Numerical Linear Algebra for Computational Science and Information Engineering

Lecture 04
Direct Methods for Dense Linear Systems

Nicolas Venkovic
nicolas.venkovic@tum.de

Group of Computational Mathematics
School of Computation, Information and Technology
Technical University of Munich

Winter 2025-26

Outline I

Methods to solve linear systems

Problem

Solve for x such that $Ax = b$ where A is an invertible matrix.

To solve this problem, we distinguish between two types of methods:

- **Direct methods**: Deploy a predictable sequence of operations to yield the exact solution (assuming exact arithmetic).
  - **Gaussian elimination**: Efficiently solves isolated systems.
  - **LU factorization**: Leverages $A = LU$, reusable for multiple right-hand sides.
  - **Cholesky factorization**: Leverages $A = LL^H$ for Hermitian positive definite matrices, reusable for multiple right-hand sides.

  In this lecture: Reminders, special cases, basic aspects of performance optimization, stability issues, and pivoting strategies.
- **Iterative methods**: Form successive approximations to the solution using $z \mapsto Az$ at each iteration.
  - **Stationary methods**: Use a consistent update formula, e.g., Jacobi, Gauss-Seidel, ...
  - **Krylov subspace methods**: Build solution in expanding (Krylov) subspaces, e.g., CG, GMRES, ...

Gaussian elimination

Section 3.1 in Darve & Wootters (2021)

Solving isolated linear systems

- Special matrices (more efficient than general case):
  - **Diagonal** matrices: **Element-wise division** — n ops.
    For $D = \mathrm{diag}(d_1, \ldots, d_n)$, solve $Dx = b$ with $x_i = b_i/d_i$.
  - **Tridiagonal** matrices: **Thomas algorithm** — $O(n)$ ops.
    A specialized form of the more general Gaussian elimination algorithm.
  - **Lower triangular** matrices: **Forward substitution** — n^2 ops.
    Start from first equation, solve downwards.
  - **Upper triangular** matrices: **Backward substitution** — n^2 ops.
    Start from last equation, solve upwards.
- **Row echelon form**:
  - Matrix where the leading non-zero coefficient (**pivot**) of each row is strictly to the right of the pivot of the row above it.
- General matrices: **Gaussian elimination** — $O(n^3)$ ops.
  Doolittle (~~Crout~~) variant:
  1. **Forward elimination**: From $[A|b]$ to $[U|c]$ where U is upper triangular.
     Apply a sequence of row operations (**breakdown** possible).
  2. **Backward substitution**: If no breakdown happened, solve $Ux = c$.

Lower triangular matrices — Forward substitution

- Consider the system $Lx = b$ with lower triangular matrix

$$L = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix}.$$

- The algorithm goes as follows:

  1. $x_1 = b_1/l_{11}$
  2. $x_2 = (b_2 - l_{21}x_1)/l_{22}$

  $\vdots$

  i. $x_i = \left(b_i - \sum_{j=1}^{i-1} l_{ij}x_j\right)/l_{ii}$

  $\vdots$

  n. $x_n = \left(b_n - \sum_{j=1}^{n-1} l_{nj}x_j\right)/l_{nn}$

- Operation count: n divs. $+\ \dfrac{n(n-1)}{2}$ mults. $+\ \dfrac{n(n-1)}{2}$ adds. $= n^2$ ops.
- Breakdown happens iff $l_{ii} = 0$ for some $1 \leq i \leq n$, i.e., **iff** L is **singular**.

Upper triangular matrices — Backward substitution

- Consider the system $Ux = b$ with upper triangular matrix

$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix}.$$

- The algorithm goes as follows:

$$\begin{aligned}
n.\quad & x_n = b_n / u_{nn} \\
n-1.\quad & x_{n-1} = (b_{n-1} - u_{n-1,n} x_n) / u_{n-1,n-1} \\
& \vdots \\
i.\quad & x_i = \left(b_i - \textstyle\sum_{j=i+1}^{n} u_{ij} x_j \right) / u_{ii} \\
& \vdots \\
1.\quad & x_1 = \left(b_1 - \textstyle\sum_{j=2}^{n} u_{1j} x_j \right) / u_{11}
\end{aligned}$$

- Operation count: n divs. $+ \dfrac{n(n-1)}{2}$ mults. $+ \dfrac{n(n-1)}{2}$ adds. $= n^2$ ops.
- Breakdown happens iff $u_{ii} = 0$ for some $1 \leq i \leq n$, i.e., **iff** U is **singular**.

General matrices — Forward elimination

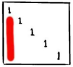
- Forward elimination is deployed to try and transform $[A|b]$ to $[U|c]$ where U is an upper triangular matrix. First, we do so without pivoting.
  - First, we want to operate a transformation of the form

$$\left[\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} & b_n \end{array}\right] \xrightarrow{(k=1)} \left[\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} & b_2^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} & \cdots & a_{3n}^{(1)} & b_3^{(1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a_{n2}^{(1)} & a_{n3}^{(1)} & \cdots & a_{nn}^{(1)} & b_n^{(1)} \end{array}\right]$$

where $a_{21}, \ldots, a_{n1}$ are eliminated by setting $b_i^{(1)} := b_i - m_i^{(1)} b_i$ and

$$a_{ij}^{(1)} := a_{ij} - m_i^{(1)} a_{1j} \text{ where } m_i^{(1)} := a_{i1}/a_{11} \text{ for } i, j \in \{2, \ldots, n\}.$$

This is equivalently done by $[A|b] \mapsto [G_1 A | G_1 b]$ where

$G_1 = I_n - v^{(1)} e_1^T$ is a **Gauss transformation** matrix with structure 

in which $v^{(1)} = [0 \;\; m_2^{(1)} \;\; \ldots \;\; m_n^{(1)}]^T$.

Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

General matrices — Forward elimination, cont'd$_1$

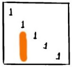
- Forward elimination is deployed to try and transform $[A|b]$ to $[U|c]$ where U is an upper triangular matrix. First, we do so without pivoting.
  - Then, we want to operate a transformation of the form

$$\left[\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} & b_2^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} & \cdots & a_{3n}^{(1)} & b_3^{(1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a_{n2}^{(1)} & a_{n3}^{(1)} & \cdots & a_{nn}^{(1)} & b_n^{(1)} \end{array}\right] \xrightarrow{(k=2)} \left[\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} & b_2^{(1)} \\ 0 & 0 & a_{33}^{(2)} & \cdots & a_{3n}^{(2)} & b_3^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & a_{n3}^{(2)} & \cdots & a_{nn}^{(2)} & b_n^{(2)} \end{array}\right]$$

  where $a_{32}^{(1)}, \ldots, a_{n2}^{(1)}$ are eliminated by setting $b_i^{(2)} := b_i^{(1)} - m_i^{(2)} b_i^{(1)}$ and

  $a_{ij}^{(2)} := a_{ij}^{(1)} - m_i^{(2)} a_{2j}^{(1)}$ where $m_i^{(2)} := a_{i2}^{(1)} / a_{22}^{(1)}$ for $i, j \in \{3, \ldots, n\}$.

  This is equivalently done by $[G_1 A | G_1 b] \mapsto [G_2 G_1 A | G_2 G_1 b]$ where

  $G_2 = I_n - v^{(2)} e_2^T$ is a **Gauss transformation** matrix with structure 

  in which $v^{(2)} = [0 \;\; 0 \;\; m_3^{(2)} \;\; \ldots \;\; m_n^{(2)}]^T$.

Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

General matrices — Forward elimination, cont'd$_2$

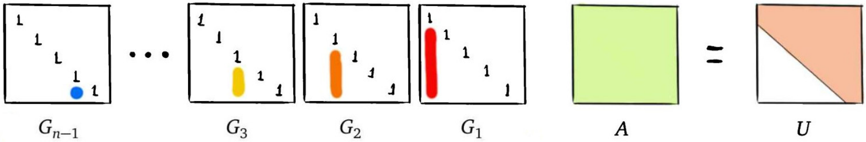
- Forward elimination is deployed to try and transform $[A|b]$ to $[U|c]$ where U is an upper triangular matrix. First, we do so without pivoting.
  - Eventually, the row-echelon form $[U|c]$ is obtained after the application of $n-1$ Gaussian transformations:

$$[G_{n-1}\dots G_1 A|G_{n-1}\dots G_1 b] = [U|c]$$

    where $G_k = I_n - v^{(k)}e_k^T$ in which

$$v_i^{(k)} = \begin{cases} 0 & \text{for } 1 \leq i \leq k \\ a_{ik}^{(k-1)}/a_{kk}^{(k-1)} & \text{for } k < i \leq n \end{cases} \quad \text{for } \ 1 < k \leq n-1.$$

  - Structurally speaking, U is formed as follows:



Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

  - **Breakdown** happens if either of the $a_{11}, a_{22}^{(1)}, \dots, a_{n-1,n-1}^{(n-2)}$ pivots is zero.

General matrices — Forward elimination, cont'd$_3$

- Operation count of $G_{n-1}\dots G_1 A$:

$$
\begin{aligned}
T_A(n) := \sum_{k=1}^{n-1} \big(n-k \text{ divs.} + (n-k)^2 \text{ mults.} + (n-k)^2 \text{ adds.}\big) \\
= \frac{(n-1)n}{2} \text{ divs.} + \frac{n(2n^2-3n+1)}{6} \text{ mults.} \\
+ \frac{n(2n^2-3n+1)}{6} \text{ adds.} \\
= \frac{n(4n^2-3n-1)}{6} \text{ ops.} = O(n^3) \text{ ops.}
\end{aligned}
$$

- Operation count of $G_{n-1}\dots G_1 b$:

$$
\begin{aligned}
T_b(n) := \sum_{k=1}^{n-1} (1 \text{ div.} + 1 \text{ mult.} + 1 \text{ add.}) \\
= (n-1) \text{ mults.} + (n-1) \text{ adds.} \\
= 2(n-1) \text{ ops.} = O(n) \text{ ops.}
\end{aligned}
$$

Tridiagonal matrices — Forward elimination

- Consider the system $Tx = b$ with tridiagonal matrix T. Then, assuming no breakdown happens, the forward elimination yields a bidiagonal matrix.
  - The first set of row operations, i.e., $k = 1$, yields

$$\left[\begin{array}{cccccc|c} t_{11} & t_{12} & & & & & b_1 \\ {\color{red}t_{21}} & {\color{blue}t_{22}} & t_{23} & & & & {\color{blue}b_2} \\ & t_{32} & t_{33} & t_{34} & & & b_3 \\ & & \ddots & \ddots & \ddots & & \vdots \\ & & & t_{n,n-1} & t_{nn} & & b_n \end{array}\right] \xrightarrow{(k=1)} \left[\begin{array}{cccccc|c} t_{11} & t_{12} & & & & & b_1 \\ {\color{red}0} & {\color{blue}t_{22}^{(1)}} & t_{23} & & & & {\color{blue}b_2^{(1)}} \\ & t_{32} & t_{33} & t_{34} & & & b_3 \\ & & \ddots & \ddots & \ddots & & \vdots \\ & & & t_{n,n-1} & t_{nn} & & b_n \end{array}\right]$$

The application of forward elimination starts by ${\color{blue}b_2^{(1)}} := b_2 - m_2^{(1)} b_2$ and

$$t_{2j}^{(1)} := t_{2j} - m_2^{(1)} t_{1j} \text{ where } m_2^{(1)} := t_{21}/t_{11} \text{ for } j \in \{2, \dots, n\}.$$

Since $t_{13} = \cdots = t_{1n} = 0$, we have

$${\color{blue}t_{22}^{(1)}} = t_{22} - m_2^{(1)} t_{12}, \quad \text{but} \quad t_{2j}^{(1)} = t_{2j} \text{ for } j \in \{3, \dots, n\}.$$

Tridiagonal matrices — Forward elimination, cont'd

- Consider the system $Tx = b$ with tridiagonal matrix T. Then, assuming no breakdown happens, the forward elimination yields a bidiagonal matrix.
  - Then, the row operations for $k = 2$ yield

$$\left[\begin{array}{ccccc|c} t_{11} & t_{12} & & & & b_1 \\ & t_{22}^{(1)} & t_{23} & & & b_2^{(1)} \\ & {\color{red}t_{32}} & {\color{blue}t_{33}} & t_{34} & & {\color{blue}b_3} \\ & & \ddots & \ddots & \ddots & \vdots \\ & & & t_{n,n-1} & t_{nn} & b_n \end{array}\right] \xrightarrow{(k=2)} \left[\begin{array}{ccccc|c} t_{11} & t_{12} & & & & b_1 \\ & t_{22}^{(1)} & t_{23} & & & b_2^{(1)} \\ & {\color{red}0} & {\color{blue}t_{33}^{(1)}} & t_{34} & & {\color{blue}b_3^{(2)}} \\ & & \ddots & \ddots & \ddots & \vdots \\ & & & t_{n,n-1} & t_{nn} & b_n \end{array}\right]$$

  Similarly, since $t_{24} = \cdots = t_{2n} = 0$, we have

$$ {\color{blue}t_{33}^{(1)}} = t_{33} - m_3^{(2)} t_{23}, \quad \text{but} \quad t_{3j}^{(1)} = t_{3j} \text{ for } j \in \{4, \dots, n\}.$$

  - And so on for $k = 3, \dots, n-1$.
- Operation count: $T_T(n) = 3(n-1)$ ops. and $T_b(n) = (n-1)$ ops.
- **Breakdown** happens iff $t_{ii} = 0$ for some $1 \leq i < n$.

Bidiagonal matrices — Simplified backward substitution

- Consider the system $Bx = b$ with (upper) bidiagonal matrix

$$B = \begin{bmatrix} b_{11} & b_{12} & & \\ & b_{22} & b_{23} & \\ & & \ddots & \ddots \\ & & & b_{nn} \end{bmatrix}.$$

- The algorithm goes as follows:

$$\begin{aligned}
n.\quad & x_n = b_n / b_{nn} \\
n-1.\quad & x_{n-1} = (b_{n-1} - b_{n-1,n} x_n) / b_{n-1,n-1} \\
& \vdots \\
i.\quad & x_i = \left(b_i - b_{i,i+1} x_{i+1}\right) / b_{ii} \\
& \vdots \\
1.\quad & x_1 = \left(b_1 - b_{12} x_2\right) / b_{11}
\end{aligned}$$

- Operation count: n divs. $+ (n-1)$ mults. $+ (n-1)$ adds. $= 3n - 2$ ops.
- Breakdown happens iff $b_{ii} = 0$ for some $1 \leq i \leq n$, i.e., **iff** B is **singular**.

LU factorization without pivoting

Section 3.1 in Darve & Wootters (2021)

LU factorization

- So far, we considered forward elimination without pivoting. **If no breakdown happens**, this process yields an **upper-triangular** matrix

$$U = G_{n-1} \cdots G_1 A$$

  where the Gauss transformation matrix $G_k = I_n - v^{(k)} e_k^T$ is lower-triangular, with ones on the diagonal, thus non-singular.
- You can verify that $G_k^{-1} = I_n + v^{(k)} e_k^T$.
- Given the structure of $v^{(k)} = [0 \ \cdots \ 0 \ m_{k+1}^{(k)} \ \ldots \ m_n^{(k)}]^T$, we also have that $k < \ell$ implies $G_k^{-1} G_\ell^{-1} = I_n + v^{(k)} e_k^T + v^{(\ell)} e_\ell^T$.
- Consequently, we have

$$\begin{aligned} G_1^{-1} \cdots G_{n-1}^{-1} U &= A \\ \left(I_n + v^{(1)} e_1^T + \cdots + v^{(n-1)} e_{n-1}^T \right) U &= A \\ LU &= A \end{aligned}$$

  where $L := G_1^{-1} \cdots G_{n-1}^{-1}$ is **lower-triangular**.

LU factorization, cont'd

- The components below the diagonal of the k-th column of L are given by the non-zero components of $v^{(k)}$, i.e.,

$$L = \begin{bmatrix} 1 & & & \\ m_2^{(1)} & \ddots & & \\ \vdots & & 1 & \\ m_n^{(1)} & \dots & m_n^{(n-1)} & 1 \end{bmatrix}$$

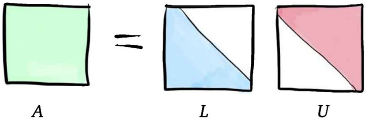
  so that L is a **by-product** of the **forward elimination** procedure, i.e., we have

$$m_i^{(k)} = \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}$$

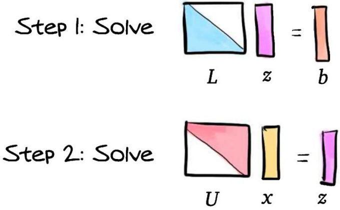
  where $a_{ij}^{(k-1)}$ are components of $A^{(k-1)} := G_{k-1} \cdots G_1 A$, and $a_{ij}^{(0)} := a_{ij}$.
- If A is non-singular, and the upper-triangular matrix U is obtained by forward elimination without breakdown, then it can be shown that there is a unique lower-triangular matrix L such that $LU = A$.

Solving linear systems with an LU factorization

- Given an LU factorization of an invertible matrix A:



the linear system $Ax = b$ can be recast into $Lz = b$, where $Ux = z$, so that one can solve for x in two steps:



- Then, solving $Ax = b$ requires two triangular solves, i.e., a forward substitution, followed by a backward propagation, totaling $2n^2$ operations.

Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

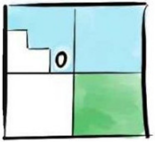
Breakdown and instability of LU factorization

Section 3.1 and 3.3 in Darve & Wootters (2021)

Breakdown of LU factorization without pivoting

- So far, we assumed **no breakdown** happens during forward elimination.
- However, **breakdowns** do happen, **even** when using **exact arithmetic** and A is **invertible**:

  E.g., applying forward elimination to $A := \begin{bmatrix} 1 & 6 & 1 & 0 \\ 0 & 1 & 9 & 0 \\ 1 & 6 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$, which **is invertible**, will **break down**.
- If $a_{kk}^{(k-1)} = 0$, then breakdown will happen when applying G_k to $A^{(k-1)}$:

  We say that $A^{(k-1)} =$  has a **zero-pivot**.

Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

In particular, breakdown happens as we attempt to **divide by zero** to form

$$m_i^{(k)} := a_{ik}^{(k-1)} / a_{kk}^{(k-1)} \text{ for } i = k+1, \dots, n.$$

Understanding the source of breakdown

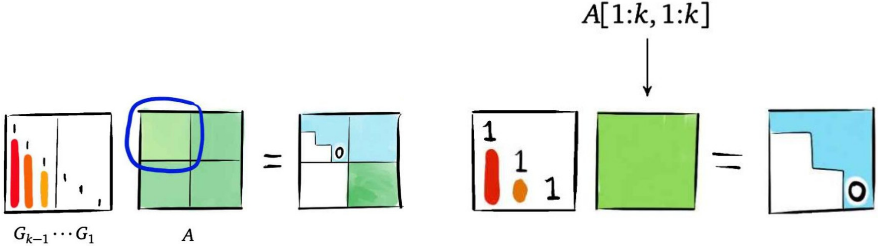
- We can think of the block $A^{(k-1)}[1:k,1:k]$ as

$$(G_{k-1}\dots G_1)[1:k,1:n]A[1:n,1:k] = A^{(k-1)}[1:k,1:k].$$

But since $(G_{k-1}\dots G_1)[1:k,k+1:n] = 0$, we have

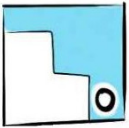
$$(G_{k-1}\dots G_1)[1:k,1:k]A[1:k,1:k] = A^{(k-1)}[1:k,1:k].$$

Thus, we can focus our investigation on the leading principal blocks:

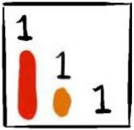


Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

Understanding the source of breakdown, cont'd

- 

  The leading block $A^{(k-1)}[1:k,1:k]$ is **singular** because it is **triangular** with a **zero on the diagonal**, i.e., $a_{kk}^{(k-1)} = 0$.

- 

  The leading block $(G_{k-1}\dots G_1)[1:k,1:k]$ of the product of Gauss transformation matrices is **non-singular** because it is **triangular** with a **ones on the diagonal**.

- 

  Therefore, the leading block $A[1:k,1:k]$ must be **singular**.

Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

Theorem (Existence of an LU factorization without pivoting)

A matrix $A \in \mathbb{F}^{n\times n}$ admits an LU factorization without pivoting iff its $n-1$ leading principal sub-matrices are non-singular.

Backward error of LU factorization without pivoting

- Consider a matrix A whose leading principal sub-matrices are non-singular, and let $\tilde{L}$ and $\tilde{U}$ be **approximations** of the factors L and U of A.
- **Backward error analysis** considers that $\tilde{L}$ and $\tilde{U}$ are **exact factors of a perturbed matrix**, i.e., there exists δA such that

$$A + \delta A = \tilde{L}\tilde{U}.$$

  The analysis consists then of bounding this perturbation.
- In Lecture 03, we introduced backward error analysis in a way that is agnostic to the algorithm. For the LU factorization, this is not the case:
  - $\tilde{L}$ and $\tilde{U}$ are specifically assumed to be computed by **forward elimination** with **floating-point arithmetic**.
  - Then, the perturbation δA is bounded **component-wise** by

$$|\delta A| \leq \gamma_n \cdot |\tilde{L}||\tilde{U}|$$

    where $\gamma_n := nu/(1 - nu)$ and $nu < 1$, in which u is the unit roundoff.
  - In general, the components of $|\tilde{L}||\tilde{U}|$ can take **arbitrary large values**, i.e.,

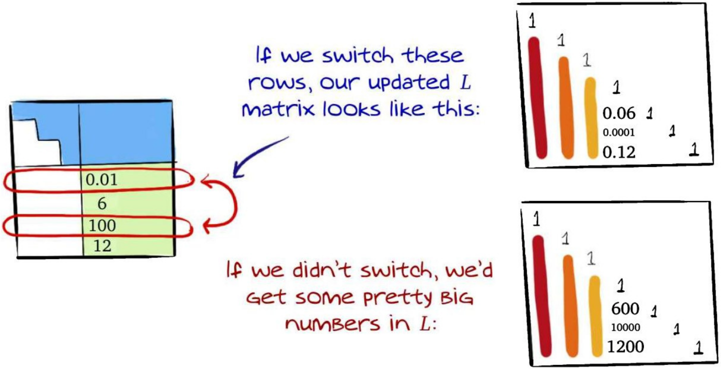
forward elimination without pivoting is not backward stable.

LU factorization with pivoting

Section 3.4 in Darve & Wootters (2021)

Row pivoting

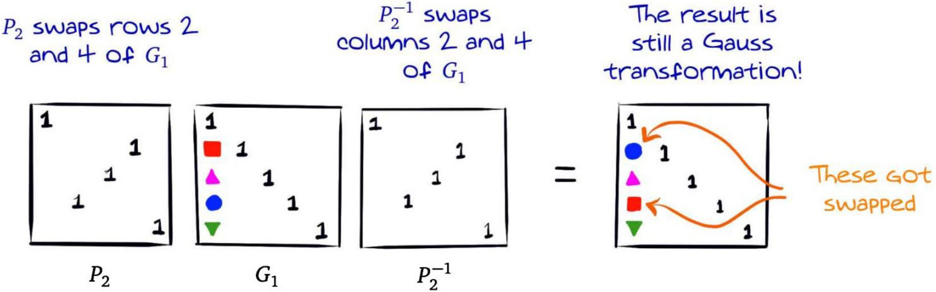
- For a given matrix A, one way to reduce the backward error of an approximate LU factorization is to contain the components of $|\tilde{L}|$.
- Since $L[i,k] = m_i^{(k)} := a_{ik}^{(k-1)}/a_{kk}^{(k-1)}$ for $i = k+1, \ldots, n$, this can be done if we allow ourselves to permute the rows of $A^{(k-1)}[k{+}1, 1{:}n]$ such that $a_{kk}^{(k-1)} \geq a_{ik}^{(k-1)}$ for $i = k+1, \ldots, n$. Then we would have $|L| \leq 1$.



Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

Row pivoting, cont'd$_1$

- When row pivoting is introduced in forward elimination, it is expressed as $G_{n-1}P_{n-1}\cdots G_1P_1A$, where $P_1,\ldots,P_k$ denote row swap permutations.
- The similarity transformation PG_kP^{-1} of a Gauss transformation matrix G_k with pivot column k using a permutation matrix P, is another Gauss transformation matrix $\widetilde{G}_k$ with pivot column k, e.g.,



Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

- Then, as we let $\widetilde{G}_k := P_{n-1}\cdots P_{k+1}G_kP_{k+1}^{-1}\cdots P_{n-1}^{-1}$, you can show that

$$G_{n-1}P_{n-1}\cdots G_1P_1A = \widetilde{G}_{n-1}\cdots\widetilde{G}_1P_{n-1}\cdots P_1A =: U.$$

Row pivoting, cont'd$_2$

- Similarly as without pivoting, this can be recast as

$$\begin{aligned}\widetilde{G}_{n-1}\cdots\widetilde{G}_1 P_{n-1}\cdots P_1 A &= U\\ P_{n-1}\cdots P_1 A &= \widetilde{G}_1^{-1}\cdots\widetilde{G}_{n-1}^{-1}U\\ PA &= LU\end{aligned}$$

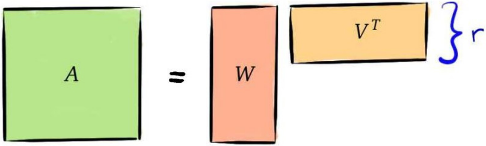
  where $L=\widetilde{G}_1^{-1}\cdots\widetilde{G}_{n-1}^{-1}$ and $P=P_{n-1}\cdots P_1$.
- That is, there is a permutation P such that an LU factorization of PA exists, and can be obtained by forward elimination without pivoting.
- Upon applying row pivoting during forward elimination, such a permutation P is recovered along with the triangular factors L and U such that $PA=LU$.

  Then, one can solve for x such that $Ax=b$ by

  1. Solving for z such that $Lz=Pb$
  2. Solving for x such that $Ux=z$.

Material we skip, for now

- **Column pivoting** (p. 101 in Darve and Wootters (2021))
  - Column pivoting is introduced to allow for the computation of rank revealing factorizations:



Darve, E., & Wootters, M. (2021). Numerical linear algebra with Julia. Society for Industrial and Applied Mathematics.

- **Full pivoting** (p. 102 in Darve and Wootters (2021))
  - Performing both row and column swaps allows for the computation of rank revealing factorization while maintaining stability.
- **Rook pivoting** (p. 103 in Darve and Wootters (2021))
  - Reduces the cost of full pivoting by simplifying the search for swaps.
- **Pivots and singular values** (p. 104 in Darve and Wootters (2021))
  - Pivoting strategies can also be used to compute approximately optimal low-rank matrix approximations.

Cholesky factorization

Section 3.5 in Darve & Wootters (2021)

Cholesky factorization

- LU factorization is intended for general square matrices. For Hermitian positive-definite matrices, it is possible to leverage the properties of such matrices to yield a better behaved factorization.

Theorem (Cholesky factorization)

- *If $A \in \mathbb{F}^{n\times n}$ is Hermitian positive-definite, then there exists a lower-triangular matrix $L \in \mathbb{F}^{n\times n}$ such that $A = LL^H$.*
- *If we limit our search to lower-triangular matrices with positive components on the diagonal, then L is unique.*

- The existence of such factors L is proven by inductive construction. In particular, A being Hermitian, if L exists, we must have

$$A = \begin{bmatrix} a_{11} & a_1^H \\ a_1 & A_1 \end{bmatrix} = \begin{bmatrix} l_{11} & 0 \\ l_1 & L_1 \end{bmatrix} \begin{bmatrix} l_{11} & l_1^H \\ 0 & L_1^H \end{bmatrix} \text{ where } L = \begin{bmatrix} l_{11} & 0 \\ l_1 & L_1 \end{bmatrix}$$

where, due to positive definiteness, $a_{11} > 0$, and the principal block A_1 is Hermitian positive-definite.

Cholesky factorization, cont'd$_1$

This is recast into $A = \begin{bmatrix} a_{11} & a_1^H \\ a_1 & A_1 \end{bmatrix} = \begin{bmatrix} l_{11}^2 & l_{11} l_1^H \\ l_{11} l_1 & L_1 L_1^H + l_1 l_1^H \end{bmatrix}$.

By construction, we impose $l_{11} > 0$, so that we have

$$l_{11} = \sqrt{a_{11}} \text{ and } l_1 = a_1 / l_{11}.$$

We rely here on the assumption that the Cholesky factorization $L_1 L_1^H = A_1 - l_1 l_1^H$ exists. To show that, A can be recast into XBX^H

$$A = \begin{bmatrix} a_{11} & a_1^H \\ a_1 & A_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ l_1 / l_{11} & I_{n-1} \end{bmatrix} \begin{bmatrix} l_{11}^2 & 0 \\ 0 & A_1 - l_1 l_1^H \end{bmatrix} \begin{bmatrix} 1 & l_1^H / l_{11} \\ 0 & I_{n-1} \end{bmatrix}$$

where $X = \begin{bmatrix} 1 & 0 \\ l_1 / l_{11} & I_{n-1} \end{bmatrix}$ and $B = \begin{bmatrix} l_{11}^2 & 0 \\ 0 & A_1 - l_1 l_1^H \end{bmatrix}$.

Since A is Hermitian positive-definite, and X is non-singular, then B must be positive-definite.

Moreover, since the principal sub-matrices of a Hermitian positive-definite matrix are positive-definite, so is $A_1 - l_1 l_1^H$.

Cholesky factorization, cont'd$_2$

- To complete the construction of L, we assume that the l_{ij} components of L are known for $i=1,\dots,k$ and $j=1,\dots,i$, s.t. $l_{11},\dots,l_k>0$ and

$$A=\begin{bmatrix} a_{11} & \dots & \overline{a_{k1}} & a_1^H \\ \vdots & \ddots & \vdots & \vdots \\ a_{k1} & \dots & a_{kk} & a_k^H \\ a_1 & \dots & a_k & A_k \end{bmatrix} = \begin{bmatrix} l_{11} & & & \\ \vdots & \ddots & & \\ l_{k1} & \dots & l_{kk} & \\ l_1 & \dots & l_k & L_k \end{bmatrix} \begin{bmatrix} l_{11} & \dots & \overline{l_{k1}} & l_1^H \\ & \ddots & \vdots & \\ & & l_{kk} & l_k^H \\ & & & L_k^H \end{bmatrix}$$

  where $A_k - l_k l_k^H - \dots - l_1 l_1^H$ is Hermitian positive-definite with Cholesky factorization $L_k L_k^H$.
- The construction of L is completed by showing that L_{k+1} can be defined under similar conditions.
- The uniqueness of L is revealed with the final requirement $|L_n|^2=a_{nn}$. Since both a_{nn} and L_n need be strictly positive, we simply have $L_n=a_{nn}$. □

Computation of the Cholesky factorization

- The procedure to compute a Cholesky factor follows the lines of our constructive proof.
  It requires about half the number of operations than that of calculating an LU factorization by forward elimination.
- **Backward error analysis** considers that $\tilde{L}$ is an **exact factor of a perturbed matrix**, i.e., there exists δA such that

$$A + \delta A = \tilde{L}\tilde{L}^H.$$

  The analysis consists then of bounding this perturbation.
  - Such analyses assume $\tilde{L}$ is specifically computed using the procedure we described, with **floating-point arithmetic**.
  - Then, the perturbation δA is bounded **component-wise** by

$$|\delta A| \leq \frac{\gamma_{n+1}}{1-\gamma_{n+1}} \cdot dd^T \;\text{ where }\; d = [a_{11}^{1/2} \;\ldots\; a_{nn}^{1/2}]^T$$

    with $\gamma_n := nu/(1-nu)$ and $nu < 1$, in which u is the unit roundoff.
  - Therefore, computing the Cholesky factorization is a **backward stable** procedure, and thus, it **does not require pivoting**.

Homework problems

Homework problems

Turn in **your own** solution to **Pb. 12**:

Pb. 11 Show that, if a leading principal sub-matrix of A, i.e., $A[1:k, 1:k]$ with k such that $1 \leq k < n$, is singular, then Doolittle's forward elimination procedure, if applied to A without pivoting, will break down. Explain when precisely and how the breakdown will happen.

Pb. 12 Answer the following questions, and provide proper explanations:

a. Are the principal sub-matrices of a Hermitian positive-definite (HPD) matrix also HPD?

b. Let $A = XBX^H$ be HPD and X be non-singular. Is B also HPD?

c. Are the principal sub-matrices of a non-singular matrix also non-singular?

Pb. 13 Consider the symmetric positive-definite matrix given by $A = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$, with eigenpairs $\left(4, [1\ 1]^T\right)$ and $\left(2, [-1\ 1]^T\right)$:

a. Construct, with a pen and paper, the Cholesky factor L with positive diagonal components such that $A = LL^T$.

b. Form the square root $A^{1/2}$ of A.

Practice session

Practice session

1. Write a function called `RowMajorForwardSubstitution` that implements forward substitution as described in slide #3.
2. Write a function called `ColumnMajorForwardSubstitution` that implements forward substitution fetching components of the lower triangular matrix in a column-wise fashion.
3. Compare the runtime of `RowMajorForwardSubstitution` and `ColumnMajorForwardSubstitution` for different matrix sizes.
4. Write a function called `get_LU` that returns the L and U factors of a matrix A obtained by forward elimination without pivoting. Test your code by solving a linear system using the L and U factors.
5. Write a function called `RowMajor_LU_InPlace!` that computes the L and U factors of A, in-place, by forward elimination without pivoting. Verify your code.
6. Write a function called `ColumnMajor_LU_InPlace!` that does in-place LU factorization without pivoting, with a column-wise data access pattern.
7. Compare the runtime of `RowMajor_LU_InPlace!` and `ColumnMajor_LU_InPlace!` for different matrix sizes.