatrix} \alpha_1 & c_1 & . & . & . & . & 0 \\ . & \alpha_2 & c_2 & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & \alpha_i & c_i & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 0 & . & . & . & . & . & \alpha_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ . \\ x_i \\ . \\ . \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ . \\ y_i \\ . \\ . \\ y_n \end{pmatrix} \Rightarrow \begin{cases} \alpha_n x_n = y_n \\ x_n = y_n/\alpha_n \\ \alpha_i x_i + c_i y_{i+1} = f_i \\ x_i = (f_i - c_i y_{i+1})/\alpha_i \\ i = n-1, n-2, ..., 1 \end{cases}$$

Thomas algorithm, 8

1. LU Decomposition

$$\alpha_1 = a_1, \beta_i = b_i/\alpha_{i-1}, \alpha_i = a_i - \beta_i c_{i-1}, i = 2, 3, ..., n$$

2. Forward substitution

Thomas algorithm, 8

1. LU Decomposition

$$\alpha_1 = a_1, \beta_i = b_i/\alpha_{i-1}, \alpha_i = a_i - \beta_i c_{i-1}, i = 2, 3, ..., n$$

2. Forward substitution

$$y_1 = f_1,\ \ y_i = f_i - \beta_i y_{i-1},\ \ i = 2, 3, ..., n$$

3. Backward substitution

Thomas algorithm, 8

1. LU Decomposition

$$\alpha_1 = a_1, \beta_i = b_i/\alpha_{i-1}, \alpha_i = a_i - \beta_i c_{i-1}, i = 2, 3, ..., n$$

2. Forward substitution

$$y_1 = f_1,\ y_i = f_i - \beta_i y_{i-1},\ i = 2, 3, ..., n$$

3. Backward substitution

$$x_n = y_n/\alpha_n,\ x_i = (f_i - c_i y_{i+1})/\alpha_i\ i = n-1, n-2, ..., 1$$

Thomas algorithm, 8

1. LU Decomposition

$$\alpha_1 = a_1, \beta_i = b_i/\alpha_{i-1}, \alpha_i = a_i - \beta_i c_{i-1}, i = 2, 3, ..., n$$

2. Forward substitution

$$y_1 = f_1,\ \ y_i = f_i - \beta_i y_{i-1},\ \ i = 2, 3, ..., n$$

3. Backward substitution

$$x_n = y_n/\alpha_n,\ \ x_i = (f_i - c_i y_{i+1})/\alpha_i\ \ i = n-1, n-2, ..., 1$$

Solving linear system, number of arithmetic operations

1. LU Decomposition $3(n-1)$ operations

Thomas algorithm, 8

1. LU Decomposition

$$\alpha_1 = a_1, \beta_i = b_i/\alpha_{i-1}, \alpha_i = a_i - \beta_i c_{i-1}, i = 2, 3, ..., n$$

2. Forward substitution

$$y_1 = f_1,\ y_i = f_i - \beta_i y_{i-1},\ i = 2, 3, ..., n$$

3. Backward substitution

$$x_n = y_n/\alpha_n,\ x_i = (f_i - c_i y_{i+1})/\alpha_i\ i = n-1, n-2, ..., 1$$

Solving linear system, number of arithmetic operations

1. LU Decomposition $3(n-1)$ operations
2. Forward substitution $2(n-1)$ operations

Thomas algorithm, 8

1. LU Decomposition

$$\alpha_1 = a_1, \beta_i = b_i/\alpha_{i-1}, \alpha_i = a_i - \beta_i c_{i-1}, i = 2, 3, ..., n$$

2. Forward substitution

$$y_1 = f_1,\ \ y_i = f_i - \beta_i y_{i-1},\ \ i = 2, 3, ..., n$$

3. Backward substitution

$$x_n = y_n/\alpha_n,\ \ x_i = (f_i - c_i y_{i+1})/\alpha_i\ \ i = n-1, n-2, ..., 1$$

Solving linear system, number of arithmetic operations

1. LU Decomposition $3(n-1)$ operations
2. Forward substitution $2(n-1)$ operations
3. Backward substitution $3(n-1)+1$ operations

Thomas algorithm, 8

1. LU Decomposition

$$\alpha_1 = a_1, \beta_i = b_i/\alpha_{i-1}, \alpha_i = a_i - \beta_i c_{i-1}, i = 2, 3, ..., n$$

2. Forward substitution

$$y_1 = f_1,\ \ y_i = f_i - \beta_i y_{i-1},\ \ i = 2, 3, ..., n$$

3. Backward substitution

$$x_n = y_n/\alpha_n,\ \ x_i = (f_i - c_i y_{i+1})/\alpha_i\ \ i = n-1, n-2, ..., 1$$

Solving linear system, number of arithmetic operations

1. LU Decomposition $3(n-1)$ operations
2. Forward substitution $2(n-1)$ operations
3. Backward substitution $3(n-1)+1$ operations
4. Total $8(n-1) \approx O(n)$ operations

Thomas algorithm, 10

- Q: When is LU Decomposition very useful?

Thomas algorithm, 10

- Q: When is LU Decomposition very useful?
- A: If solving the same system with many different right hand sides is needed

Thomas algorithm, 10

- Q: When is LU Decomposition very useful?
- A: If solving the same system with many different right hand sides is needed

Example 7.26

- Finding inverse of a matrix

Thomas algorithm, 10

- Q: When is LU Decomposition very useful?
- A: If solving the same system with many different right hand sides is needed

Example 7.26

- Finding inverse of a matrix
- $AB = I$ or $BA = I \Rightarrow B = A^{-1}$

Thomas algorithm, 10

- Q: When is LU Decomposition very useful?
- A: If solving the same system with many different right hand sides is needed

Example 7.26

- Finding inverse of a matrix
- $AB = I$ or $BA = I \Rightarrow B = A^{-1}$
- $I = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ & & \dots & 0 \\ 0 & 0 & \dots & 1 \end{pmatrix} I_1 = \begin{pmatrix} 1 \\ 0 \\ \\ 0 \end{pmatrix} I_2 = \begin{pmatrix} 0 \\ 1 \\ \\ 0 \end{pmatrix} \dots I_n = \begin{pmatrix} 0 \\ 0 \\ \\ 1 \end{pmatrix} \Rightarrow I = (I_1 \;\; I_2 \; \dots \; I_n)$

Thomas algorithm, 10

- Q: When is LU Decomposition very useful?
- A: If solving the same system with many different right hand sides is needed

Example 7.26

- Finding inverse of a matrix
- $AB = I$ or $BA = I \Rightarrow B = A^{-1}$
- $I = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ & & \dots & 0 \\ 0 & 0 & \dots & 1 \end{pmatrix}$ $I_1 = \begin{pmatrix} 1 \\ 0 \\ \\ 0 \end{pmatrix}$ $I_2 = \begin{pmatrix} 0 \\ 1 \\ \\ 0 \end{pmatrix}$ $\dots I_n = \begin{pmatrix} 0 \\ 0 \\ \\ 1 \end{pmatrix}$ $\Rightarrow I = (I_1 \;\; I_2 \;\; \dots \;\; I_n)$
- $AB = I \Rightarrow Ab_i = I_i, i = 1, 2, ..n \Rightarrow B = (b_1 \;\; b_2 \;\; \dots b_n) = A^{-1}$

Thomas algorithm, 11

- Q: Is Thomas algorithm always good to apply?

Thomas algorithm, 11

- Q: Is Thomas algorithm always good to apply?
- A: No:

Thomas algorithm, 11

- Q: Is Thomas algorithm always good to apply?
- A: No:
  - if matrix is degenerate

Thomas algorithm, 11

- Q: Is Thomas algorithm always good to apply?
- A: No:
  - if matrix is degenerate
  - if stability conditions are not satisfied

Thomas algorithm, 11

- Q: Is Thomas algorithm always good to apply?
- A: No:
  - if matrix is degenerate
  - if stability conditions are not satisfied

$$|a_1| > |c_1| > 0,$$
$$|a_i| \geq |b_i| + |c_i|, i = 2, 3, ..., n-1,$$
$$|a_n| > |c_n| > 0.$$

Thomas algorithm, 11

- Q: Is Thomas algorithm always good to apply?
- A: No:
  - if matrix is degenerate
  - if stability conditions are not satisfied

$$|a_1| > |c_1| > 0,$$

$$|a_i| \geq |b_i| + |c_i|, i = 2, 3, ..., n-1,$$

$$|a_n| > |c_n| > 0.$$

Example 7.27

```
lower diagonal =  [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
main diagonal =  [-2. -2. -2. -2. -2. -2. -2. -2. -2. -2.]
upper diagonal =  [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
right hand side =  [-1.  0.  0.  0.  0.  0.  0.  0.  0. -1.]
solution =  [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

Figure: Stability condition satisfied

Thomas algorithm, 12

Example 7.28

```
lower diagonal =  [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
main diagonal =  [-2. -2. -2. -2. -2. -2. -2. -2. -2. -2.]
upper diagonal =  [3. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
right hand side =  [1. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
solution =  [nan nan nan nan nan nan nan nan nan nan]
```

Figure: Stability condition is not satisfied

Example 7.29

```
lower diagonal =  [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
main diagonal =  [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
upper diagonal =  [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
right hand side =  [2. 2. 2. 2. 2. 2. 2. 2. 2. 2.]
solution =  [nan nan nan nan nan nan nan nan nan nan]
```

Figure: Stability condition is not satisfied

Thomas algorithm, 13

Stability

- strictly diagonally dominant by rows

Thomas algorithm, 13

Stability

- strictly diagonally dominant by rows
- $$|a_1| > |c_1| > 0,$$
  $$|a_i| > |b_i| + |c_i|, i = 2, 3, ..., n-1,$$
  $$|a_n| > |c_n| > 0.$$

Thomas algorithm, 13

Stability

- strictly diagonally dominant by rows
- $$|a_1| > |c_1| > 0,$$
  $$|a_i| > |b_i| + |c_i|, i = 2, 3, ..., n-1,$$
  $$|a_n| > |c_n| > 0.$$
- Assume perturbations in RHS $Ax = f, A\tilde{x} = \tilde{f}$

Thomas algorithm, 13

Stability

- strictly diagonally dominant by rows
- $$|a_1| > |c_1| > 0,$$
$$|a_i| > |b_i| + |c_i|, i = 2, 3, ..., n-1,$$
$$|a_n| > |c_n| > 0.$$
- Assume perturbations in RHS $Ax = f, A\tilde{x} = \tilde{f}$
- $A(x - \tilde{x}) = f - \tilde{f}$

Thomas algorithm, 13

Stability

- strictly diagonally dominant by rows
- $$|a_1| > |c_1| > 0,$$
  $$|a_i| > |b_i| + |c_i|, i = 2, 3, ..., n-1,$$
  $$|a_n| > |c_n| > 0.$$
- Assume perturbations in RHS $Ax = f, A\tilde{x} = \tilde{f}$
- $A(x - \tilde{x}) = f - \tilde{f}$
- $b_i(x_{i-1} - \tilde{x}_{i-1}) + a_i(x_i - \tilde{x}_i) + c_i(x_{i+1} - \tilde{x}_{i+1}) = f_i - \tilde{f}_i$

Thomas algorithm, 13

Stability

- strictly diagonally dominant by rows
- $$|a_1| > |c_1| > 0,$$
  $$|a_i| > |b_i| + |c_i|, i = 2, 3, ..., n-1,$$
  $$|a_n| > |c_n| > 0.$$
- Assume perturbations in RHS $Ax = f, A\tilde{x} = \tilde{f}$
- $A(x - \tilde{x}) = f - \tilde{f}$
- $b_i(x_{i-1} - \tilde{x}_{i-1}) + a_i(x_i - \tilde{x}_i) + c_i(x_{i+1} - \tilde{x}_{i+1}) = f_i - \tilde{f}_i$

$$\Downarrow$$

- $\|x - \tilde{x}\|_\infty \leq \frac{1}{|a_i| - |b_i| - |c_i|} \|f - \tilde{f}\|_\infty$

Q & A