Elementary Matrices

This turns row operations into matrix multiplication.
It lets us use matrix algebra to reason about elimination.

Def: An **elementary matrix** is a matrix obtained from the identity matrix by doing one row operation.

Eg:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \xrightarrow{R_2 += 2R_1} \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \xrightarrow{R_2 \times = 3} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \xrightarrow{R_1 \longleftrightarrow R_2} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Fact: If E is the elementary matrix for a row operation, then

$$E \cdot A = \left(\begin{array}{c}\text{the matrix obtained by doing the} \\ \text{same row operation to } A\end{array}\right)$$

$$\boxed{\text{row operations} \equiv \text{left-multiplication by elementary matrices}}$$

Eg:

$$A=\begin{pmatrix} 1 & 1 & -1 & 3 \\ 0 & 4 & 3 & 3 \\ 5 & -3 & -6 & -6 \end{pmatrix} \xrightarrow{R_3 - 5R_1} \boxed{\begin{pmatrix} 1 & 1 & -1 & 3 \\ 0 & 4 & 3 & 3 \\ 0 & -8 & -1 & -21 \end{pmatrix}}$$

$$E=\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \xrightarrow{R_3 - 5R_1} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -5 & 0 & 1 \end{pmatrix} \quad \Big\| \text{ same!}$$

$$EA = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -5 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 1 & -1 & 3 \\ 0 & 4 & 3 & 3 \\ 5 & -3 & -6 & -6 \end{pmatrix} = \boxed{\begin{pmatrix} 1 & 1 & -1 & 3 \\ 0 & 4 & 3 & 3 \\ 0 & -8 & -1 & -21 \end{pmatrix}}$$

Left-multiplication by $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -5 & 0 & 1 \end{pmatrix}$ does $\xrightarrow{R_3 - 5R_1}$

Fact: An elementary matrix is invertible. Its inverse is the elementary matrix that un-does the row operation.

Why?

$$E_1 = \begin{pmatrix} \text{elementary} \\ \text{matrix for} \\ \xrightarrow{R_3 - 5R_1} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -5 & 0 & 1 \end{pmatrix}$$

$$E_2 = \begin{pmatrix} \text{elementary} \\ \text{matrix for} \\ \xrightarrow{R_3 + 5R_1} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 5 & 0 & 1 \end{pmatrix}$$

$$E_2E_1 = E_2E_1I_3 = E_2(E_1I_3) = \begin{pmatrix} \text{first multiply } I_3 \text{ by } E_1 \\ \text{then multiply that by } E_2 \end{pmatrix}$$

What does that do?

$$I_3 \xrightarrow[\text{mult. by } E_1]{R_3 -= 5R_1} E_1 I_3 \xrightarrow[\text{mult. by } E_2]{R_3 += 5R_1} E_2(E_1 I_3)$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \xrightarrow[\text{mult. by } E_1]{R_3 -= 5R_1} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -5 & 0 & 1 \end{pmatrix} \xrightarrow[\text{mult. by } E_2]{R_3 += 5R_1} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

This does the row operation $R_3 -= 5R_1$, then un-does it! So you get I_3 back.

$$E_2 E_1 = I_3 \quad \text{means} \quad E_1^{-1} = E_2$$

This illustrates the following important point:

> If E is an elementary matrix, then compute $E \cdot A$ by doing a row operation.
> Not by multiplying matrices!

What if you do multiple row operations?

Consider these elementary matrices:

$E_1 : R_1 += 2R_2 \qquad E_2 : R_2 \times= 2 \qquad E_3 : R_2 \leftrightarrow R_3$

Let's apply these in order to a matrix A:

$$A \xrightarrow{R_1 += 2R_2} E_1 A \xrightarrow[\substack{\text{do } R_2 \times= 2 \\ \text{to } E_1 A}]{R_2 \times= 2} E_2(E_1 A) \xrightarrow[\substack{\text{do } R_2 \leftrightarrow R_3 \\ \text{to } E_2 E_1 A}]{R_2 \leftrightarrow R_3} E_3(E_2(E_1 A))$$

$$= (E_3 E_2 E_1) A$$

Why did the elementary matrices end up in the opposite order?

$$E_3E_2E_1A = E_3E_2(E_1A)$$
$$= \text{first multiply by } E_1, \text{ then by } E_2, \text{ then } E_3$$

Application to Invertibility:

Suppose $A \xrightarrow{RREF} I_n$. That means you can do some number of row operations to A to get I_n. Let $E_1, \ldots, E_r$ be the elementary matrices for these row operations. Then

$$I_n = (E_r \cdots E_2E_1)A$$
$$\Longrightarrow A^{-1} = E_r \cdots E_2E_1$$

In particular, A is invertible!

It also tells us how to compute A^{-1} using row operations:

column-first matrix multiplication $\Rightarrow C(A|B) = (CA|CB)$

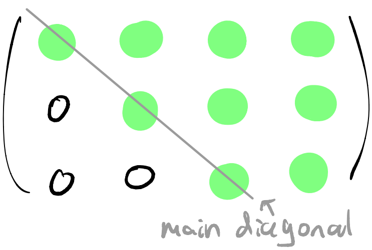
$$(E_r \cdots E_2E_1)(A|I_n) = \big((E_r \cdots E_2E_1)A \,\big|\, (E_r \cdots E_2E_1)I_n\big)$$
$$= (I_n \mid A^{-1})$$

This means if you do the same row operations to I_n, then you get A^{-1}: that's our algorithm from last time.

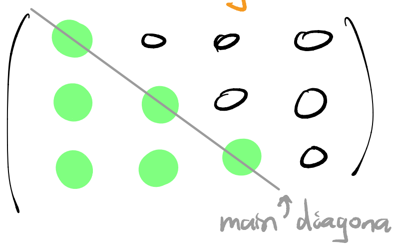
Triangular Matrices

Def: A matrix is upper/lower triangular if all of the entries below/above the main diagonal are zero.

upper triangular



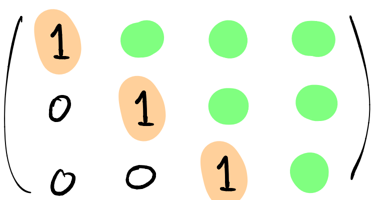
lower triangular



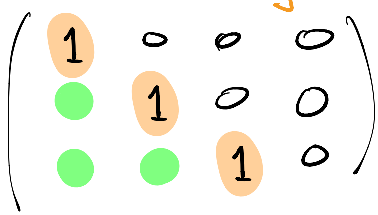
 = any number

Def: A matrix is upper/lower unitriangular if it is upper/lower triangular and all diagonal entries = 1.

upper unitriangular



lower unitriangular



 = any number

Eg: These matrices are upper-Δular but not uniΔular:

$$\begin{pmatrix} 1 & 4 & 5 & 7 \\ 0 & 2 & 6 & 8 \\ 0 & 0 & 3 & 9 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 4 & 5 & 7 \\ 0 & 0 & 6 & 8 \\ 0 & 0 & 0 & 9 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

 = any number (including zero!)

Eg: These matrices are lower uniΔular:

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 4 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 5 & 1 \\ 4 & 6 & 7 \end{pmatrix}$$

= any number (including zero!)

NB: A matrix is diagonal $\iff$ it is both upper- & lower-Δular: that means <u>all</u> off-diagonal entries are zero.

Eg: A matrix in REF is upper-Δular:

$$\begin{pmatrix} 1 & 1 & -1 & 3 \\ 0 & 4 & 3 & 3 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

Eg: The elementary matrix for $R_i \mathrel{+}= cR_j$, $i > j$

(add a multiple of a row to a row below it)

is lower-uniΔular:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \xrightarrow{R_3 \mathrel{+}= 2R_1} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 0 & 1 \end{pmatrix}$$

Fact: If A & B are $n \times n$ upper (uni)Δular matrices then so are AB and A^{-1} (if A is invertible).
Likewise for lower (uni)Δular matrices.

Eg:
$$\begin{pmatrix}1&0&0\\1&1&0\\2&3&1\end{pmatrix}\begin{pmatrix}1&0&0\\4&1&0\\5&6&1\end{pmatrix}=\begin{pmatrix}1&0&0\\5&1&0\\19&9&1\end{pmatrix}$$

$$\begin{pmatrix}1&0&0\\1&1&0\\2&3&1\end{pmatrix}^{-1}=\left(\text{matrix inversion procedure...}\right)=\begin{pmatrix}1&0&0\\-1&1&0\\1&-3&1\end{pmatrix}$$

NB: Any square, uniΔular matrix is invertible:
$$\begin{pmatrix}1&a&b\\0&1&c\\0&0&1\end{pmatrix}\qquad \bullet = \text{pivots}$$

LU Decompositions:

If Gaussian elimination on A requires no row swaps, then

Using the prescribed algorithm, not your clever row ops!

$$A = LU$$

where:

L is lower uniΔular

U is an REF for A ($\Rightarrow$ upper-Δular)

Eg:
$$\begin{pmatrix}2&1&0\\4&4&4\\6&1&0\end{pmatrix}=A=LU=\begin{pmatrix}1&0&0\\2&1&0\\3&-1&1\end{pmatrix}\begin{pmatrix}2&1&0\\0&2&4\\0&0&4\end{pmatrix}$$

How does this speed up solving $Ax=b$?

How to solve $Ax=b$ using $A=LU$:

(1) Solve $Ly=b$ using substitution.

(2) Solve $Ux=y$ using substitution.

Then $Ax=(LU)x=L(Ux)=Ly=b$ ✓

Eg: Solve $\begin{pmatrix} 2 & 1 & 0 \\ 4 & 4 & 4 \\ 6 & 1 & 0 \end{pmatrix} x = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$ using

$$\begin{pmatrix} 2 & 1 & 0 \\ 4 & 4 & 4 \\ 6 & 1 & 0 \end{pmatrix} = A = LU = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & -1 & 1 \end{pmatrix}\begin{pmatrix} 2 & 1 & 0 \\ 0 & 2 & 4 \\ 0 & 0 & 4 \end{pmatrix}$$

(1) Solve $\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & -1 & 1 \end{pmatrix} y = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$ using substitution.

$$\left(\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 0 \\ 3 & -1 & 1 & 1 \end{array}\right) \xrightarrow[R_3 -= 3R_1]{R_2 -= 2R_1} \left(\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -2 \\ 0 & -1 & 1 & -2 \end{array}\right)$$

$$\xrightarrow{R_3 += R_2} \left(\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & -4 \end{array}\right) \rightsquigarrow y = \begin{pmatrix} 1 \\ -2 \\ -4 \end{pmatrix}$$

(2) Solve $\begin{pmatrix} 2 & 1 & 0 \\ 0 & 2 & 4 \\ 0 & 0 & 4 \end{pmatrix} x = \begin{pmatrix} 1 \\ -2 \\ -4 \end{pmatrix}$ using substitution.

$$\left(\begin{array}{ccc|c} 2 & 1 & 0 & 1 \\ 0 & 2 & 4 & -2 \\ 0 & 0 & 4 & -4 \end{array}\right) \xrightarrow{R_3 \div= 4} \left(\begin{array}{ccc|c} 2 & 1 & 0 & 1 \\ 0 & 2 & 4 & -2 \\ 0 & 0 & 1 & -1 \end{array}\right)$$

$$\xrightarrow{R_2 -= 4R_3} \left(\begin{array}{ccc|c} 2 & 1 & 0 & 1 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 1 & -1 \end{array}\right) \xrightarrow{R_2 \div= 2} \left(\begin{array}{ccc|c} 2 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -1 \end{array}\right)$$

$$\xrightarrow{R_1 -= R_2} \left(\begin{array}{ccc|c} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -1 \end{array}\right) \xrightarrow{R_1 \div= 2} \left(\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -1 \end{array}\right)$$

$$\rightsquigarrow x = \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix}$$

Check: $\begin{pmatrix} 2 & 1 & 0 \\ 4 & 4 & 4 \\ 6 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$ ✓

Wait, was that really any faster than elimination?

(1) Solve $\left(\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 0 \\ 3 & -1 & 1 & 1 \end{array}\right)$ using substitution:

This elimination goes down instead of up, but it amounts to the same thing — just reorder the rows if you like. $\rightsquigarrow$ n^2 flops

But the pivots are already $= 1$ (L is uniΔular)
so you don't have to do the n row scaling ops

$\rightsquigarrow$ $n^2 - n$ flops

(2) Solve $\left(\begin{array}{ccc|c} 2 & 1 & 0 & 1 \\ 0 & 2 & 4 & -2 \\ 0 & 0 & 4 & -4 \end{array}\right)$ using substitution:

This is just Jordan substitution $\rightsquigarrow$ n^2 flops.

Total: $\boxed{\text{Solving } \overset{n\times n}{A}x=b \text{ using } A=LU \text{ takes } 2n^2-n \approx 2n^2 \text{ flops}}$

This turned elimination+substitution $\approx \frac{2}{3}n^3$ flops into $2n^2$!
Of course, you still have to compute $A=LU$:

Where does $A=LU$ come from?

If you can do Gaussian elimination without row swaps then the only row operations you're allowed to do are

$R_i \mathrel{+}= cR_j,\ i>j$ (add a multiple of a row to a row below it)

The corresponding elementary matrices are lower-uniΔular:

$$\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix} \xrightarrow{R_3 \mathrel{+}= 2R_1} \begin{pmatrix}1&0&0\\0&1&0\\2&0&1\end{pmatrix} = E$$

Let $E_1,\dots,E_r$ be the elementary matrices for the row ops you performed in Gaussian elimination. They are lower-uniΔular.

$$A \xrightarrow[\text{ops}]{\text{row}} U = \mathrm{REF}(A) \quad \text{means}$$

$$U = E_r E_{r-1} \cdots E_2 E_1 A$$

(left-multiplication by E_i does row operation i)

Since the E_i's are lower-uniΔular, so are

$$(E_r E_{r-1} \cdots E_2 E_1) \quad \text{and} \quad L = (E_r E_{r-1} \cdots E_2 E_1)^{-1}$$
$$= E_1^{-1} E_2^{-1} \cdots E_{r-1}^{-1} E_r^{-1}$$

Then $LU = (E_r E_{r-1} \cdots E_2 E_1)^{-1} (E_r E_{r-1} \cdots E_2 E_1) A$
$= A$ ✓

NB: $L = E_1^{-1} E_2^{-1} \cdots E_{r-1}^{-1} E_r^{-1}$ "keeps track" of the row operations you did in Gaussian elimination.

NB: $A = LU$ is a matrix factorization: it is a way of writing a matrix as a product of simpler matrices.

→ We'll learn many of these

→ They all make different calculations faster.

This also gives you a way of computing L & U.

$$L = E_1^{-1} E_2^{-1} \cdots E_{r-1}^{-1} E_r^{-1} = E_1^{-1} E_2^{-1} \cdots E_{r-1}^{-1} E_r^{-1} I_m$$

This means:

- start with I_m
- multiply by E_r^{-1} means undo the last row op.
- multiply by E_{r-1}^{-1} means undo the $(r-1)^{st}$ row op.

  ⋮
- multiply by E_2^{-1} means undo the 2^{nd} row op.
- multiply by E_1^{-1} means undo the 1^{st} row op.

This is L. (U is still the REF.)

Eg: $$A = \begin{pmatrix} 2 & 1 & 0 \\ 4 & 4 & 4 \\ 6 & 1 & 0 \end{pmatrix} \xrightarrow{R_2 -= 2R_1} \begin{pmatrix} 2 & 1 & 0 \\ 0 & 2 & 4 \\ 6 & 1 & 0 \end{pmatrix} \xrightarrow{R_3 -= 3R_1} \begin{pmatrix} 2 & 1 & 0 \\ 0 & 2 & 4 \\ 0 & -2 & 0 \end{pmatrix}$$

$$\xrightarrow{R_3 += R_2} \begin{pmatrix} 2 & 1 & 0 \\ 0 & 2 & 4 \\ 0 & 0 & 4 \end{pmatrix} = U$$

Compute L:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \xrightarrow[\text{undo last row op}]{R_3 -= R_2} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \xrightarrow[\text{undo 2nd row op}]{R_3 += 3R_1} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & -1 & 1 \end{pmatrix}$$

$$\xrightarrow[\text{undo 1st row op}]{R_2 += 2R_1} \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & -1 & 1 \end{pmatrix} = L$$

✓

Here's a better way of doing the bookkeeping.

Computing $A=LU$ using the 2-Column Method:

(1) Start with a blank $m\times m$ "L" matrix on the left and your matrix A on the right.

↳ Using the prescribed algorithm!

(2) Perform Gaussian elimination on A. For each row replacement $R_i \mathrel{+}= cR_j$ put $-c$ in the (i,j) entry of the L matrix.

(3) Add 1's to the diagonal entries of L & 0's to the rest of the blank entries.

Now L is on the left and U on the right.

Eg:

2 columns

	L	U
start blank	$\begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}$	$\begin{pmatrix} 2 & 1 & 0 \\ 4 & 4 & 4 \\ 6 & 1 & 0 \end{pmatrix}$ start with A
$\underbrace{R_2 \mathrel{-}= 2R_1}_{(c=-2)}$	$\begin{pmatrix} & & \\ 2 & & \\ & & \end{pmatrix}$ (2,1) entry	$\begin{pmatrix} 2 & 1 & 0 \\ 0 & 2 & 4 \\ 6 & 1 & 0 \end{pmatrix}$
$\underbrace{R_3 \mathrel{-}= 3R_1}_{(c=-3)}$	$\begin{pmatrix} & & \\ 2 & & \\ 3 & & \end{pmatrix}$ (3,1) entry	$\begin{pmatrix} 2 & 1 & 0 \\ 0 & 2 & 4 \\ 0 & -2 & 0 \end{pmatrix}$
$\underbrace{R_3 \mathrel{+}= 1R_2}_{(c=1)}$	$\begin{pmatrix} & & \\ 2 & & \\ 3 & -1 & \end{pmatrix}$ (3,2) entry	$\begin{pmatrix} 2 & 1 & 0 \\ 0 & 2 & 4 \\ 0 & 0 & 4 \end{pmatrix} = U$

Fill in the blank entries of L:

$$\begin{pmatrix} & & \\ 2 & & \\ 3 & -1 & \end{pmatrix} \rightsquigarrow \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & -1 & 1 \end{pmatrix} = L$$

Computational Complexity

Finding $A=LU$ is just Gaussian elimination + some extra bookkeeping.

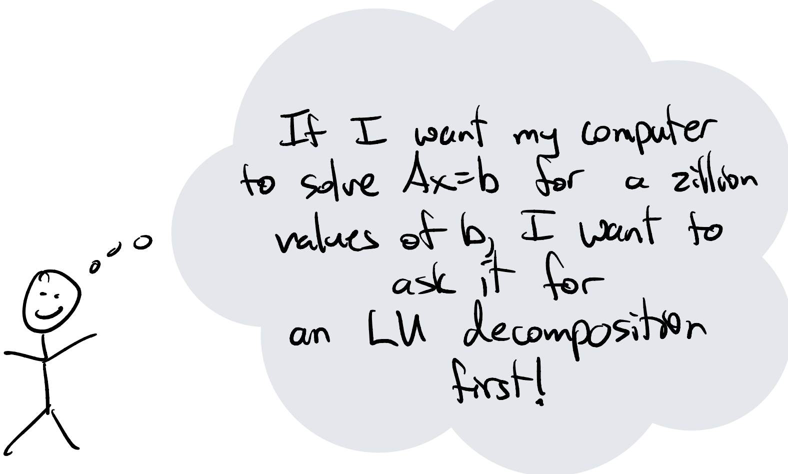
$$\boxed{\begin{array}{ll} \text{Computing } A=LU: & \approx \frac{2}{3}n^3 \text{ flops} \\ \text{Solving } Ax=b \text{ given } A=LU: & \approx 2n^2 \text{ flops} \end{array}}$$

How does this help? Isn't this just as long as Gauss-Jordan elimination?

Yes, unless you have to solve $Ax=b$ for 1,000,000 different values of b! In this case, you just do elimination once and substitution 1,000,000 x.

Eg: If $n=10^3$ and you have 10^6 b's:

- Gauss-Jordan 10^6 times is $10^6 \times \frac{2}{3}(10^3)^3 = \frac{2}{3}\cdot 10^{15}$ flops
- LU + substitution 10^6 times is $\frac{2}{3}(10^3)^3 + 10^6 \times 2(10^3)^2 \approx 2\cdot 10^{12}$ flops



This is faster in SymPy too:

```
from sympy import *
from time import time

  # This is the 10x10 matrix with 2's on the diagonal and 1's elsewhere
  #   eye(n) = nxn identity matrix; ones(n) = nxn matrix of 1's
  # (multiply by 1.0 to force it to use floating point arithmetic)
A = (eye(10) + ones(10)) * 1.0
  # This is the vector [1,1,1,1,1,1,1,1,1,1]
b = ones(10, 1) * 1.0

start = time()
  # Compute LU decomposition
L, U, _ = A.LUdecomposition()
  # Solve using substitution 1000 times
for _ in range(1000):
    U.upper_triangular_solve(L.lower_triangular_solve(b))
end = time()
print(end - start)
  # "7.144780397415161" (seconds)

  # Now solve using elimination 1000 times
start = time()
for _ in range(1000):
    A.solve(b) end = time()
print(end - start)
  # "48.048372983932495" (seconds)
  # Roughly 10x slower!
```

What about inverses? Isn't it faster to solve for x by multiplying by A^{-1}? $Ax=b \iff x=A^{-1}b$

- Computing A^{-1} uses $\approx \frac{4}{3}n^3$ flops.
- Multiplying $A^{-1}b$ uses $\approx 2n^2$ flops too!
- Computing A^{-1} ends up introducing more rounding errors.

Maximal Partial Pivoting

The system of equations

$$\begin{aligned} x_2 &= 1 \\ x_1 + x_2 &= 2 \end{aligned}$$

has one solution

$$\begin{aligned} x_1 &= 1 \\ x_2 &= 1 \end{aligned}$$

Let's tweak it just a little bit:

$$\begin{aligned} -10^{-17}x_1 + x_2 &= 1 \\ x_1 + x_2 &= 2 \end{aligned}$$

Presumably this has one solution $x_1 \simeq 1$, $x_2 \simeq 1$.

$$\left(\begin{array}{cc|c} -10^{-17} & 1 & 1 \\ 1 & 1 & 2 \end{array}\right) \xrightarrow{R_2 += 10^{17}R_1} \left(\begin{array}{cc|c} -10^{-17} & 1 & 1 \\ 0 & 1+10^{17} & 2+10^{17} \end{array}\right)$$

$$\xrightarrow{R_2 \div= 1+10^{17}} \left(\begin{array}{cc|c} -10^{-17} & 1 & 1 \\ 0 & 1 & \frac{2+10^{17}}{1+10^{17}} \end{array}\right)$$

$$\frac{2+10^{17}}{1+10^{17}} = \frac{1+10^{17}}{1+10^{17}} + \frac{1}{1+10^{17}} = 1 + \frac{1}{1+10^{17}}$$

$$\xrightarrow{R_1 -= R_2} \left(\begin{array}{cc|c} -10^{-17} & 0 & -\frac{1}{1+10^{17}} \\ 0 & 1 & 1+\frac{1}{1+10^{17}} \end{array}\right) \xrightarrow{R_1 \times\!\!= -10^{17}} \left(\begin{array}{cc|c} 1 & 0 & \frac{10^{17}}{1+10^{17}} \\ 0 & 1 & 1+\frac{1}{1+10^{17}} \end{array}\right)$$

So $x_1 = \frac{10^{17}}{1+10^{17}} \simeq 1$ and $x_2 = 1 + \frac{1}{1+10^{17}} \simeq 1$ ✓

Let's try this on the computer.

```python
from sympy import *
  # 1e-17 is 10^(-17)
A = Matrix([[-1e-17, 1.0, 1.0],
            [   1.0, 1.0, 2.0]])

  # This does R2 += 10^(17) R1
  # (force sympy to use the smaller pivot)
A.row_op(1, lambda v, j: v + 1e17*A[0,j])
pprint(A)
  # [-1e-17     1     1]
  # [      0 1e17 1e17]

  # Now do Jordan substitution
pprint(A.rref(pivots=False))
  # [1 0 0]
  # [0 1 1]
  # This answer is numerically very wrong!
```

What went wrong?

Most computers represent decimal numbers in 64-bit floating point format.

https://en.wikipedia.org/wiki/IEEE_754

This amounts to ≈ 17 decimal digits of precision.

When the computer does

$$\left(\begin{array}{cc|c} -10^{-17} & 1 & 1 \\ 1 & 1 & 2 \end{array}\right) \xrightarrow{R_2 \mathrel{+}= 10^{17} R_1} \left(\begin{array}{cc|c} -10^{-17} & 1 & 1 \\ 0 & 1+10^{17} & 2+10^{17} \end{array}\right)$$

it computes

$$1+10^{17} = \overbrace{10000000000000000}^{17 \text{ digits}}1$$

but it forgets the least significant (18^{th}) digit:

$$\begin{aligned} 1+10^{17} &= 100000000000000001 \\ &= 100000000000000000 \\ &= 10^{17} \end{aligned}$$

Likewise:

$$\begin{aligned} 2+10^{17} &= 100000000000000002 \\ &= 100000000000000000 \\ &= 10^{17} \end{aligned}$$

So the computer does

$$\left(\begin{array}{cc|c} -10^{-17} & 1 & 1 \\ 1 & 1 & 2 \end{array}\right) \xrightarrow{R_2 += 10^{17} R_1} \left(\begin{array}{cc|c} 10^{-17} & 1 & 1 \\ 0 & 10^{17} & 10^{17} \end{array}\right)$$

$$\xrightarrow{R_2 \div= 10^{17}} \left(\begin{array}{cc|c} -10^{-17} & 1 & 1 \\ 0 & 1 & 1 \end{array}\right) \xrightarrow{R_1 -= R_2} \left(\begin{array}{cc|c} -10^{-17} & 0 & 0 \\ 0 & 1 & 1 \end{array}\right)$$

$$\xrightarrow{R_1 \times= -10^{17}} \left(\begin{array}{cc|c} 1 & 0 & 0 \\ 0 & 1 & 1 \end{array}\right) \rightsquigarrow \begin{array}{l} x_1 = 0 \\ x_2 = 1 \end{array} \quad ✗$$

Summary:

We had to divide R_1 by the 1^{st} pivot $= -10^{-17}$

ie, we multiplied it by 10^{17}

then added it to R_2

which had the effect of forgetting the rest of the entries of R_2.

How to fix this?

Row swap to chose the larger pivot!

$1 + 10^{-17} = 1$
$= 1 + 2 \cdot 10^{-17}$

$$\left(\begin{array}{cc|c} -10^{-17} & 1 & 1 \\ 1 & 1 & 2 \end{array}\right) \xrightarrow{R_1 \leftrightarrow R_2} \left(\begin{array}{cc|c} 1 & 1 & 2 \\ -10^{-17} & 1 & 1 \end{array}\right) \xrightarrow{R_2 += 10^{-17} R_1} \left(\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & 1 & 1 \end{array}\right)$$

$$\xrightarrow{R_1 -= R_2} \left(\begin{array}{cc|c} 1 & 0 & 1 \\ 0 & 1 & 1 \end{array}\right) \rightsquigarrow \begin{array}{l} x_1 = 1 \\ x_2 = 1 \end{array} \quad ✓$$

Much better.

There are several pivoting strategies for avoiding this kind of rounding error. Here is the simplest.

Gaussian Elimination with Maximal Partial Pivoting:

In step (a) of Gaussian elimination, perform a row swap so that (one of) the largest numbers in the column (in absolute value) becomes the new pivot.

NB: SymPy does this by default.

```
A = Matrix([[1e-17, 1.0, 1.0],
            [  1.0, 1.0, 2.0]])
  # Do Gauss-Jordan with MPP
pprint(A.rref(pivots=False))
  # [1 0 1]
  # [0 1 1]
```

The story so far:

- Can solve $Ax=b$ faster with $A=LU$, which only works when you can do Gaussian elimination with no row swaps.
- Elimination is much more numerically accurate if you row swap for every pivot.

The best of both worlds is:

PALU Decompositions

Def: A permutation matrix is a product of elementary matrices for row swaps.

So if P is a permutation matrix then

$$PA = (\text{do a bunch of row swaps on } A)$$
$$= (\text{rearrange the rows of } A)$$

PALU Decomposition:

Any matrix A has a factorization

$$PA = LU$$

where:

- P is a permutation matrix
- L is lower uniΔular
- U is an REF for A

Idea: First do all the row swaps on A (PA), then compute its LU decomposition ($PA = LU$).

Of course, you don't know which row swaps you'll need to do before doing elimination! Thankfully, this is taken care of with some slightly fancier bookkeeping.

Computing PA=LU Using the 3-Column Method.

(1) Start with the $m \times m$ identity matrix "P" on the left, a blank $m \times m$ "L" matrix in the middle, and your matrix A on the right.

(2) Do Gaussian Elimination on A.
↳ Using the prescribed algorithm!

- For each row replacement $R_i += cR_j$ put $-c$ in the (i,j) entry of the L matrix.
- For each row swap $R_i \leftrightarrow R_j$, swap the corresponding rows of L (including blank entries!) and P.

(3) Add 1's to the diagonal entries of L & 0's to the rest of the blank entries.

Now P is on the left, L in the middle, and U on the right.

Important: This only works if you do Gaussian elimination as prescribed! It will fail if you try to be more clever with your row operations.

Eg (PALU with Maximal Partial Pivoting):

Compute a $PA=LU$ decomposition using MPP for

$$A=\begin{pmatrix} 1 & 1 & 1 \\ -10 & -20 & -30 \\ 5 & 15 & 10 \end{pmatrix}.$$

	P	L	U
	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ -10 & -20 & -30 \\ 5 & 15 & 10 \end{pmatrix}$
$R_1 \leftrightarrow R_2$ $\xrightarrow{}$ choose largest pivot	$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}$	$\begin{pmatrix} -10 & -20 & -30 \\ 1 & 1 & 1 \\ 5 & 15 & 10 \end{pmatrix}$
$R_2 \mathrel{+}= \frac{1}{10}R_1$ $\xrightarrow{}$ $R_3 \mathrel{+}= \frac{1}{2}R_1$	$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} & & \\ -1/10 & & \\ -1/2 & & \end{pmatrix}$	$\begin{pmatrix} -10 & -20 & -30 \\ 0 & -1 & -2 \\ 0 & 5 & -5 \end{pmatrix}$
$R_2 \leftrightarrow R_3$ $\xrightarrow{}$ choose largest pivot	$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} & & \\ -1/2 & & \\ -1/10 & & \end{pmatrix}$	$\begin{pmatrix} -10 & -20 & -30 \\ 0 & 5 & -5 \\ 0 & -1 & -2 \end{pmatrix}$
$R_3 \mathrel{+}= \frac{1}{5}R_2$ $\xrightarrow{}$	$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} & & \\ -1/2 & & \\ -1/10 & -1/5 & \end{pmatrix}$	$\begin{pmatrix} -10 & -20 & -30 \\ 0 & 5 & -5 \\ 0 & 0 & -3 \end{pmatrix}$

Fill in the blank entries of L:

$$\begin{pmatrix} & & \\ -1/2 & & \\ -1/10 & -1/5 & \end{pmatrix} \rightsquigarrow \begin{pmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ -1/10 & -1/5 & 1 \end{pmatrix} = L$$

Check:

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}\begin{pmatrix} 1 & 1 & 1 \\ -10 & -20 & -30 \\ 5 & 15 & 10 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ -1/10 & -1/5 & 1 \end{pmatrix}\begin{pmatrix} -10 & -20 & -30 \\ 0 & 5 & -5 \\ 0 & 0 & -3 \end{pmatrix} \checkmark$$

Of course, this is only useful if we can use it to solve $Ax=b$:

How to solve $Ax=b$ using $PA=LU$:

(0) Compute Pb. (rearrange the entries of b)

(1) Solve $Ly=Pb$ using substitution.

(2) Solve $Ux=y$ using substitution.

Then $PAx=(LU)x=L(Ux)=Ly=Pb$.

(Multiply both sides by $P^{-1} \Rightarrow Ax=b$.)

Eg: Solve $Ax=b$ where

$$A=\begin{pmatrix} 1 & 1 & 1 \\ -10 & -20 & -30 \\ 5 & 15 & 10 \end{pmatrix} \qquad b=\begin{pmatrix} 0 \\ -10 \\ 10 \end{pmatrix}$$

using

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}\begin{pmatrix} 1 & 1 & 1 \\ -10 & -20 & -30 \\ 5 & 15 & 10 \end{pmatrix}=\begin{pmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ -1/10 & -1/5 & 1 \end{pmatrix}\begin{pmatrix} -10 & -20 & -30 \\ 0 & 5 & -5 \\ 0 & 0 & -3 \end{pmatrix}$$

(0) $Pb=\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}\begin{pmatrix} 0 \\ -10 \\ 10 \end{pmatrix}=\begin{pmatrix} -10 \\ 10 \\ 0 \end{pmatrix}$ (rearrange)

(1) Solve $Ly=Pb$ using substitution:

$$\left(\begin{array}{ccc|c} 1 & 0 & 0 & -10 \\ -1/2 & 1 & 0 & 10 \\ -1/10 & -1/5 & 1 & 0 \end{array}\right) \xrightarrow{(\dots)} \left(\begin{array}{ccc|c} 1 & 0 & 0 & -10 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & 0 \end{array}\right) \qquad y=\begin{pmatrix} -10 \\ 5 \\ 0 \end{pmatrix}$$

(2) Solve $Ux=y$ using substitution:

$$\left(\begin{array}{ccc|c} -10 & -20 & -30 & -10 \\ 0 & 5 & -5 & 5 \\ 0 & 0 & -3 & 0 \end{array}\right) \overset{(\dots)}{\rightsquigarrow} \left(\begin{array}{ccc|c} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{array}\right) \qquad x = \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix}$$

Check: $\begin{pmatrix} 1 & 1 & 1 \\ -10 & -20 & -30 \\ 5 & 15 & 10 \end{pmatrix} \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ -10 \\ 10 \end{pmatrix}$ ✓

Computational Complexity

Finding $PA=LU$ is just Gaussian elimination + some extra bookkeeping, as with $A=LU$.

And solving $Ax=b$ using $PA=LU$ only had the extra step (0), which is just rearranging (no flops).

So the complexity is the same as $A=LU$.

$$\boxed{\begin{array}{ll} \text{Computing } PA=LU: & \approx \frac{2}{3}n^3 \text{ flops} \\ \text{Solving } Ax=b \text{ given } PA=LU: & \approx 2n^2 \text{ flops} \end{array}}$$