Today:

- make c
- integral operators $\rightarrow O(n^2)$
Matvec: A Slow Algorithm

Matrix-vector multiplication: our first ‘slow’ algorithm. $O(N^2)$ complexity.

$$\beta_i = \sum_{j=1}^{N} A_{ij} \alpha_j$$

Assume $A$ dense.
Matrices and Point Interactions

\[ A_{ij} = G(x_i, y_j) \]

Does that actually change anything?

\[ \psi_\theta(x_i) = \sum_{j=1}^{N} G(x_i, y_j) \varphi(y_j) \]

- \( x_i \rightarrow \) Targets / Observation points
- \( y_j \rightarrow \) Sources
- \( G(x, y) \rightarrow \) Kernel
Matrices and Point Interactions

\[ A_{ij} = G(x_i, y_j) \]

Graphically, too:
Matrices and point Interactions

\[ \psi(x_f) = \sum_{j=1}^{N} G(x_f, y_j) \varphi(y_j) \]

This *feels* different.

\[ \varphi(x) = \int \mathbb{R} G(x, y) \varphi(y) \]

Q: Are there enough matrices that come from globally defined \( G \) to make this worth studying?
Interpolation:
\[ \tilde{f}(x) = \sum_{j=1}^{N} l_j(x) \varphi(y_j) \]

Interpolation error:
\[ R_N = f(x) - \sum_{j=1}^{N} l_j(x) \varphi(y_j) \]

Num diff:
\[ \varphi(x) = \sum_{j=1}^{N} l_j'(x) \, \varphi(y_j) \]
Identify

\[ \varphi(x) = \int_{\mathbb{R}} \delta(x-y) \, \gamma(y) \, dy \]

\[ z_i^i = \sum_j A_{ij} z_j \]
Point Interaction Matrices: Examples (II)

\[ x = \sum_{y \rightarrow \text{in}} \sum_{x \rightarrow \text{out}} f(x, y) \]

Diagram: A matrix with a diagonal set to 1 and all other elements set to 0.
Point Interaction Matrices: Examples (III)

Convolutions
\[ \varphi(x) = \int G(x-y) \varphi(y) dy \]

Circular matrices

\[ G(x, y) = C \cdot \frac{1}{|x-y|} \quad \in \begin{cases} \Delta u = 0 \quad \text{fuld. solub.} \\ (30) \end{cases} \]

\[ C \cdot \log(|x-y|) \quad \in \mathcal{O} \]

So yes, there are indeed lots of these things.
Integral Operators

Why did we go through the trouble of rephrasing matvecs as

$$\psi(x_i) = \sum_{j=1}^{N} G(x_i, y_j) \phi(y_j)?$$
Cheaper Matvecs

\[ \psi(x_i) = \sum_{j=1}^{N} G(x_i, y_j) \varphi(y_j) \]

So what can we do to make evaluating this cheaper?

- Sparse
- Circulant / Toeplitz / convolutions
- Low rank
Fast Dense Matvecs

Consider

\[ A_{ij} = u_i v_j, \]

let \( \mathbf{u} = (u_i) \) and \( \mathbf{v} = (v_j) \).

Can we compute \( \mathbf{A} \mathbf{x} \) quickly? (for a vector \( \mathbf{x} \))

\[
\mathbf{A} \mathbf{x} = (\mathbf{u} \mathbf{v}^\top) \mathbf{x} = \hat{\mathbf{u}} (\mathbf{v}^\top \mathbf{x})
\]
Fast Dense Matvecs (II)

\[ A = \mathbf{u}_1 \mathbf{v}_1^T + \cdots + \mathbf{u}_K \mathbf{v}_K^T \]

Does this generalize? What is \( K \) here?

Cost: \( O(NK) \)
Low-Rank Point Interaction Matrices

Usable with low-rank complexity reduction?

\[ \psi(x_i) = \sum_{j=1}^{N} G(x_i, y_j) \varphi(y_j) \]

\[ A = u v^T \]

\[ G(x, y) = u(x) \cdot v(y) \]

L simple "compact operators"}

"compact" \( \Rightarrow \) "approximately finite-dim. range"
Numerical Rank

What would a *numerical* generalization of ‘rank’ look like?

\[ A \text{ has rank } k \text{ if } A \in \mathbb{R}^{m \times n} \]

\[ A = UV \]

\[ \begin{bmatrix} U & k \end{bmatrix} \]

\[ k \]

\[ \text{num rank} (A, \varepsilon) = \min \{ k : \exists U \in \mathbb{R}^{m \times k}, V \in \mathbb{R}^{k \times n} \quad \| A - UV \|_F \leq \varepsilon \} \]
Eckart-Young-Mirsky Theorem

**Theorem (Eckart-Young-Mirsky)**

\[ \text{SVD } A = U\Sigma V^T. \text{ If } k < r = \text{rank}(A) \text{ and } \]

\[ A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T, \]

then

\[ \min_{\text{rank}(B)=k} |A - B|_2 = |A - A_k|_2 = \sigma_{k+1} \]

**Q:** What’s that error in the Frobenius norm? So in principle that’s good news:

- We can find the numerical rank.
- We can also find a factorization that reveals that rank (!)

**Demo:** Rank of a Potential Evaluation Matrix (Attempt 2)
Constructing a tool

There is still a slight downside, though.

\[ \text{build: } O(N^2) \]
\[ \text{character: } O(N^3) \]
\[ \text{malvec: } O(N^k) \]
Representation

\[ Ax = b \quad \quad M \preceq A^{-1} \]

What does all this have to do with (right-)preconditioning?

\[
(MA) \quad \leftarrow \text{left preconditioning} \\
(AM) \quad \leftarrow \text{right preconditioning}
\]
Representation (in context)
Outline

Introduction

Dense Matrices and Computation

Tools for Low-Rank Linear Algebra
  Low-Rank Approximation: Basics
  Low-Rank Approximation: Error Control
  Reducing Complexity

Rank and Smoothness

Near and Far: Separating out High-Rank Interactions

Outlook: Building a Fast PDE Solver

Going Infinite: Integral Operators and Functional Analysis

Singular Integrals and Potential Theory

Boundary Value Problems

Back from Infinity: Discretization

Computing Integrals: Approaches to Quadrature

Going General: More PDEs
Rephrasing Low-Rank Approximations

SVD answers low-rank-approximation (‘LRA’) question. But: too expensive. First, rephrase the LRA problem:

\[ A = U V^T \in \text{factorization form} \]

\[ A \approx Q_k Q_k^T A \in \text{projection} \]
Using LRA bases

If we have an LRA basis $Q$, can we compute an SVD?

1. $B = QA^T$
2. $B = \tilde{U} \Sigma V^T$
3. $A - OB = Q \tilde{U} \Sigma V^T$
Finding an LRA basis

How would we *find* an LRA basis?

**Goal:** Find Q columns

\[ \| A - QQ^T A \|_2 \leq \varepsilon \]

- Fixed-rank approximation
- Adaptive CCA

**Ideal:** SVD

**Ideal 2:** Randomized algorithm for rangefinding
Giving up optimality

What problem should we actually solve then?

\[ \| A - QQ^T A \|_2 = \min_{\text{rank}(X) \leq k} \| A - X \|_2 = \sigma_{k+1} \]

\[ \| A - QQ^T A \|_2 = \min_{\text{rank}(X) \leq k} \| A - X \|_2 = \sigma_{k+1} + \epsilon \]
Recap: The Power Method

How did the power method work again?