COMPSCI 514: Algorithms for Data Science

Cameron Musco

University of Massachusetts Amherst. Spring 2026.

Lecture 18

Logistics

- Midterm **in class next Thursday 4/23**.
- See Piazza/last lecture/midterm study guide for details.
- Will cover material up to this lecture.
- The quiz this week is **optional**. If you take it, it will replace your second lowest regular quiz grade (the lowest is dropped).

Summary

Last Few Classes: Spectral Graph Partitioning

- Focus on separating graphs with small but relatively balanced cuts.
- Connection to second smallest eigenvector of graph Laplacian.
- Provable guarantees for stochastic block model.
- Expectation analysis in class. Quick sketch of full analysis.

This Class: Computing the SVD/eigendecomposition.

- Efficient algorithms for SVD/eigendecomposition.
- Iterative methods: power method, Krylov subspace methods.
- High level: a glimpse into fast methods for linear algebraic computation, which are workhorses behind modern data science/ML.

Efficient Eigendecomposition and SVD

We have talked about the eigendecomposition and SVD as ways to compress data, to embed entities like words and documents, to compress/cluster non-linearly separable data and graphs.

How efficient are these techniques? Can they be run on very large datasets?

Computing the SVD

Basic Algorithm: To compute the SVD of full-rank $\mathbf{X} \in \mathbb{R}^{n \times d}$, $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$:

- Compute $\mathbf{X}^T\mathbf{X}$ – $O(nd^2)$ runtime.
- Find eigendecomposition $\mathbf{X}^T\mathbf{X} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$ – $O(d^3)$ runtime.
- Compute $\mathbf{L} = \mathbf{X}\mathbf{V}$ – $O(nd^2)$ runtime. Note that $\mathbf{L} = \mathbf{U}\mathbf{\Sigma}$.
- Set $\sigma_i = \|\mathbf{L}_i\|_2$ and $\mathbf{U}_i = \mathbf{L}_i / \|\mathbf{L}_i\|_2$. – $O(nd)$ runtime.

  Total runtime: $O(nd^2 + d^3) = O(nd^2)$ (assume w.l.o.g. $n \geq d$)
- If we have $n = 10$ million images with $200 \times 200 \times 3 = 120,000$ pixel values each, runtime is 1.5×10^{17} operations!
- The worlds fastest super computers compute at ≈ 100 petaFLOPS = 10^{17} FLOPS (floating point operations per second).
- This is a relatively easy task for them – but no one else.

Faster Algorithms

To speed up SVD computation we will take advantage of the fact that we typically only care about computing the top (or bottom) k singular vectors of a matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ for $k \ll d$.

- Suffices to compute $\mathbf{V}_k \in \mathbb{R}^{d \times k}$ and then compute $\mathbf{U}_k \mathbf{\Sigma}_k = \mathbf{X}\mathbf{V}_k$.
- Use an iterative approximation algorithm to compute an approximation to the top k singular vectors $\mathbf{V}_k$ (the top k eigenvectors of $\mathbf{X}^T\mathbf{X}$.)
- Runtime will be roughly $O(ndk)$ instead of $O(nd^2)$.

Sparse (iterative) vs. Direct Method. `svd` vs. `svds`.

Power Method

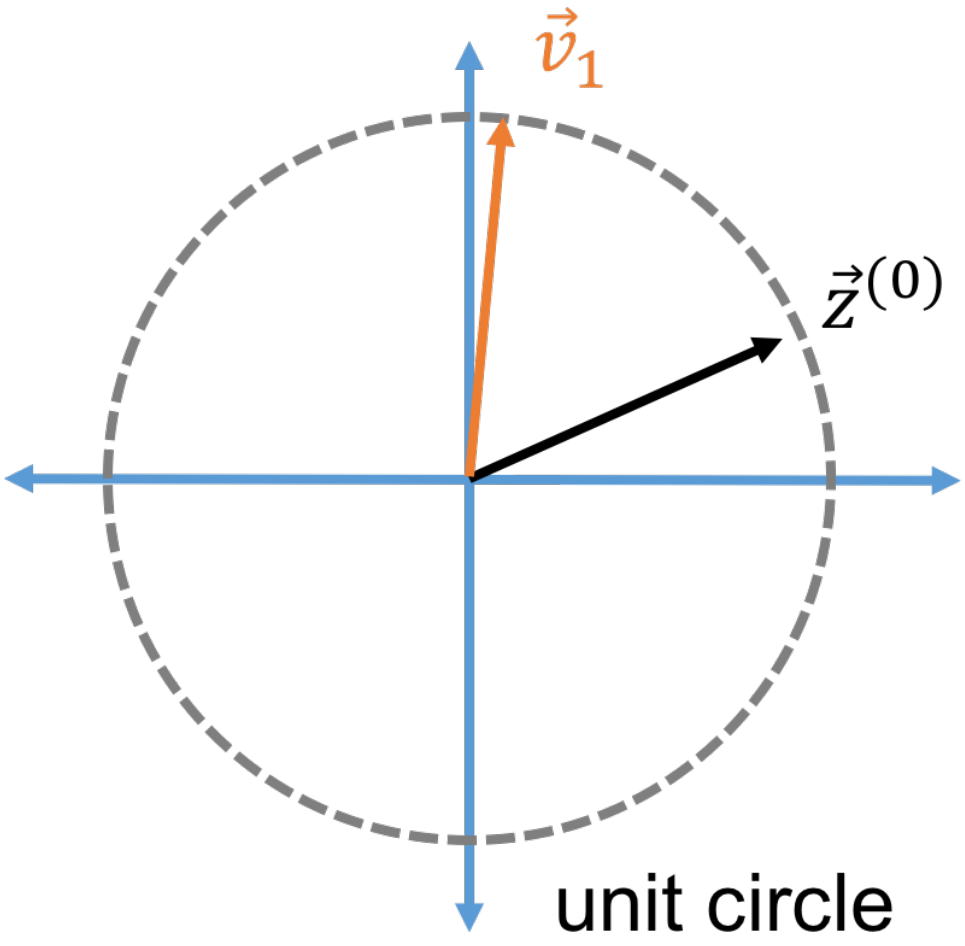
Power Method: The most fundamental iterative method for approximate SVD/eigendecomposition. Applies to computing $k = 1$ eigenvectors, but can be generalized to larger k.

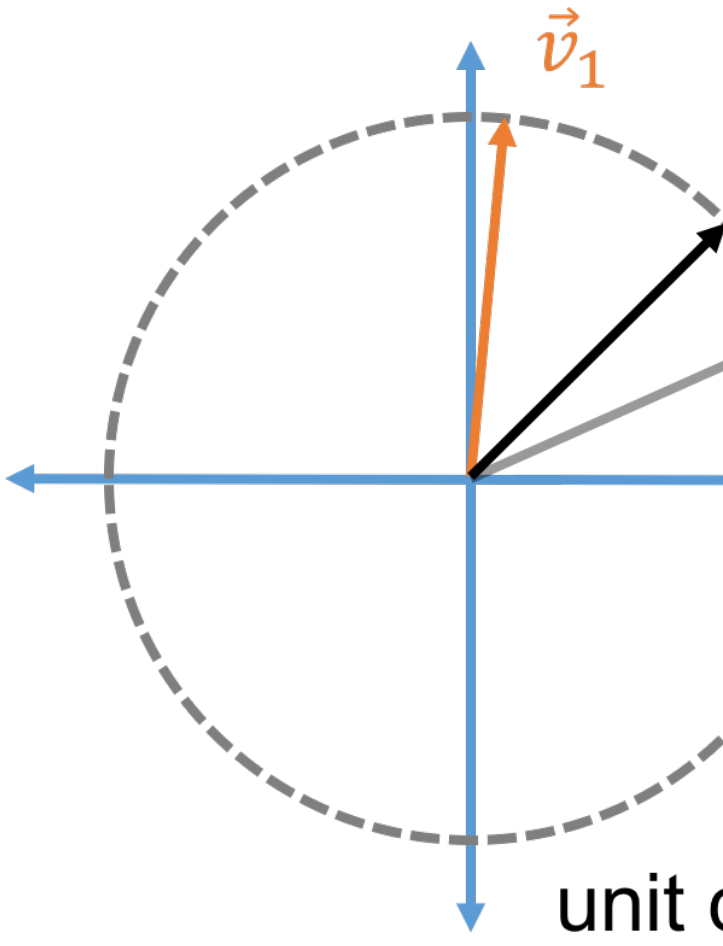
Goal: Given symmetric $\mathbf{A} \in \mathbb{R}^{d \times d}$, with eigendecomposition $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$, find $\vec{z} \approx \vec{v}_1$. I.e., the top eigenvector of $\mathbf{A}$.

Apply to $\mathbf{A} = \mathbf{X}^T\mathbf{X}$ to approximate the top right singular vector of $\mathbf{X}$.

- **Initialize:** Choose $\vec{z}^{(0)}$ randomly. E.g. $\vec{z}^{(0)}(i) \sim \mathcal{N}(0, 1)$.
- For $i = 1, \ldots, t$
  - $\vec{z}^{(i)} := \mathbf{A} \cdot \vec{z}^{(i-1)}$
  - $\vec{z}_i := \frac{\vec{z}^{(i)}}{\|\vec{z}^{(i)}\|_2}$
- Return $\vec{z}_t$

Power Method





Power Method Analysis

Power method:

- **Initialize:** Choose $\vec{z}^{(0)}$ randomly. E.g. $\vec{z}^{(0)}(i) \sim \mathcal{N}(0,1)$.
- For $i = 1, \ldots, t$
  - $\vec{z}^{(i)} := \mathbf{A} \cdot \vec{z}^{(i-1)}$
  - $\vec{z}_i := \frac{\vec{z}^{(i)}}{\|\vec{z}^{(i)}\|_2}$
- Return $\vec{z}_t$.

Theoretically equivalent to:

- For $i = 1, \ldots, t$
  - $\vec{z}^{(i)} := \mathbf{A} \cdot \vec{z}^{(i-1)}$
- $\vec{z}_i := \frac{\vec{z}^{(i)}}{\|\vec{z}^{(i)}\|_2}$.
- Return $\vec{z}_t$.

Power Method Analysis

Write $\vec{z}^{(0)}$ in $\mathbf{A}$'s eigenvector basis:

$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d = \mathbf{V}\mathbf{c}.$$

Update step: $\vec{z}^{(i)} = \mathbf{A} \cdot \vec{z}^{(i-1)} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T \cdot \vec{z}^{(i-1)}$ (then normalize)

$$\mathbf{V}^T\vec{z}^{(0)} =$$

$$\mathbf{\Lambda}\mathbf{V}^T\vec{z}^{(0)} =$$

$$\vec{z}^{(1)} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T \cdot \vec{z}^{(0)} =$$

$\mathbf{A} \in \mathbb{R}^{d\times d}$: input matrix with eigendecomposition $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$. $\vec{v}_1$: top eigenvector, being computed, $\vec{z}^{(i)}$: iterate at step i, converging to $\vec{v}_1$.

Power Method Analysis

Claim 1 : Writing $\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d$,

$$\vec{z}^{(1)} = c_1 \cdot \lambda_1\vec{v}_1 + c_2 \cdot \lambda_2\vec{v}_2 + \ldots + c_d \cdot \lambda_d\vec{v}_d.$$

$$\vec{z}^{(2)} = \mathbf{A}\vec{z}^{(1)} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T\vec{z}^{(1)} =$$

Claim 2:

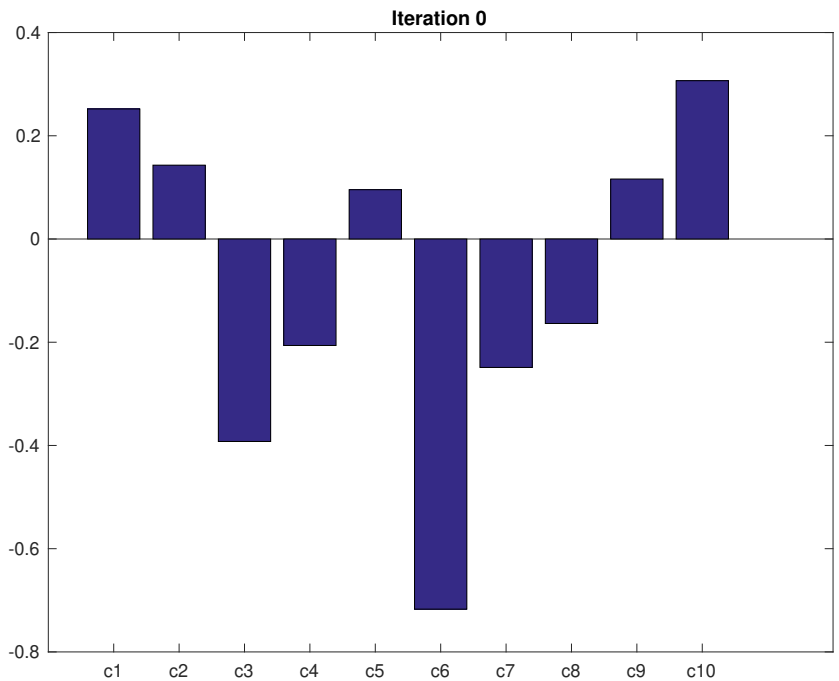
$$\vec{z}^{(t)} = c_1 \cdot \lambda_1^t\vec{v}_1 + \mathbf{c}_2 \cdot \lambda_2^t\vec{v}_2 + \ldots + c_d \cdot \lambda_d^t\vec{v}_d.$$

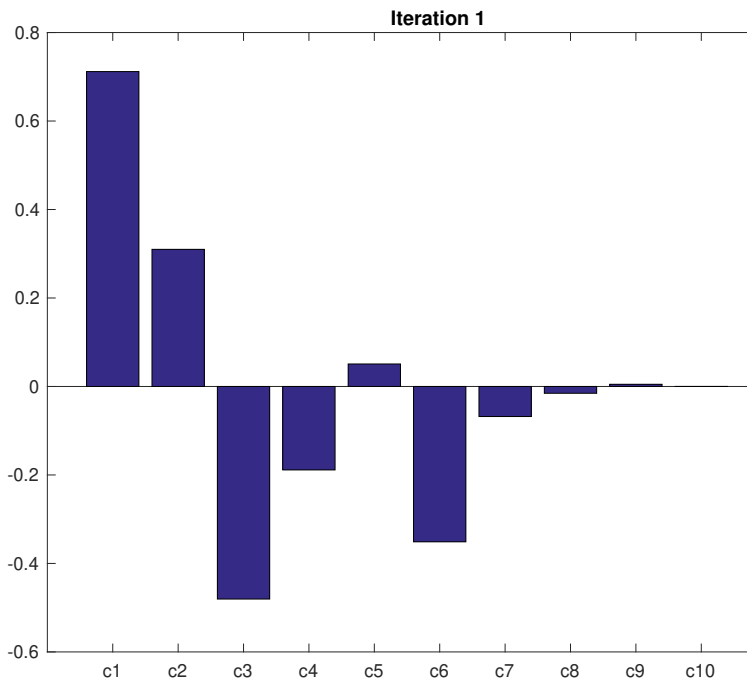
$\mathbf{A} \in \mathbb{R}^{d\times d}$: input matrix with eigendecomposition $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$. $\vec{v}_1$: top eigenvector, being computed, $\vec{z}^{(i)}$: iterate at step i, converging to $\vec{v}_1$.

Power Method Convergence

After t iterations, we have 'powered' up the eigenvalues, making the component in the direction of v_1 much larger, relative to the other components.

$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



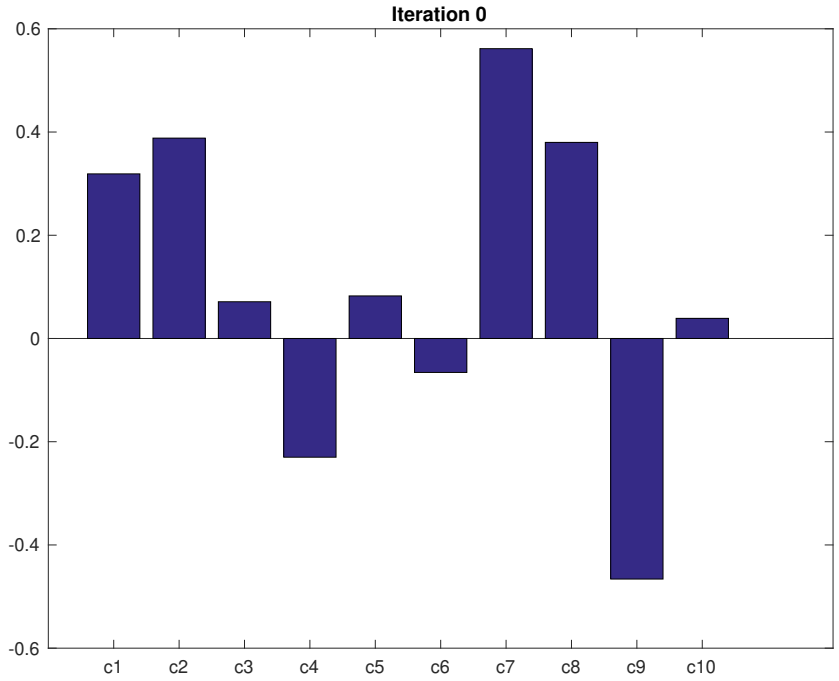


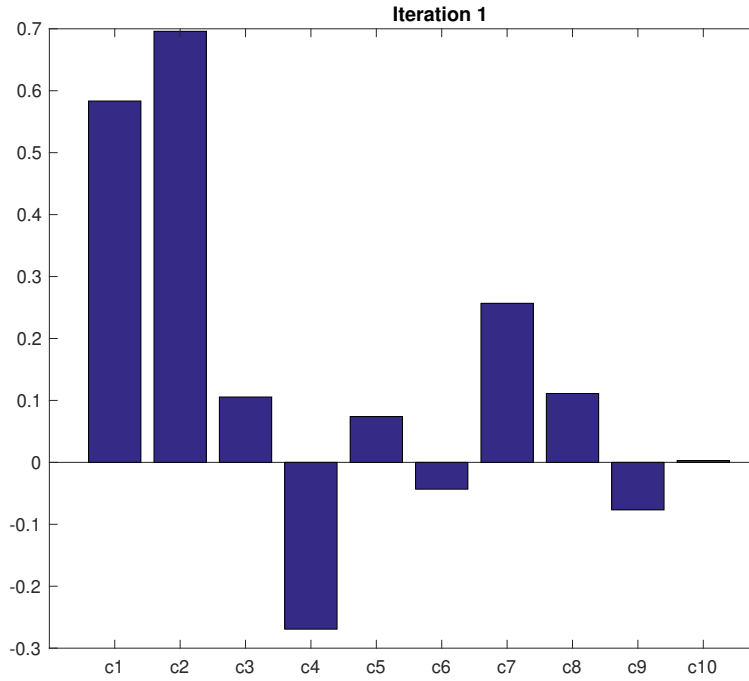
When will convergence be slow?

Power Method Slow Convergence

Slow Case: A has eigenvalues: $\lambda_1 = 1, \lambda_2 = .99, \lambda_3 = .9, \lambda_4 = .8, \ldots$

$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$





Power Method Convergence Rate

$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$

Write $|\lambda_2| = (1-\gamma)|\lambda_1|$ for 'gap' $\gamma = \frac{|\lambda_1|-|\lambda_2|}{|\lambda_1|}$.

How many iterations t does it take to have $|\lambda_2|^t \leq \delta \cdot |\lambda_1|^t$ for $\delta > 0$?

$$\begin{aligned} |\lambda_2|^t &= (1-\gamma)^t \cdot |\lambda_1|^t \\ &= (1-\gamma)^{1/\gamma \cdot \gamma t} \cdot |\lambda_1|^t \\ &\leq e^{-\gamma t} \cdot |\lambda_1|^t \end{aligned}$$

So it suffices to set $\gamma t = \ln(1/\delta)$. Or $t = \frac{\ln(1/\delta)}{\gamma}$.

How small must we set δ to ensure that $c_1\lambda_1^t$ dominates all other components and so $\vec{z}^{(t)}$ is very close to $\vec{v}_1$?

$\vec{v}_1$: top eigenvector, being computed, $\vec{z}^{(i)}$: iterate at step i, converging to $\vec{v}_1$. $\lambda_1, \lambda_2, \ldots \lambda_n$: eigenvalues of $\mathbf{A}$, $\gamma = \frac{|\lambda_1|-|\lambda_2|}{|\lambda_1|}$: eigengap controlling convergence rate

Random Initialization

Claim: When $z^{(0)}$ is chosen with random Gaussian entries, writing $z^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d$, with very high probability, for all i:

$$O(1/d^2) \leq |c_i| \leq O(\log d)$$

Corollary:

$$\max_j \left|\frac{c_j}{c_1}\right| \leq O(d^2 \log d).$$

$\mathbf{A} \in \mathbb{R}^{d\times d}$: input matrix with eigendecomposition $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$. $\vec{v}_1$: top eigenvector, being computed, $\vec{z}^{(i)}$: iterate at step i, converging to $\vec{v}_1$.

Random Initialization

Claim 1: When $z^{(0)}$ is chosen with random Gaussian entries, writing $z^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d$, with very high probability, $\max_j \left|\frac{c_j}{c_1}\right| \leq O(d^2 \log d)$.

Claim 2: For gap $\gamma = \frac{|\lambda_1| - |\lambda_2|}{|\lambda_1|}$, and $t = \frac{\ln(1/\delta)}{\gamma}$, $\left|\frac{\lambda_i^t}{\lambda_1^t}\right| \leq \delta$ for all i.

$$\vec{z}^{(t)} := \frac{c_1\lambda_1^t\vec{v}_1 + \ldots + c_d\lambda_d^t\vec{v}_d}{\|c_1\lambda_1^t\vec{v}_1 + \ldots + c_d\lambda_d^t\vec{v}_d\|_2} \implies$$

$$\|\vec{z}^{(t)} - \vec{v}_1\|_2 \leq \left\| \frac{c_1\lambda_1^t\vec{v}_1 + \ldots + c_d\lambda_d^t\vec{v}_d}{\|c_1\lambda_1^t\vec{v}_1\|_2} - \vec{v}_1 \right\|_2$$

$$= \left\| \frac{c_2\lambda_2^t}{c_1\lambda_1^t}\vec{v}_2 + \ldots + \frac{c_d\lambda_d^t}{c_1\lambda_1^t}\vec{v}_d \right\|_2 = \left|\frac{c_2\lambda_2^t}{c_1\lambda_1^t}\right| + \ldots + \left|\frac{c_d\lambda_d^t}{c_1\lambda_1^t}\right| \leq \delta \cdot O(d^2 \log d) \cdot d.$$

Setting $\delta = O\left(\frac{\epsilon}{d^3 \log d}\right)$ gives $\|\vec{z}^{(t)} - \vec{v}_1\|_2 \leq \epsilon$.

$A \in \mathbb{R}^{d\times d}$: input with eigenvalues $\lambda_1 \ldots, \lambda_d$ and eigenvectors $\vec{v}_1, \ldots, \vec{v}_d$. $\vec{z}^{(i)}$: iterate at step i. $c_1, \ldots, c_d$: coefficients of $\vec{z}^{(0)}$ in the eigenvector basis.

Power Method Theorem

Theorem (Basic Power Method Convergence)
Let $\gamma = \frac{|\lambda_1| - |\lambda_2|}{|\lambda_1|}$ be the relative gap between the first and second eigenvalues. If Power Method is initialized with a random Gaussian vector $\vec{v}^{(0)}$ then, with high probability, after $t = O\left(\frac{\ln(d/\epsilon)}{\gamma}\right)$ steps:

$$\|\vec{z}^{(t)} - \vec{v}_1\|_2 \leq \epsilon.$$

Total runtime: $O(t)$ matrix-vector multiplications. If $\mathbf{A} = \mathbf{X}^T\mathbf{X}$:

$$O\left(\text{mv}(\mathbf{X}) \cdot \frac{\ln(d/\epsilon)}{\gamma} \cdot\right) = O\left(nd \cdot \frac{\ln(d/\epsilon)}{\gamma}\right).$$

How is ϵ dependence?

How is γ dependence?

Krylov Subspace Methods

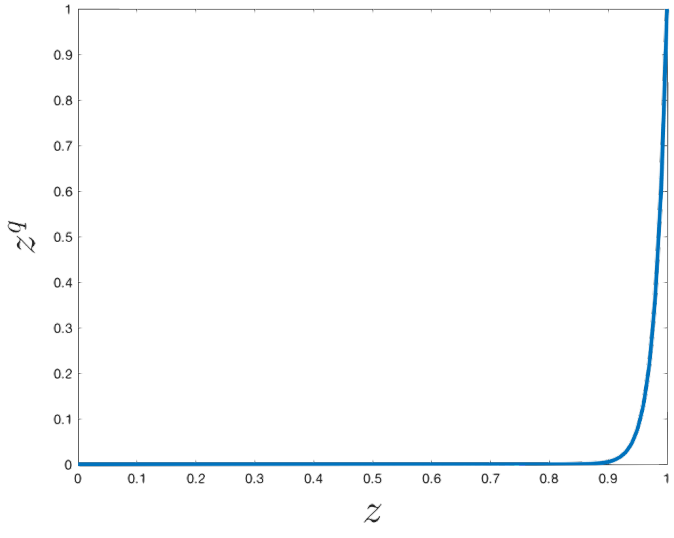
Krylov subspace methods (Lanczos method, Arnoldi method.)

- How `svds`/`eigs` are actually implemented. Only need $t = O\left(\frac{\ln(d/\epsilon)}{\sqrt{\gamma}}\right)$ steps for the same guarantee.

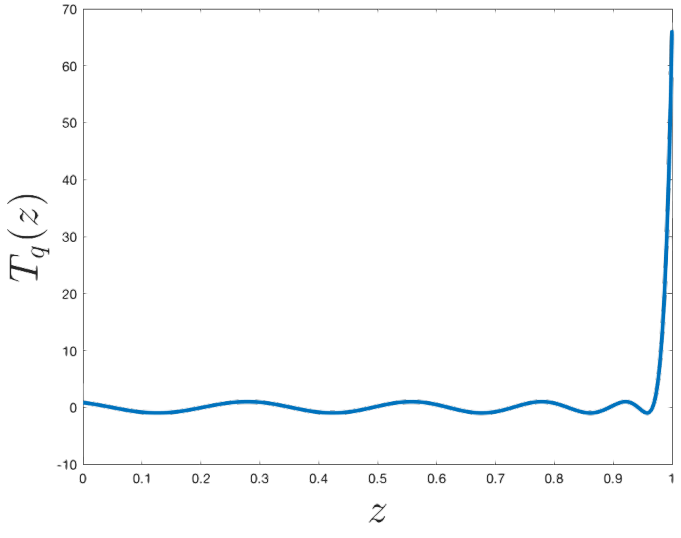
Main Idea: Need to separate λ_1 from λ_i for $i \geq 2$.

- Power method: power up to λ_1^t and λ_i^t.
- Krylov methods: apply a better degree t polynomial $T_t(\cdot)$ to the eigenvalues to separate $T_t(\lambda_1)$ from $T_t(\lambda_i)$.
- Still requires just t matrix vector multiplies. Why?

krylov subspace methods



vs.



Optimal 'jump' polynomial in general is given by a degree t Chebyshev polynomial. Krylov methods find a polynomial tuned to the input matrix that does at least as well.

Generalizations to Larger k

- Block Power Method (a.k.a. Simultaneous Iteration, Subspace Iteration, or Orthogonal Iteration)
- Pick random block of vectors $\mathbf{Z}^{(0)} \in \mathbb{R}^{n \times k}$. $\mathbf{Z}^{(i)} = \mathsf{orth}(\mathbf{A}\mathbf{Z}^{(i-1)})$.
- Block Krylov methods use an analogous approach.

$$\text{Runtime: } O\left(ndk \cdot \frac{\ln(d/\epsilon)}{\sqrt{\gamma}}\right)$$

to accurately compute the top k singular vectors.

$$\text{'Gapless' Runtime: } O\left(ndk \cdot \frac{\ln(d/\epsilon)}{\sqrt{\epsilon}}\right)$$

if you just want a set of vectors that gives an ϵ-optimal low-rank approximation when you project onto them.