

Square Linear Systems

Spring 2021 , A. D ÖNER

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

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

e.g.

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

(1)

$$\xleftarrow{\text{A}} \xrightarrow{\text{X}} = \xrightarrow{\text{b}}$$

e.g. $A = \begin{bmatrix} 3 & 2 & 0 \\ 1 & -1 & 1 \\ 2 & 0 & 3 \end{bmatrix}$ $b = \begin{bmatrix} 2 \\ 1 \\ 5 \end{bmatrix}$

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

(2)

Standard approach is to use
Gaussian elimination

Step 1: Eliminate x_1
 Denote $\tilde{A}^{(1)} = [3 \times 3] = A$

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

Step as subscript

To eliminate x_1 from 2nd eq,
 multiply first row by $a_{21}^{(1)}$

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

and subtract from 2nd ③

To eliminate x_1 from 3rd eq,

multiply first row by

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

eq # $\nearrow 31$ variable #

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

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

will be the general k^{th} step

(4)

where

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

$$\left[a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)} a_{kj}^{(k)}}{a_{kk}^{(k)}} \right]$$

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

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

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

$$b_i^{(k+1)} = b_i^{(k)} - l_{ik} b_k^{(k)}$$
(5)

Step 2 : Eliminate x_2 from
all subsequent equations

$$\left[\begin{array}{ccc|c} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & b_1^{(1)} \\ -\cancel{\ell_{21}} & a_{22}^{(2)} & a_{23}^{(2)} & b_2^{(2)} \\ -\cancel{\ell_{31}} & a_{32}^{(2)} & a_{33}^{(2)} & b_3^{(2)} \end{array} \right] = \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array} \right]$$

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

Multiply second row (red part only) by

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

and subtract from 3rd eq. ⑥

$$\left[\begin{array}{ccc|c} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & | \\ \emptyset & a_{22}^{(2)} & a_{23}^{(2)} & | \\ \emptyset & \emptyset & a_{33}^{(3)} & | \end{array} \right] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(3)} \end{bmatrix}$$

ℓ_{21} ℓ_{31} ℓ_{32}

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 $a_{22}^{(2)}x_2 + a_{23}^{(2)}x_3 = b_2^{(2)}$ (7)

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

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

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

⑧

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

⑨

Numerical linear algebra
is a study of matrix
factorizations

How do we show / prove this?

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

$$\left\{ \begin{array}{l} a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik} \cdot a_{kj}^{(k)} \\ b_i^{(k+1)} = b_i^{(k)} - l_{ik} b_k \end{array} \right.$$

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

(10)

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) (11)

for $j = k+1 : n$

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

$$\underline{A((k+1) : n, k)} * A(k, j);$$

$$a_{ij} \leftarrow a_{ij} - l_{ik} a_{kj} \quad [i, j > k]$$

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 "Practise"
textbook of Greenbaum

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

(12)

"Proof" that $A = LU$

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

l_{ik} *u_{kj}*
 contracted

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

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

and if $j \geq i$:

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

& $l_{ii} = 1 \neq 0$

From this we directly get
 (see (2.18, 2.19) in theory book)
 (13)

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

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

c.f. previous

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

$$\textcircled{2} \quad \left(a_{ij} - \sum_{h=1}^{j-1} l_{ih} u_{kj} \right)$$

$$l_{ij} = \frac{u_{jj}}{u_{ij}}$$

c.f. our

$i = 2, \dots, n$
 $j = 1, \dots, i-1$

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

(14)

The LU factorization will complete successfully if

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

\uparrow 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

$$\xrightarrow{\Downarrow} \left[\begin{array}{ccc|c} 1 & 1 & 3 \\ 2 & 2 & 2 \\ 3 & 6 & 4 \end{array} \right] \left[\begin{array}{c} x_1 = 1 \\ x_2 = 1 \\ x_3 = 1 \end{array} \right] = \left[\begin{array}{c} 5 \\ 6 \\ 13 \end{array} \right]$$

$$\Downarrow$$

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

swap 2nd &
3rd equations

$$\Downarrow$$

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

we are done
here since
no x_2 in 3rd
equation

$$L = \left[\begin{array}{c|c|c|c} 1 & & & \\ \hline 3 & 1 & & \\ \hline 2 & \emptyset & 1 & \end{array} \right]$$

$$U = \left[\begin{array}{c|c|c|c} 1 & 1 & 3 & \\ \hline & 3 & -5 & \\ \hline & & -4 & \end{array} \right]$$

(16)

Now

$$L U = P A = \begin{bmatrix} 1 & 1 & 3 \\ 3 & 6 & 4 \\ 2 & 2 & 2 \end{bmatrix}$$

Permutation
matrix

(not so
important)

Called

$$= P b =$$

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

Row pivoting

Theorem:

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

$$P A = L U$$

for some permutation.

What is the "best" P?

(17)

Example

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



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

Due to roundoff error

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

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

$$\Rightarrow x_2 = \frac{1}{20}$$

$$10^{-20} x_1 + 1 = 1 \quad (18)$$

$$\Rightarrow x_1 = 0$$

But the true solution is

$$x_1 \approx x_2 \approx 1$$

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

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

$$\sim \begin{bmatrix} 1 & 1 \\ \emptyset & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$\Rightarrow x_1 = 1 \quad x_2 = 1$$

as needed

(19)

Or if we did column pivoting
and swapped x_1 and x_2

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

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

$\text{LU } \Downarrow \Rightarrow x_2 = 1 = x_1$

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

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 lose 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

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

(21)

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

↓ LU

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

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

↓ LU

$$\left[\begin{array}{ccc|c} 7 & 8 & 0 & 2 \\ 0 & 6/7 & 3 & 5/7 \\ 0 & 0 & 9/2 & -3/2 \end{array} \right] \quad (22)$$

$$PA = LU = P\ell$$

(requires permuting the rhs)

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

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

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

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

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 or *backward substitution*
for $(Ly=b)$ $(Ux=y)$

$$\left[\begin{array}{cc} l_{11} & \\ \hline l_{21} & l_{22} \\ \hline & \dots \end{array} \right] \left[\begin{array}{c} y_1 \\ \hline y_2 \\ \hline \dots \end{array} \right] = \left[\begin{array}{c} b_1 \\ \hline b_2 \\ \hline \dots \end{array} \right]$$

$$y_1 = \frac{b_1}{l_{11}}$$

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

1
II

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

Matlab code

for $i = 1 : n$

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

(25)

In MATLAB:

$x = A \backslash b$ backslash
 help on "ml divide"

Never do $x = \text{inv}(A) * b$

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

Or, do:

$$[L, U] = \text{lu}(A)$$

$$y = L \backslash b$$

$$x = U \backslash y$$

(note: Matlab can give you P also) (26)

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 :

$$y(i) = b(i) - \sum L(i, 1:i-1) * y(1:i-1)$$

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

total:

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

Forward / Backward substitution

Costs $\sim n^2$ FLOPS

Usually we just write $O(n^2)$ since
this is just a rough estimate 29

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

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

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!

(31)

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
 $n = n_0^6$ 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,

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

(35)

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

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

This is called backward
stability and means we

found an approximate
solution even if it is not
the solution. Pivoted LU is
backwards stable

(35)

