LU Decomposition cont'd

Fact: If Gaussian elimination on A requires no row swaps, then

$$A = L\,U$$

for L lower-uniΔular and U in REF.

In this case $U = (E_r \cdots E_1)A$, $U = \text{REF}$

E_i = elementary matrix for $R_i \mathrel{+}= cR_j$, $i > j$ (clear down)

E_i is lower-Δular $\quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ c & 0 & 1 \end{bmatrix} R_3 \mathrel{+}= cR_1$

$\Rightarrow$ $E_r \cdots E_1$ is lower-Δular

$\Rightarrow$ $L = (E_r \cdots E_1)^{-1}$ is lower-Δular and $A = LU$

NB: $L = (E_r \cdots E_1)^{-1}$ "records the row operations"
$\rightarrow$ keeps track of elimination

NB: $A = LU$ is a matrix factorization: it is a way to write a matrix as a product of simpler matrices. We will learn many more of these.

Eg: $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \xrightarrow{R_2 -= 4R_1} \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 7 & 8 & 9 \end{bmatrix} \qquad E_1 = \begin{bmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\xrightarrow{R_3 -= 7R_1} \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & -6 & -12 \end{bmatrix} \qquad E_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -7 & 0 & 1 \end{bmatrix}$$

$$\xrightarrow{R_3 -= 2R_2} \underbrace{\begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & 0 \end{bmatrix}}_{U} \qquad E_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{bmatrix}$$

$$L = (E_3 E_2 E_1)^{-1} = E_1^{-1} E_2^{-1} E_3^{-1}$$

To compute $E_1^{-1} E_2^{-1} E_3^{-1} = E_1^{-1} E_2^{-1} E_3^{-1} I_3$,

start with I_3: $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$$\xrightarrow[E_3^{-1} = \text{undo } E_3]{R_3 += 2R_2} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix}$$

$$\xrightarrow[E_2^{-1} = \text{undo } E_2]{R_3 += 7R_1} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 7 & 2 & 1 \end{bmatrix}$$

$$\xrightarrow[E_1^{-1} = \text{undo } E_1]{R_2 += 4R_1} \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 2 & 1 \end{bmatrix} = L$$

Check: $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & 0 \end{bmatrix}$

Here's a better way to do the bookkeeping.

Algorithm (LU Decomposition; 2 column method):

Input: A matrix where Gaussian elimination requires no row swaps

Output: A factorization $A = LU$ for L lower-uniΔular and U in REF (the output of Gaussian elimination).

Procedure: Do Gaussian elimination, keeping track of the row operations as follows: start with a blank $m \times m$ matrix L.

- for each row replacement $R_i \mathrel{+}= c R_j$, put $-c$ in the (i,j) entry of L.

Add 1's & 0's to the part of L above the diagonal. Then

$$A = LU.$$

Eg: $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$

2 columns

	L	U
	[blank]	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ start with A in U column
$R_2 -= 4R_1$	$\begin{bmatrix} & & \\ 4 & & \\ & & \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 7 & 8 & 9 \end{bmatrix}$
$R_3 -= 7R_1$	$\begin{bmatrix} & & \\ 4 & & \\ 7 & & \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & -6 & -12 \end{bmatrix}$
$R_3 -= 2R_2$	$\begin{bmatrix} & & \\ 4 & & \\ 7 & 2 & \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & 0 \end{bmatrix}$

add these

$L = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 2 & 1 \end{bmatrix}$ $\qquad = U$

NB: Finding $A=LU$ is just Gaussian elimination + extra bookkeeping $\rightsquigarrow$ same complexity. $\approx \frac{2}{3}n^3$ flops

How does this help?

Algorithm (solving $Ax=b$ using $A=LU$)

Input: An LU factorization $A=LU$ & vector b

Output: A solution of $Ax=b$

Procedure:

(1) Solve $Ly=b$ using forward-substitution

$$\begin{bmatrix} 1 & 0 & 0 \\ \bullet & 1 & 0 \\ \bullet & \bullet & 1 \end{bmatrix} y = b \equiv \begin{aligned} y_1 &= b_1 \\ y_2 + \bullet\, y_1 &= b_2 \\ y_3 + \bullet\, y_2 + \bullet\, y_1 &= b_3 \end{aligned}$$

($\approx n^2$ flops)

(2) Solve $Ux=y$ using backward-substitution

($\approx n^2$ flops)

Then $Ax = LUx = L(Ux) = Ly = b$ ✓

NB: total complexity is $\approx 2n^2$ flops

$\rightarrow$ way faster than $\frac{2}{3}n^3$!

Still have to do elimination once, but then solving $Ax=b$ for new values of b is much faster.

Eg: Solve $Ax = \begin{pmatrix}1\\0\\1\end{pmatrix}$ using

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

(1) $Ly=b$: $\begin{aligned} y_1 \quad\quad\quad &= 1 \\ 2y_1 + y_2 \quad &= 0 \\ 3y_1 - y_2 + y_3 &= 1 \end{aligned}$ $\xrightarrow[\text{subst}]{\text{forward}}$ $y = \begin{pmatrix}1\\-2\\-4\end{pmatrix}$

(1) $Ux=y$: $\begin{aligned} 2x_1 + x_2 \quad\quad &= 1 \\ 2x_2 + 4x_3 &= -2 \\ 4x_3 &= -4 \end{aligned}$ $\xrightarrow[\text{subst}]{\text{backward}}$ $x = \begin{pmatrix}0\\1\\-1\end{pmatrix}$

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

Eg: If A is 1000×1000 and we want to solve $Ax=b$ for 1000 values of b:

- $\frac{2}{3}$ gigaflops to compute $A=LU$
- 2 megaflops $\times$ 1000 = 2 gigaflops to solve $Ax=b$ 1000 times.

That's 250× faster than $\frac{2}{3}$ teraflops from doing elimination 1000 times!

What about inverses?

Wouldn't it be better to compute A^{-1} and solve $A^{-1}x=b$ 1000 times?

No, for 2 reasons:

(1) Computing A^{-1} takes $\approx \frac{4}{3}n^3$ flops: twice as long as the elimination step!

(2) Computing A^{-1} is not numerically stable (less accurate due to rounding errors).

[linalg.js example below: try it yourself! You can copy/paste the code.]

```
  // This is the matrix [2 1 1 ... 1]
  //                    [1 2 1 ... 1]
  //                    [. . . ... .]
  //                    [1 1 1 ... 2]
A = Matrix.identity(1000).add(Matrix.constant(1,1000));
  // Computes and caches a PA=LU decomposition
A.PLU();
  // The vector (1,1,...,1)
b = Vector.constant(1000,1);
  // Solve Ax=b 1000 times using the cached PA=LU decomposition
  // This takes a few seconds on my browser.
for(i = 0; i < 1000; ++i) A.solve(b)
  // Solve Ax=b 1000 times *without* PA=LU decomposition.
  // This crashes my browser.
for(i = 0; i < 1000; ++i) { A.invalidate(); A.solve(b) }
```

Maximal Partial Pivoting

Eg: $\begin{cases} x_2 = 1 \\ x_1 + x_2 = 2 \end{cases}$ has one solution: $x_1 = x_2 = 1$

Let's tweak it a little bit:

$\begin{cases} 10^{-17}x_1 + x_2 = 1 \\ x_1 + x_2 = 2 \end{cases} \rightsquigarrow$ presumably has one solution $(x_1, x_2) \approx (1, 1)$.

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

$$\rightsquigarrow \begin{array}{l} 10^{-17}x_1 + x_2 = 1 \\ (1-10^{17})x_2 = 2 - 10^{17} \end{array}$$

$$\Rightarrow x_2 = \frac{2-10^{17}}{1-10^{17}} = \frac{1+(1-10^{17})}{1-10^{17}} = 1 + \frac{1}{1-10^{17}} \approx 1$$

$$x_1 = 10^{17}(1-x_2) = \frac{10^{17}}{10^{17}-1} \approx 1$$

Let's try this on a computer. [linalg.js]

What went wrong?

→ Javascript uses IEEE 754 64-digit floating point numbers. That means it has ≈ 16 digits of precision.

The computer thinks $1 - 10^{17} = -10^{17} = 2 - 10^{17}$

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

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

$$\rightsquigarrow \quad x_1 = 10^{17}(0) = 0$$

$(0, 1) \not\approx (1, 1)$: not numerically stable.

What went wrong?

Dividing by 10^{-17} produced a huge number 10^{17}

$\rightsquigarrow$ all errors just exploded!

> On a computer, you never want to divide by tiny numbers!

Solution: Use the larger pivot (in absolute value)

$$\left[\begin{array}{cc|c} 10^{-17} & 1 & 1 \\ 1 & 1 & 2 \end{array}\right] \xrightarrow{R_1 \leftrightarrow R_2} \left[\begin{array}{cc|c} 1 & 1 & 2 \\ 10^{-17} & 1 & 1 \end{array}\right]$$

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

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

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

[linalg.js]

```
  // Create the matrix [10^(-17) 1 1]
  //                   [       1 1 2]
A = mat([1e-17,1,1],[1,1,2]);
  // R2 -= 1/10^(-17) R1
A.rowReplace(1,0,-A[1][0]/A[0][0]);
  // What did this do to A?
console.log(A.toString());
  // Output: [0.0000                   1.0000                   1.0000]
  //         [0.0000 -100000000000000000.0000 -100000000000000000.0000]
  // Solve for x2; you get x2 = 1
x2 = A[1][2]/A[1][1];
  // solve for x1 using back-substitution
x1 = (A[0][2] - A[0][1]*x2) / A[0][0];
  // you get x1 = 0!
  // Evaluate this:
1 - 1e-17 // = 1

  // Let's try again using maximal partial pivoting
  // Create the matrix [10^(-17) 1 1]
  //                   [       1 1 2]
A = mat([1e-17,1,1],[1,1,2]);
  // R1 <-> R2
A.rowSwap(0,1);
  // R2 -= 10^(-17)/1 R1
A.rowReplace(1,0,-A[1][0]/A[0][0]);
  // Solve for x2; you get x2 = 1
x2 = A[1][2]/A[1][1];
  // solve for x1 using back-substitution
x1 = (A[0][2] - A[0][1]*x2) / A[0][0];
  // you get x1 = 1
```

Gaussian Elimination using Maximal Partial Pivoting:

In step (a), row swap so that the largest number in the column (in absolute value) is the pivot.

This is much more numerically stable.
→ avoids dividing by tiny numbers.

PA=LU Decompositions

Recall: LU only works when you don't need to row swap when eliminating.

But elimination works much better with row swaps!
Need to tweak LU.

Idea: Do all the row swaps you need first, then do elimination without row swaps.

(How do you know in advance which row swaps to do? You don't - need to do more bookkeeping)

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

PA=LU Decomposition: Any matrix A has a factorization

$$PA = LU$$

Do a bunch of row swaps on A...

... then do an LU decomp

where:

- P: permutation matrix
- L: lower-unitriangular matrix
- U: REF matrix

maximal partial pivoting

Eg:

$$A = \begin{bmatrix} 1 & 1 & 1 \\ -10 & -20 & -30 \\ 5 & 15 & 10 \end{bmatrix} \xrightarrow{R_1 \leftrightarrow R_2} \begin{bmatrix} -10 & -20 & -30 \\ 1 & 1 & 1 \\ 5 & 15 & 10 \end{bmatrix} \quad P_1$$

$$\xrightarrow[R_3 += \frac{1}{2}R_1]{R_2 += \frac{1}{10}R_1} \begin{bmatrix} -10 & -20 & -30 \\ 0 & -1 & -2 \\ 0 & 5 & -5 \end{bmatrix} \quad E_1, E_2$$

maximal partial pivoting

$$\xrightarrow{R_2 \leftrightarrow R_3} \begin{bmatrix} -10 & -20 & -30 \\ 0 & 5 & -5 \\ 0 & -1 & -2 \end{bmatrix} \quad P_2$$

$$\xrightarrow{R_3 += \frac{1}{5}R_2} \underbrace{\begin{bmatrix} -10 & -20 & -30 \\ 0 & 5 & -5 \\ 0 & 0 & -3 \end{bmatrix}}_{U} \quad E_3$$

wish P_2 were here...

$$U = \underbrace{E_3 P_2 E_2 E_1}_{\text{not lower-uni}\Delta\text{ular!}} P_1 A$$

Trick: $P_2 E_2 E_1 = \underbrace{(P_2 E_2 E_1 P_2)}_{\text{is lower-uni}\triangle\text{ular!}} P_2$ $P_2 P_2 = I_3$

$$P_2 E_2 E_1 P_2 = P_2 E_2 E_1 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\xrightarrow[R_3 += \frac{1}{2} R_1]{R_2 += \frac{1}{10} R_1} P_2 \begin{bmatrix} 1 & 0 & 0 \\ 1/10 & 0 & 1 \\ 1/2 & 1 & 0 \end{bmatrix}$$

$$\xrightarrow{R_1 \leftrightarrow R_2} \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 1/10 & 0 & 1 \end{bmatrix} \quad \text{lower-uni}\triangle\text{ular!}$$

Then $U = E_3 (P_2 E_2 E_1 P_2) P_2 P_1 A$

$\Longrightarrow PA = LU$

for $P = P_2 P_1 \quad L = (E_3 P_2 E_2 E_1 P_2)^{-1}$

Here is an efficient way of doing the bookkeeping.

Algorithm (PA=LU Decomposition; 3-column method):

Input: Any matrix A

Output: A factorization $PA = LU$ where:

P: permutation matrix

L: lower-unitriangular matrix

U: REF matrix

Procedure: Perform Gaussian elimination using any pivoting strategy (eg. maximal partial pivoting). Keep track of row operations as follows: start with a blank matrix L and an identity matrix P.

- for each row replacement $R_i \mathrel{+}= c R_j$, put $-c$ in the (i,j) entry of L (as before)
- For each row swap $R_i \longleftrightarrow R_j$, swap the corresponding rows of L and P.

Add 1's & 0's to the part of L above the diagonal. Then

$$PA = LU.$$

Eg: $A = \begin{bmatrix} 1 & 1 & 1 \\ -10 & -20 & -30 \\ 5 & 15 & 10 \end{bmatrix}$

3 columns

	P	L	U
	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} & \text{blank} & \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ -10 & -20 & -30 \\ 5 & 15 & 10 \end{bmatrix}$ ← Start with A in the U col
$R_1 \leftrightarrow R_2$	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} & \text{blank} & \end{bmatrix}$	$\begin{bmatrix} -10 & -20 & -30 \\ 1 & 1 & 1 \\ 5 & 15 & 10 \end{bmatrix}$
$R_2 += \frac{1}{10}R_1$, $R_3 += \frac{1}{2}R_1$	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} & & \\ -1/10 & & \\ -1/2 & & \end{bmatrix}$	$\begin{bmatrix} -10 & -20 & -30 \\ 0 & -1 & -2 \\ 0 & 5 & -5 \end{bmatrix}$
$R_2 \leftrightarrow R_3$	$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} & & \\ -1/2 & & \\ -1/10 & & \end{bmatrix}$	$\begin{bmatrix} -10 & -20 & -30 \\ 0 & 5 & -5 \\ 0 & -1 & -2 \end{bmatrix}$
$R_3 += \frac{1}{5}R_2$	$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} & & \\ -1/2 & & \\ -1/10 & -1/5 & \end{bmatrix}$	$\begin{bmatrix} -10 & -20 & -30 \\ 0 & 5 & -5 \\ 0 & 0 & -3 \end{bmatrix}$
	P		U

{add these}

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

Check:

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

Can we still use this to solve $Ax=b$?

Algorithm (solving $Ax=b$ using $PA=LU$)

Solving $Ax=b$ is the same as $PAx=Pb$, so:

(0) Compute Pb (re-order the entries of b)

(1) Solve $Ly=Pb$ using forward-substitution

(2) Solve $Ux=y$ using backward-substitution

Then $PAx = LUx = Ly = Pb$

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

Eg: Solve $Ax=b$ for

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

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

(1) $$\begin{bmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ -1/10 & -1/5 & 1 \end{bmatrix} y = Pb = \begin{bmatrix} -10 \\ 10 \\ 0 \end{bmatrix} \rightsquigarrow \begin{matrix} y_1=-10 \\ y_2=5 \\ y_3=0 \end{matrix}$$

(2) $$\begin{bmatrix} -10 & -20 & -30 \\ 0 & 5 & -5 \\ 0 & 0 & -3 \end{bmatrix} x = \begin{bmatrix} -10 \\ 5 \\ 0 \end{bmatrix} \rightsquigarrow \begin{matrix} x_1=-1 \\ x_2=1 \\ x_3=0 \end{matrix} \qquad x=\begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$$