Overview

- L2: cost
- L2: apps (L)
- L2 for interpolation

\[ Q; \text{ ill vs. well} \]

- P vs.

\[ 10^{-10} \leq 10^5 \cdot 10^{-15} \]

rel. out
\[ \text{cond} \text{#} \]
rel. inp. err

ill \[ \sim 10^5 \]

\[ 10^{-4} \]
Computational Cost

- What is the computational cost of multiplying two $n \times n$ matrices?

\[ O(n^3) \]

- What is the computational cost of carrying out LU factorization on an $n \times n$ matrix?

\[ n \cdot O(n^3) \rightarrow \text{not really } O(n^4) \]

\[ \begin{pmatrix} M & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix} \begin{pmatrix} n \cdot \mathbf{0} \end{pmatrix} \rightarrow \text{LU actually } O(n^3) \]

- M: any matrix $O(n^2)$
- P: any matrix $O(n^2)$
Demo: Complexity of Mat-Mat multiplication and LU

\[ n \text{ grows } \sim 10^6 \quad (1000 \times 1000 \sim 1\text{s}) \]

\[ n \text{ grows } \sim 10^9 \quad (10^9 \times 10^9 \sim (10^6)^3 \sim 1\text{s}) \]
More cost concerns

- What’s the cost of solving $Ax = b$?
- What’s the cost of solving $Ax = b_1, b_2, ..., b_n$?
- What’s the cost of finding $A^{-1}$?
Cost: Worrying about the Constant, BLAS

$O(n^3)$ really means

$$\alpha \cdot n^3 + \beta \cdot n^2 + \gamma \cdot n + \delta.$$ 

All the non-leading and constants terms swept under the rug. But: at least the leading constant ultimately matters.

Getting that constant to be small is surprisingly hard, even for something deceptively simple such as matrix-matrix multiplication.

**Idea:** Rely on library implementation: BLAS (Fortran)
Level 1: \( z = \alpha x + y \)  
vector-vector operations  
\( O(n) \)  
\(?axpy\)

Level 2: \( z = Ax + y \)  
matrix-vector operations  
\( O(n^2) \)  
\(?gemv\)

Level 3: \( C = AB + \beta C \)  
matrix-matrix operations  
\( O(n^3) \)  
\(?gemm\)

LAPACK: Implements ‘higher-end’ things (such as LU) using BLAS
Special matrix formats can also help save const significantly, e.g.

- banded
- sparse
LU: Special cases

- What happens if we feed a non-invertible matrix to LU?

- What happens if we feed LU an $m \times n$ non-square matrices?
In-class activity: LU factorization 2
9  LU: Applications
9.1 Linear Algebra Applications
Solve a Linear System

- LU factorization gives us

\[ PA = LU, \]

so that \( P \) is a permutation matrix, \( L \) is lower triangular, \( U \) is upper triangular. How does that help solve a linear system \( Ax = b \)?

\[
\begin{align*}
Ax &= b \\
P^T L U x &= b \\
P^T L y &= P b \\
y &= P b & (w \text{ subst}) \\
x &= y & (b \text{ subst})
\end{align*}
\]
Solve a Matrix Equation

- Suppose we want to solve $AX = B$.
  - $A$ and $B$ are given, $X$ is unknown.
  - (Assume: square and have same size) How can we do that using LU?

\[
\begin{pmatrix}
A & \vdots \\
\end{pmatrix}
\begin{pmatrix}
x_1 \\
\end{pmatrix}
= 
\begin{pmatrix}
b_1 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
A & \vdots \\
\end{pmatrix}
\begin{pmatrix}
x_2 \\
\end{pmatrix}
= 
\begin{pmatrix}
b_2 \\
\end{pmatrix}
\]

\text{(3) solve column by column}

\text{no: solve n linear systems: } O(n^4) \text{?}

\text{yes: calculate } L \text{ and } U: O(n^3)

\text{(2) } n \times \text{few/backsolves: } n \cdot O(n^3) = O(n^3)
Compute an Inverse

- Suppose we want to compute the inverse $A^{-1}$ of a matrix $A$. How do we do that using LU?

  $\text{solve } A \cdot x = I$

- What’s the computational cost of doing so?

  $O(n^3)$
Find the Determinant of a Matrix

- How can we find the determinant of a matrix using LU?

$PA = LU \implies A = P^T LU$

$\det(A) = \det(P^T) \cdot \det(L) \cdot \det(U)$

$\pm 1$

$0^{n \times n}$
Find Row Echelon Form... if we can?

- The factor $U$ in pivoted LU looks like it is in upper echelon form. Is it?

\[ \text{rank } (A) = \dim (\text{col space }) = \dim (\text{row space}) \]
Finding the Rank of a Matrix Numerically... if we can?

- Can we find the rank of a matrix numerically?

\[
\begin{pmatrix}
- & - \\
- & - \\
\end{pmatrix}
\]

To ask about rank, we need a tolerance.

\( \text{LU can't do tolerances} \) \( \rightarrow \) postpone.
9.2 Interpolation
Recap: Interpolation

Starting point: Looking for a linear combination of functions $\varphi_i$ to hit given data points $(x_i, y_i)$.

Interpolation becomes solving the linear system:

$$y_i = f(x_i) = \sum_{j=0}^{N_{\text{func}}} \alpha_j \varphi_j(x_i) \leftrightarrow V\alpha = y.$$  

Want unique answer: Pick $N_{\text{func}} = N \rightarrow V$ square.

$V$ is called the (generalized) Vandermonde matrix.

Main lesson:

$$V(\text{coefficients}) = (\text{values at nodes}).$$
Rethinking Interpolation

We have so far always used monomials ($1, x, x^2, x^3, ...$) and equispaced points for interpolation. It turns out that this has *significant problems*.

**Demo:** Monomial interpolation
Demo: Choice of Nodes for Polynomial Interpolation
Interpolation: Choosing Basis Function and Nodes

Both function basis and point set are under our control. What do we pick?

Ideas for basis functions:

- Monomials 1, x, x^2, x^3, x^4, ...
- Functions that make \( V = I \) → ‘Lagrange basis’
- Functions that make \( V \) triangular → ‘Newton basis’
- Splines (piecewise polynomials)
- Orthogonal polynomials
- Sines and cosines
- ‘Bumps’ (‘Radial Basis Functions’)

Ideas for nodes:

- Equispaced
- ‘Edge-Clustered’ (so-called Chebyshev/Gauss/... nodes)
Better Conditioning: Orthogonal Polynomials

- What caused monomials to have a terribly conditioned Vandermonde?
- What's a way to make sure two vectors are *not* like that?
- But polynomials are functions!

- But how can I practically compute the Legendre polynomials?