

MA5233 Computational Mathematics

Lecture 3: LU Factorisation

Simon Etter



2019/2020

LU Factorisation

LU factorisation

Algorithm of choice for solving dense linear systems.

Outline

- ▶ LU factorisation: why and how.
- ▶ `lu()` in Julia.
- ▶ Computational complexity of LU factorisation.
- ▶ Conditioning and stability of LU factorisation.

LU Factorisation

Linear system of equations

Given $A \in \mathbb{K}^{n \times n}$ and $b \in \mathbb{K}^n$, find $x \in \mathbb{K}^n$ such that $Ax = b$.

Observation

Problem is easy if A is triangular, i.e.

$$A(i,j) = 0 \quad \text{for} \quad \begin{cases} i > j & \text{(upper triangular),} \\ i < j & \text{(lower triangular).} \end{cases}$$

LU Factorisation

Example

$$\begin{pmatrix} 4 & 1 & -2 \\ 0 & 2 & -1 \\ 0 & 0 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 4 \\ 6 \end{pmatrix}$$

Third eq.: $3x_3 = 6 \implies x_3 = 2$

Second eq.: $2x_2 - x_3 = 4 \implies x_2 = 3$

First eq.: $4x_1 + x_2 - 2x_3 = 3 \implies x_1 = 1$

LU Factorisation

LU factorisation theorem

For every invertible matrix $A \in \mathbb{K}^{n \times n}$, there exist

- ▶ a permutation matrix $P \in \mathbb{K}^{n \times n}$,
- ▶ a lower-triangular matrix $L \in \mathbb{K}^{n \times n}$ with unit diagonal, and
- ▶ an upper-triangular matrix $U \in \mathbb{K}^{n \times n}$

such that $PA = LU$.

L , U are unique for fixed P .

Why is there a P in this theorem?

Because of cases like this one:

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$$

LU Factorisation

Example

$$\begin{pmatrix} 4 & 1 & -2 \\ -8 & 0 & 3 \\ 12 & 7 & -5 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & 0 & 1 \end{pmatrix} \begin{pmatrix} 4 & 1 & -2 \\ 0 & 2 & -1 \\ 0 & 4 & 1 \end{pmatrix}$$
$$= \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & 2 & 1 \end{pmatrix} \begin{pmatrix} 4 & 1 & -2 \\ 0 & 2 & -1 \\ 0 & 0 & 3 \end{pmatrix}$$

LU Factorisation

Column/partial pivoting

Use row with largest entry in first column to eliminate the other rows.

Example

$$\begin{pmatrix} 4 & 1 & -2 \\ -8 & 0 & 3 \\ \textcolor{red}{12} & 7 & -5 \end{pmatrix} \rightarrow \begin{pmatrix} \textcolor{red}{12} & 7 & -5 \\ -8 & 0 & 3 \\ 4 & 1 & -2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{2}{3} & 1 & 0 \\ \frac{1}{3} & 0 & 1 \end{pmatrix} \begin{pmatrix} \textcolor{red}{12} & 7 & -5 \\ 0 & x & x \\ 0 & x & x \end{pmatrix}$$

LU Factorisation

Complete pivoting

Use largest overall entry (i,j) to eliminate the other entries in column j .

Example

$$\begin{pmatrix} 1 & 4 & -2 \\ 0 & -8 & 3 \\ 7 & 12 & -5 \end{pmatrix} \rightarrow \begin{pmatrix} 12 & 7 & -5 \\ -8 & 0 & 3 \\ 4 & 1 & -2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{2}{3} & 1 & 0 \\ \frac{1}{3} & 0 & 1 \end{pmatrix} \begin{pmatrix} 12 & 7 & -5 \\ 0 & x & x \\ 0 & x & x \end{pmatrix}$$

LU Factorisation

Permutations

Bijjective map $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$.

Example

$$\pi(1) = 2, \quad \pi(2) = 4, \quad \pi(3) = 3, \quad \pi(4) = 1$$

Representations

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \quad \text{or} \quad p = \begin{pmatrix} 2 & 4 & 3 & 1 \end{pmatrix}^T$$

LU Factorisation

Remarks on permutations

- ▶ PA permutes rows of A .
- ▶ Matrix representation is convenient, but very inefficient.
Never use permutation matrices in your code!
- ▶ Applying permutation p in Julia:

```
data = ["a", "b", "c", "d"]  
p = [2, 4, 3, 1]  
data[p] -> ["b", "d", "c", "a"]
```
- ▶ Permutations are by definition bidirectional.
Be careful which direction you represent in your code!

LU Factorisation

Solving $Ax = b$ via LU factorisation

- ▶ Compute LU factorisation $PA = LU$.
- ▶ Permute the RHS: $\hat{b} = Pb$.
- ▶ Solve $y = L^{-1}\hat{b}$.
- ▶ Solve $x = U^{-1}y$.

LU Factorisation

Solving $Ax = b$ in Julia

`F = lu(A, pivot=Val{true})` computes “factorisation object”.

- ▶ Access factors through `F.L`, `F.U`, `F.p` (vector) and `F.P` (matrix).
- ▶ `F.L * F.U == A[F.p, :]` == `F.P * A`.
- ▶ `x = F\b` computes solution.
- ▶ `A\b` solves $A*x = b$ directly.

LU Factorisation

Measuring the “required effort” of an algorithm

Some ideas:

- ▶ Count the number of $+$, $-$, \times , $/$, sqrt .
Very tedious, and makes it hard to compare algorithms.
- ▶ Measure its runtime.
Too dependent on input, hardware, etc.

Most common measure: big- \mathcal{O} estimate.

Examples

- ▶ Evaluating $x^T y := \sum_{k=1}^n x_k y_k$ takes
 - ▶ n multiplications, and
 - ▶ $n - 1$ additions.

Computing inner products takes $\mathcal{O}(n)$ floating-point operations.

- ▶ Ax for $A \in \mathbb{K}^{n \times n}$ can be computed as n inner products.
Evaluating matvec takes $\mathcal{O}(n^2)$ floating-point operations.

LU Factorisation

Why is big- \mathcal{O} notation useful?

It tells us the functional dependency of runtime on problem size.

$\mathcal{O}(n)$ algorithm \implies Changing $n \rightarrow 2n$ multiplies runtime by 2.

$\mathcal{O}(n^2)$ algorithm \implies Changing $n \rightarrow 2n$ multiplies runtime by 4.

...

Computational cost of LU factorisation

- ▶ Factorisation: $\mathcal{O}(n^3)$.
- ▶ Triangular solves: $\mathcal{O}(n^2)$.

Hence, reuse factorisation if possible.

LU Factorisation

Conditioning of linear systems

Assume

- ▶ $Ax = b$, and
- ▶ $(A + \Delta A)(x + \Delta x) = b + \Delta b$ with $\|\Delta A\| < \|A^{-1}\|^{-1}$.

Then,

$$\begin{aligned}\frac{\|\Delta x\|}{\|x\|} &\leq \frac{\kappa(A)}{1 - \kappa(A) \frac{\|\Delta A\|}{\|A\|}} \left(\frac{\|\Delta A\|}{\|A\|} + \frac{\|\Delta b\|}{\|b\|} \right) \\ &\approx \kappa(A) \left(\frac{\|\Delta A\|}{\|A\|} + \frac{\|\Delta b\|}{\|b\|} \right) + \mathcal{O}(\kappa(A)^2).\end{aligned}$$

Statement to remember

Relative error in x is at most $\kappa(A)$ times relative errors in A and b up to higher-order terms.

LU Factorisation

Proof. Solving the perturbed equation for Δx yields

$$\begin{aligned}\Delta x &= (A + \Delta A)^{-1} (b + \Delta b - (A + \Delta A)x) \\ &= A^{-1} (I + A^{-1} \Delta A)^{-1} (\Delta b - \Delta A x).\end{aligned}$$

We obtain using

- ▶ the Neumann estimate¹ $\|(1 + M)^{-1}\| \leq \frac{1}{1 - \|M\|}$ for $\|M\| < 1$,
- ▶ $\|b\| = \|Ax\| \leq \|A\| \|x\|$, and
- ▶ the monotonicity of $\frac{1}{x}$ and $\frac{1}{1-x}$

that

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\|A\| \|A^{-1}\|}{1 - \|A^{-1}\| \|\Delta A\|} \left(\frac{\|\Delta b\|}{\|b\|} + \frac{\|\Delta A\|}{\|A\|} \right).$$

¹ https://en.wikipedia.org/wiki/Neumann_series

LU Factorisation

Stability of solving linear systems via LU factorisation

Numerical solution \tilde{x} to $Ax = b$ computed via LU factorisation satisfies

$$(A + \Delta A) \tilde{x} = b \quad \text{where} \quad \frac{\|\Delta A\|}{\|L\| \|U\|} \approx \mathcal{O}(\varepsilon_{\text{mach}}).$$

Combined with condition number for linear systems, this yields

$$\frac{\|\tilde{x} - x\|}{\|x\|} \approx \kappa(A) \frac{\|L\| \|U\|}{\|A\|} \mathcal{O}(\varepsilon_{\text{mach}}).$$

Error is small if

- ▶ $\kappa(A)$ is not too large (problem is well-conditioned), and
- ▶ $\frac{\|L\| \|U\|}{\|A\|}$ is not too large (LU factorisation is stable).

LU Factorisation

Impact of pivoting

No pivoting: $\|L\|, \|U\| = \infty$ is possible.

- ▶ We will see special matrices which do not require pivoting.
- ▶ Do not use this algorithm unless you know what you are doing.

Partial pivoting: $\|L\|, \|U\| \leq 2^{n-1}$ is a sharp upper bound.

- ▶ However, exponential growth of $\|L\|, \|U\|$ has never been observed in practice.
- ▶ Famous quote: "Anyone that unlucky has already been run over by a bus."
- ▶ This is the recommended algorithm in most applications.

Complete pivoting: probably $\|L\|, \|U\| = \mathcal{O}(n)$.

- ▶ No one uses this algorithm.

LU Factorisation

References and further reading

- ▶ G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press (1996),
- ▶ L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics (1997),
- ▶ J. W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics (1997),
doi:10.1137/1.9781611971446
- ▶ N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics (2002),
doi:10.1137/1.9780898718027