Gaussian Elimination & LU factorization.

Is there how to solve \( Ax=b \) unless you know something nice or horrible about \( A \).

For today's lecture, \( A \) is an \( n \times n \) invertible matrix.

The LU factorization of \( A \) produces

a lower triangular (henceforth \( L \)) matrix \( L \), and an upper \( \Delta \) or matrix \( U \) such that \( LU = A \).

1. Whence \( L \)?
2. Pivoting to avoid \( 0 \).
3. Why \( L \)?

It is easy to solve equations in \( \Delta \) matrices, so

once we compute \( L \) and \( U \), it is easy to solve equations in \( A \).

"Easy": count ops

where \( f(n) \approx \Theta(g(n)) \) if \( f = c_1 g + c_2 n^z \), no \( z \),

\( c_1 g(n) \leq f(n) \leq c_2 g(n) \) for all \( n \geq n_0 \).

To compute \( Ax \) requires \( 2n^2 - n \) flops - floating point ops

Think of \( \Theta(n^2) \) ops to include memory ref, branches, etc.
Forward Substitution: solving $Lx = b$

$L$ looks like

$$
\begin{pmatrix}
L(1,1) & 0 & 0 & \cdots & 0 \\
L(2,1) & L(2,2) & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
L(n,1) & \cdots & \cdots & L(n,n) & 0
\end{pmatrix}
\begin{pmatrix}
x(1) \\
x(2) \\
\vdots \\
x(n-1) \\
x(n)
\end{pmatrix}
= 
\begin{pmatrix}
b(1) \\
b(2) \\
\vdots \\
b(n-1) \\
b(n)
\end{pmatrix}
$$

This says $L(i,i) \cdot x(i) = b(i)$, so set $x(i) = b(i)/L(i,i)$

$L(2,1) \cdot x(1) + L(2,2) \cdot x(2) = b(2)$,

so, once we know $x(i)$, we can set

$x(2) = \frac{1}{L(2,2)} \left( b(2) - L(2,1) \cdot x(1) \right)$

$x(3) = \frac{1}{L(3,3)} \left( b(3) - L(3,1) \cdot x(1) - L(3,2) \cdot x(2) \right)$

# flops to compute $x(i)$: $1 \text{ (\checkmark)}$

$x(2) : 3 \text{ (\checkmark - \times)}$

$x(3) : 5 \text{ (\checkmark - - \times \times)}$

$x(n) : 2n-1 \text{ one } \backslash \ (2n-1) - \ (2n-1) \times$

So, # flops to compute $x = 1 + 3 + 5 + \cdots + 2n-1 = n^2$

(from 1st lecture) or $\Theta(n^2)$ ops

One way to write this:

```
for i in 1 to n
    \[ x(i) = b(i) \]
    for j in 1 to (i-1)
        \[ x(i) = x(i) - L(i,j) \cdot x(j) \]
    \[ x(i) = x(i) / L(i,i) \]
```
Another way to write this:

\[ x = b \]
\[
\text{for } i \text{ in } 1 \text{ to } n
\]
\[
\begin{align*}
x(i) &= x(i) / L(i,i) \\
\text{for } j \text{ in } i+1 \text{ to } n
\end{align*}
\]
\[
\begin{align*}
x(j) &= x(j) - L(j,i) x(i)
\end{align*}
\]

This code is an operator that multiplies by \( L^{-1} \)

Backwards solve: solve \( Ux=b \)

\[
\begin{pmatrix}
U(1,1) & U(1,2) & \cdots & U(1,n) \\
0 & U(2,2) & \cdots & U(2,n) \\
\vdots & & \ddots & \vdots \\
0 & \cdots & 0 & U(n,n)
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
\vdots \\
x_n
\end{pmatrix}
= b
\]

Some things but start with \( x(n) \), so also \( n^2 \) flops

So, to solve \( Ax=b \), solve \( LUx=b \) by

1. Find \( y \) s.t. \( Ly=b \)
2. Find \( x \) s.t. \( Ux=y \)

\[
\Rightarrow \quad LUx = Ly = b
\]

Total \# flops = \( 2n^2 \)
Why do I call this easy?
A has $n^2$ entries, so is time $\sim \#$entries in $A$.
the ideal.

Multiplying $Ax$ uses $2n^2 - n$ flops, so is comparable
to multiplication.

How do we get $L$ and $U$?
And, why not compute $A^{-1}$, or $L^{-1}$ or $U^{-1}$?

Answers are mostly numerics and run time (it takes longer)
To start, let’s ask “why even compute $L$ or $U$”

Main goal: produce a sequence of operators
$T_1, \ldots, T_k$ so that $x = T_1(T_2(\ldots(T_k b)\ldots))$
which write as $x = T_1 \circ T_2 \circ \ldots \circ T_k b$.

E.g. $T_1 = L^{-1}$, $T_2 = U^{-1}$.
Computing $L$: apply operations to $A$ until it is upper $\Delta$ or.

Elimination: making entries of $A$ zero.

Idea: gradually zero out entries of $A$ until it is upper $\Delta$ or.

in order, zero out $A(2,1), A(3,1), \ldots, A(n,1)$

$A(3,2), A(4,2), \ldots, A(n,2)$

\ldots

$A(n, n-1)$

Example: $A(2,1) = A(2,1) - A(1,1) \cdot \frac{A(2,1)}{A(1,1)}$

subtract $\frac{A(2,1)}{A(1,1)}$ times first row from second.

To avoid changing $A$, let $M = A$, and modify $M$

Let $O_{ij}^c$ be operation on a vector $x$

that subtracts $c \cdot x[i,j]$ from $x[i,j]$

$O_{ij}^c (x|\{k\}) = \begin{cases} x[k] & \text{if } k \neq j \\ x[i,j] - c \cdot x[i,j] & \text{if } k = j \end{cases}$

$O_{ij}^c \circ O_{ij}^c \circ x = x$

Write $O_{ij}^c$ as a matrix $= I - cE_{j,i}$,

where $E_{j,i}$ is zero except $E(j,i) = 1$

$E(2,1) = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$

$E_{j,i} \cdot x = x[i,j] E_j$
Can apply to a matrix $M$ by applying to each of its columns.

We begin with $M = O_{\mu_2}^{M(\mu_1, \mu_1)} \cdot M$.

Eliminate all entries in first column, except $M(\mu_1, \mu_1)$, by

for $i$ in 2 to $n$

$c = M(\mu_1, i) / M(\mu_1, \mu_1)$

$M = O_{\mu_1 i}^c \cdot M$

In 2nd column, eliminate all entries in rows 3...$n$

for $i$ in 3 to $n$

$c = M(\eta_2, i) / M(\eta_2, \eta_2)$

$M = O_{\eta_2 i}^c \cdot M$

As $M(\eta, \mu_1) = 0$ for $i > 1$ at this point, none of these entries change.

Full algorithm:

$M \leftarrow A$

For $i$ in 1 to $n-1$

For $j$ in $i+1$ to $n$ (so $j > i$)

$c_{ij} = M(\eta_i, \mu_1) / M(\eta_i, \eta_i)$

$M = O_{\eta_i \mu_1}^{c_{ij}} \cdot M$
Example: \[
\begin{bmatrix}
2 & 1 & 0 \\
4 & 3 & 1 \\
8 & 9 & 5 \\
6 & 9 & 8
\end{bmatrix} \rightarrow
\begin{bmatrix}
2 & 1 & 0 \\
0 & 1 & 1 \\
8 & 9 & 5 \\
6 & 9 & 8
\end{bmatrix} \rightarrow
\begin{bmatrix}
2 & 1 & 0 \\
0 & 1 & 1 \\
8 & 9 & 5 \\
0 & 9 & 8
\end{bmatrix}
\]

\[
\begin{bmatrix}
2 & 1 & 0 \\
0 & 1 & 1 \\
0 & 0 & 2 \\
0 & 4 & 6
\end{bmatrix} \rightarrow
\begin{bmatrix}
2 & 1 & 0 \\
0 & 1 & 1 \\
0 & 0 & 2 \\
0 & 0 & 2
\end{bmatrix} = U
\]

If we store the \(C_{ij}\), then we can use them to solve linear equations:

\[O_{m \times n} \cdots O_{1 \times 2} \cdot A = U\]

So solve \(Ax = b\) by applying these operators to \(b\),

giving \(Ux = O_{m \times n} \cdots O_{1 \times 2} \cdot b\)

Then solve for \(x\).

As an algorithm, this is:

for \(i\) in 1 to \(n-1\)
  for \(j\) in \(i+1\) to \(n\)
    \[b = O_{i \times j} \cdot b \iff b[i] = b[j] - C_{i,j} \cdot b[i]\]

Is same as forward solve in \(L\) with

\(L(i,i) = 1\) and \(L(j,i) = C_{i,j}\)
This is how we build $L$.

How many ops?

\[
\begin{aligned}
\text{for } i \text{ in } 1 \text{ to } n-1 \\
\quad \text{for } j \text{ in } i+1 \text{ to } n \\
\quad \quad \text{apply } O^c_{i,j} \text{ to } M - \text{ but only in cols } i+1 \text{ to } n
\end{aligned}
\]

\[
\sum_{i=1}^{n-1} (n-i)^2 \approx \frac{1}{3} n^3, \text{ so } \Theta(n^3) \text{ ops}
\]

Why not build $L^{-1}$? Seems to take $\Theta(n^3)$ ops, is wasteful and we do not need it.

What to do when $L(i,i) = 0$?

Pivot. Find $k$ s.t. $L(k,i) \neq 0$, and eliminate entries in column $i$ using $k$ th row.

Or, swap rows $k$ and $i$ (virtually)

Instead of $L$, construct $PL$ for a permutation matrix $L$.

A permutation matrix is a matrix $P$

that is all zeros except for one 1 in every row and column.

For each $i$, let $\pi(i)$ be s.t. $P[i,\pi(i)] = 1$

$P_{x} = x(\pi(1)), x(\pi(2)), \ldots, x(\pi(n))$,

a permutation of $x$. 
It is easy to multiply a vector by $P$—takes $\Theta(n)$ ops by using formula.

$P$ is orthogonal, so $P^T P = I \Rightarrow P^{-1} = P^T$.

$\Rightarrow$ is easy to apply $P^{-1}$.

We compute $P$ during the algorithm, and go back and adjust $c_{ij}$ at the end—more work if do it along the way.

Example: $A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ compute $LU$ for $P^T A$

$P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ $P^T A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$

But, this is not the only problem!

What if $A = \begin{bmatrix} \varepsilon & 1 \\ 0 & 1 \end{bmatrix}$ $L = \begin{bmatrix} 1 & 0 \\ \frac{\varepsilon}{2} & 1 \end{bmatrix}$ $U = \begin{bmatrix} \varepsilon & 1 \\ 0 & 1 - \frac{\varepsilon}{2} \end{bmatrix}$

If $\varepsilon < 10^{-16}$, rounding error gives $\tilde{U} = \begin{bmatrix} \varepsilon & 1 \\ 0 & -\frac{\varepsilon}{2} \end{bmatrix}$

and $L \tilde{U} = \begin{bmatrix} \varepsilon & 1 \\ 1 & 0 \end{bmatrix}$ a very different matrix.
So, pivot not just when $O$, but when small. Try to keep $L$ and $U$ small.

A theorem of Wilkinson says if compute with precision $u$,

\[ \|A - LU\| \leq u \cdot \Theta (\|L\| \|U\|) \]

Is a good reason to think about $L$ and $U$. 