Square Linear Systems

Spring 2021, A. DONEV

The majority of numerical computing is about solving systems of m linear equations in n variables or unknowns:

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \qquad i = 1, \dots, m$$

e.g.

$$\begin{cases} 3x_1 + 2x_2 = 2 \\ x_1 - x_2 + x_3 = 1 \\ 2x_1 + 3x_3 = 5 \end{cases}$$

$$\overleftrightarrow{A}\,\vec{x} = \vec{b}$$

e.g. $A = \left[\begin{array}{c|c|c} 3 & 2 & 0 \\ \hline 1 & -1 & 1 \\ \hline 2 & 0 & 3 \end{array}\right]$ $b = \begin{bmatrix} 2 \\ 1 \\ 5 \end{bmatrix}$

Here we focus on square linear systems $m = n$.

If A is invertible, there is a unique solution

$$x = A^{-1} b$$

but this is NOT how we compute x numerically.

Q: What if A is not invertible? How many solutions are there

Standard approach is to use

Gaussian elimination

Step 1: Eliminate x_1

$A^{(1)} = [3 \times 3] = A$

Denote

$$\left[\begin{array}{c|cc} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ \hline a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & a_{33}^{(1)} \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array}\right] = \left[\begin{array}{c} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{array}\right]$$

step as subscript

To eliminate x_1 from 2nd eq, multiply first row by

$$\ell_{21} = \frac{a_{21}^{(1)}}{a_{11}^{(1)}}$$

and subtract from 2nd

To eliminate x_1 from 3rd eq, multiply first row by

$$l_{31} = \frac{a_{31}^{(1)}}{a_{11}^{(1)}}$$

eq # ↑ variable #

$$\left[\begin{array}{c|c|c} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ \hline a_{21}^{(1)} - l_{21} a_{11} = 0 & a_{22}^{(2)} = a_{22}^{(1)} - l_{21} \cdot a_{12}^{(1)} & \cdots \\ \hline 0 & a_{32}^{(2)} = a_{32}^{(1)} - l_{31} \cdot a_{12}^{(1)} & \cdots \end{array}\right]$$

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik} \cdot a_{kj}^{(k)}$$

will be the general k^{th} step

Where $\boxed{l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}}$

(i: eq, k: var)

$$\boxed{a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} a_{kj}^{(k)}}$$

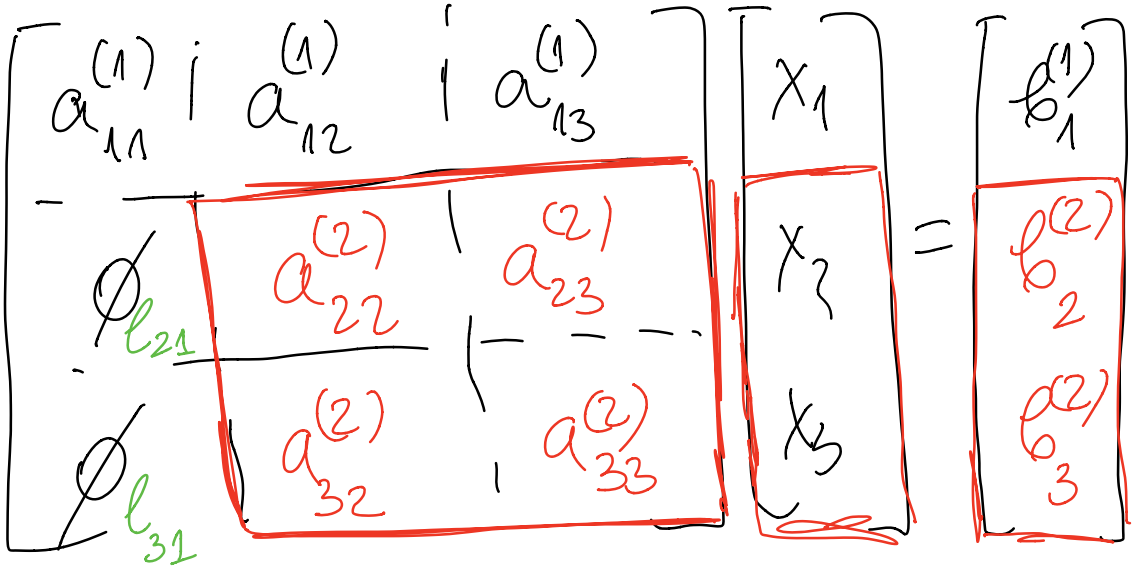
$\boxed{i, j > k}$ $\quad a_{kk}^{k} \neq 0$

We also do this for the right-hand side (r.h.s)

$$b^{(2)} = \begin{bmatrix} b_1^{(1)} = b_1 \\ \text{---------} \\ b_2^{(1)} - l_{21} b_1 \\ \text{---------} \\ b_3^{(1)} - l_{31} b_1 \end{bmatrix}$$

$$b_i^{(k+1)} = b_i^{(k)} - l_{ik} b_k^{(k)}$$

Step 2 : Eliminate x_2 from all subsequent equations

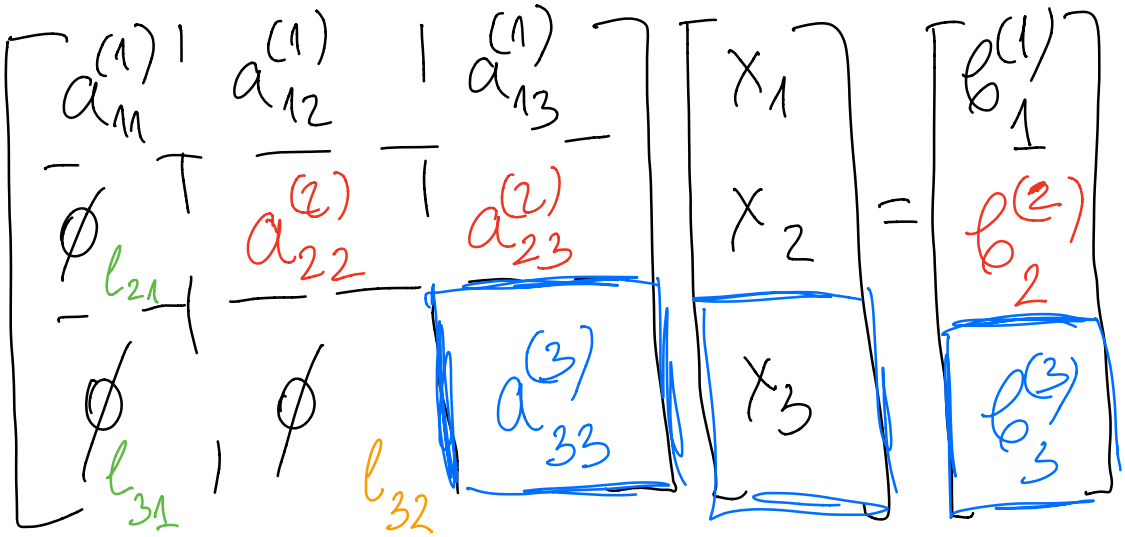


Now we have a 2x2 system to solve - no more x_1!

Multiply second row (red part only) by

$$l_{32} = \frac{a_{32}^{(2)}}{a_{22}^{(2)}}$$

and subtract from 3rd eq. ⑥



Now there is only one variable left!

$$x_3 = \frac{b_3^{(3)}}{a_{33}^{(3)}}$$

So we now have only two unknowns left, x_1 & x_2.

So plug x_3 into 2nd equation $a_{22}^{(2)} x_2 + a_{23}^{(2)} x_3 = b_2^{(2)}$

$$\Rightarrow \quad a_{22}^{(2)} x_2 = b_2^{(2)} - a_{23}^{(2)} x_3$$

and now solve for x_2

(remember $a_{22}^{(2)} \neq 0$ by assumption)

and then repeat process one more time to get x_1.

Define the unit lower triangular matrix

$$L = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix}$$

and the upper triangular matrix

$$U = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & 0 & a_{33}^{(3)} \end{bmatrix}$$

Claim / theorem:

$$\boxed{A = L\,U}$$

LU factorization of A

So instead of thinking about solving a specific equation (specific rhs b), think of factorizing A

Numerical linear algebra is a study of matrix factorizations

How do we show/prove this?

Recall: $l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \Rightarrow a_{ik}^{(k)}$

$$\begin{cases} a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik} \cdot a_{kj}^{(k)} \\ b_i^{(k+1)} = b_i^{(k)} - l_{ik} b_k \end{cases}$$

Assume here that it is OK to overwrite A and b as we do this, so use

lower triangle of A to store elements of L (called "in-place factorization"

MATLAB code : MyLU.m (on webpage)

for Gaussian elimination or LU factorization

for $k = 1 : (n-1)$ [eliminate x_k]

$A((k+1):n, k) = A((k+1):n, k) / A(k,k)$;

$i > k$: $l_{ik} = \frac{a_{ik}}{a_{kk}} \Rightarrow a_{ik}$

Remember $l_{kk} = 1$ (not stored)

for $j = k+1 : n$

$A((k+1):n, j) = A((k+1):n, j)$

$- A((k+1):n, k) * A(k, j)$;

$a_{ij} <= a_{ij} - l_{ik} a_{kj}$ $[i,j > k]$

Note: Equivalent code is

for $i = k+1 : n$

$A(i, (k+1):n) = A(i, (k+1):n)$

$A(i,k) * A(k, (k+1):n)$

which is closer to "Practice" textbook of Greenbaum

Actual code used by MATLAB is very different but does the same.

"Proof" that $A = LU$

$$a_{ij} = \sum_{k=1}^{n \text{ or } \min\{i,j\}} l_{ik} u_{kj}$$

(k and j contracted)

Assume $1 \leq j < i \leq n$:

$$a_{ij} = \sum_{k=1}^{j} l_{ik} u_{kj}$$

and if $j \geq i$:

$$a_{ij} = \sum_{k=1}^{i} l_{ik} u_{kj}$$

& $l_{ii} = 1 \quad \forall i$

From this we directly get (see (2.18, 2.19) in theory book)

(1) $$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}$$

$i = 1, \dots, n$

$j = i, \dots, n$

c.f. previous

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik} \cdot a_{kj}^{(k)}$$

(2) $$l_{ij} = \frac{\left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}\right)}{u_{jj}}$$

$i = 2, \dots, n$

$j = 1, \dots, i-1$

c.f. our

$$l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$$

The LU factorization will complete successfully if

$$a_{kk}^{(k)} \neq 0$$

↑ kk

pivot element

But if pivot is zero it will fail.

The fix is easy:

Order of equations (and of unknowns) is arbitrary, so swap order to ensure that diagonal pivot is not zero

$$\begin{bmatrix} 1 & 1 & 3 \\ 2 & 2 & 2 \\ 3 & 6 & 4 \end{bmatrix} \begin{bmatrix} x_1 = 1 \\ x_2 = 1 \\ x_3 = 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \\ 13 \end{bmatrix}$$

$$\Downarrow$$

$$\left[\begin{array}{ccc} 1 & 1 & 3 \\ \hline 0 & 0 & -4 \\ 0 & 3 & -5 \end{array}\right]$$

swap 2nd & 3rd equations

$$\Downarrow$$

$$\left[\begin{array}{ccc} 1 & 1 & 3 \\ \hline 0 & 3 & -5 \\ 0 & \emptyset & -4 \end{array}\right]$$

we are done here since no x_2 in 3rd equation

$$L = \left[\begin{array}{ccc} 1 & & \\ \hline 3 & 1 & \\ \hline 2 & \emptyset & 1 \end{array}\right] \qquad U = \left[\begin{array}{ccc} 1 & 1 & 3 \\ \hline & 3 & -5 \\ \hline & & -4 \end{array}\right]$$

Now

$$LU = PA = \begin{bmatrix} 1 & 1 & 3 \\ 3 & 6 & 4 \\ 2 & 2 & 2 \end{bmatrix}$$

Permutation matrix (not so important)

$$= Pb = \begin{bmatrix} 5 \\ 13 \\ 6 \end{bmatrix}$$

Called row pivoting

Theorem:

If A is non singular row-pivoted LU factorization will succeed, i.e.

$$PA = LU$$

for some permutation.

What is the "best" P?

Example

$$\left[\begin{array}{c|c} 10^{-20} & 1 \\ \hline 1 & 1 \end{array}\right]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$\Downarrow$$

$$\left[\begin{array}{c|c} 10^{-20} & 1 \\ \hline 0 & 1-10^{20} \end{array}\right]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2-10^{20} \end{bmatrix}$$

Due to roundoff error

$$1 - 10^{20} \simeq 2 - 10^{20} \simeq -10^{20}$$

$$\left[\begin{array}{c|c} 10^{-20} & 1 \\ \hline 0 & -10^{20} \end{array}\right]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -10^{20} \end{bmatrix}$$

$$\Rightarrow x_2 = 1$$

$$10^{-20} x_1 + 1 = 1$$

$$\Rightarrow \quad x_1 = 0$$

But the true solution is

$$x_1 \simeq x_2 \simeq 1$$

This is easy to see if we had swapped order of equations or variables

$$\begin{bmatrix} 1 & 1 \\ 10^{-20} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$\simeq \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$\Rightarrow x_1 = 1 \qquad x_2 = 1$$

as needed

Or if we did column pivoting and swapped x_1 and x_2

$$\begin{bmatrix} 1 & 10^{-20} \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$\simeq \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

LU $\Downarrow$ $\Rightarrow x_2 = 1 = x_1$

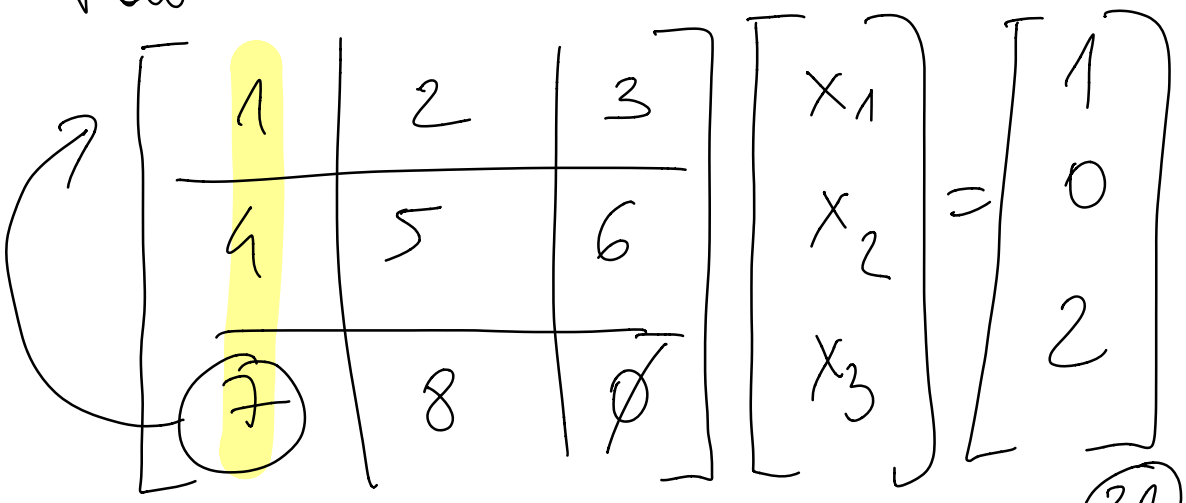
$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

and LU / Gauss would work just fine.

This shows that small pivots are a problem numerically

(dividing by number close to zero will lead to very large numbers and we will loose digits rapidly)

Idea: Partial (row) pivoting finds the largest pivot in a column at each step k and swaps that row with the k^{th} row



$$\left[\begin{array}{c|c|c} 7 & 8 & 0 \\ \hline 4 & 5 & 6 \\ 1 & 2 & 3 \end{array}\right] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}$$

$\Downarrow$ LU

$$\left[\begin{array}{c|c|c} 7 & 8 & 0 \\ \hline 0 & 3/7 & 6 \\ \hline 0 & 6/7 & 3 \end{array}\right] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ -8/7 \\ 5/7 \end{bmatrix}$$

$$\left[\begin{array}{ccc|c} 7 & 8 & 0 & 2 \\ 0 & 6/7 & 3 & 5/7 \\ 0 & 3/7 & 6 & -8/7 \end{array}\right]$$

$\Downarrow$ LU

$$\left[\begin{array}{ccc|c} 7 & 8 & 0 & 2 \\ 0 & 6/7 & 3 & 5/7 \\ 0 & 0 & 9/2 & -3/2 \end{array}\right]$$

$$PA = LU = Pb$$

(requires permuting the rhs)

$$P^{-1} = P^T$$ (reverse permutation)

$$(P^{-1}P)A = (P^{-1}L)U = b$$

$$A = (P^T L)U$$

$$A = \tilde{L}U$$

where $\tilde{L}$ can be made lower triangular by permuting the rows

(MATLAB has algorithms to find the right permutation)

So without loss of generality we can forget about the permutation in MATLAB for simplicity.

How do we solve

$$Ax = b \quad ?$$

$$L(Ux) = b$$

$$\begin{cases} Ly = b & \text{solve first} \\ Ux = y & \text{solve second} \end{cases}$$

These systems are trivial to solve by forward substitution for $(Ly = b)$ or backward substitution $(Ux = y)$

$$\begin{bmatrix} l_{11} & & \\ l_{21} & l_{22} & \\ \cdots & & \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \cdots \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \end{bmatrix}$$

$$y_1 = b_1 / l_{11}$$

$$y_2 = \frac{b_2 - l_{21} y_1}{l_{22}}$$

$\frac{1}{11}$

$$\Rightarrow \quad y_i = \left(b_i - \sum_{j=1}^{i-1} l_{ij} y_j\right) / l_{ii}$$

Matlab code

for $i = 1 : n$

```
y(i) = b(i) - sum( L(i,1:i-1).*
                   y(1:i-1) )
```

In MATLAB:

$x = A \backslash b$ ← backslash

help on "mldivide"

Never do $x = inv(A) * b$

unless there is a really good reason since may NOT be numerically stable & may loose digits & also more expensive (~ factor of 2)!

Or, do:

$[L, U] = lu(A)$

$y = L \backslash b$

$x = U \backslash y$

(note: Matlab can give you P also)

Computational cost

It is important to have some estimates of how many computations an algorithm does. This is the most direct (but not the only or even the most important on modern computers) indicator of computational efficiency or cost.

E.g.

(1) If we double the size of A, how much longer will I need to wait for the answer?

(2) How much more powerful of a computer do I need to be able to solve a system with $n = 10^5$ variables in less than 10 mins?

The exact answers are essentially impossible to get without actually running code, but we can get some idea about scalability of code by counting arithmetic operations $(+, -, /, *)$ — called FLOPs (floating-point operations) — between "real" numbers.

Let's do this for GEM

Forward / Backward substitution:

At step i:

$$y(i) = b(i) - \text{sum}\left(L(i,1:i-1) .* y(1:i-1)\right)$$

$(i-1)$ multiplications and additions, plus one subtraction, giving total:

$$\sim 2\sum_{i=1}^{n}(i-1) = 2\sum_{i=0}^{n-1} i = 2\frac{n(n-1)}{2}$$

$$\simeq 2\frac{n^2}{2} = n^2$$

Forward / backward substitution costs $\sim n^2$ FLOPS

Usually we just write $O(n^2)$ since this is just a rough estimate

Now, to do the same for LU factorization is more tedious & you will do that in Worksheet #3.

But roughly, at step k you need to operate on a matrix of size $(n-k) \times (n-k)$ to eliminate x_k from all subsequent eqs.

So the # of FLOPS is roughly

$$2\sum_{k=1}^{n-1}(n-k)^2 = 2\sum_{k=1}^{(n-1)} k^2$$

In the worksheet, you will see that $\sum_{k=1}^{m} k^2 = \frac{1}{3}m^3 + O(m^2)$

(think of $\int x^2 dx = \frac{x^3}{3}$)

so the leading-order term (one that has the highest power of n and thus dominates for large n) is

Cost $\sim \frac{2}{3}n^3 + O(n^2)$

LU factorization costs $O(n^3)$ FLOPS

If you double n the time taken increases 8 times!

The cost of solving

$$Ax = b$$

is therefore dominated by LU factorization (not forward or backward substitution) and is $\sim n^3$ FLOPS

This is the curse of numerical LA - we cannot do it for $n = 10^6$ variables as we often do in say engineering or data science.

Roundoff & GEM

Remember that the conditioning number

$$K(A) = \|A\| \|A^{-1}\| > 1$$

tells us how accurately we can get x in the presence of roundoff:

$$\frac{\|\delta x\|}{\|x\|} \leq K(A) \frac{\|\delta b\|}{\|b\|}$$

Due to roundoff error,

$$\frac{\|\delta b\|}{\|b\|} \gtrsim \varepsilon \sim 10^{-16}$$ for double precision

So we expect that the relative error

$$\frac{\|\delta x\|}{\|x\|} \gtrsim 10^{-16} \cdot K(A)$$

So $K(A) = 10^6$ means we loose 6 digits and only get 10 digits in x.

This is the best-case scenario. All of the arithmetic operations ($O(n^3)$ of them!) could cause a much bigger error.

Pivoting is crucial to control roundoff error in GEM

Sometimes it is not enough, but this is rare in practice. A weaker goal is to at least require that the residual

$$r = Ax - b$$

is small

$$\frac{\|\delta r\|}{\|b\|} \sim \varepsilon = 10^{-16}$$

This is called backward stability and means we found an approximate solution even if it is not the solution. Pivoted LU is backwards stable

A more precise definition of backward stability is that the solution we obtain is the correct solution to a nearby problem, i.e., that

$$x = A \backslash b \implies$$

$$(A + \delta A)x = b + \delta b$$

where

$$\frac{\|\delta A\|}{\|A\|} \sim \varepsilon \sim 10^{-16}$$

$$\frac{\|\delta b\|}{\|b\|} \sim \varepsilon \sim 10^{-16}$$

No guarantees on how x relates to the true solution of $A x_{true} = b$