COMPSCI 514: Algorithms for Data Science

Cameron Musco
University of Massachusetts Amherst. Fall 2024.
Lecture 20

Logistics

- Problem Set 4 is due 11/25.
- See Piazza for some updates/clarifications on Problem 1.
- No class or quiz next week.
- Additional office hours Friday 10am.

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 data science.

1 Multiple Choice 1 point

Consider $X \in \mathbb{R}^{n\times d}$. Let $U_k \in \mathbb{R}^{n\times k}$ and $V_k \in \mathbb{R}^{d\times k}$ contain its top k left and right singular vectors respectively.

When do we have $U_k U_k^T X = X V_k V_k^T$?

- When $U_k = V_k$.
- Always
- Never
- When X is symmetric.
- When X is symmetric with non-negative eigenvalues.

$X = U \Sigma V^T \rightarrow U_k U_k^T X = U_k \Sigma_k V_k^T = X V_k V_k^T$

($n\times k$, $k\times k$, $k\times d$)

best rank-k approximation of X

$X = U \Sigma V^T$

right s.v.s

left s.v.s.

columns: $\mathbb{R}^n$ U_k

rows: $\mathbb{R}^d$ V_k

Quiz Review

2 Multiple Choice 1 point

Under what conditions is the SVD of X equal to the eigendecomposition of X?

- ◯ X is symmetric.
- ◯ X has integer entries.
- ◯ X is symmetric and has non-negative eigenvalues.
- ◯ X is square and has non-negative entries.

$X \in \mathbb{R}^{n \times d}$

$X = U\Sigma V^T$

$X = U \Sigma S S V^T$ (orthogonal, orthogonal)

$\begin{bmatrix} -1 & & & \\ & 1 & & \\ & & 1 & \\ & & & -1 \end{bmatrix}$

$X \in \mathbb{R}^{d \times d}$ symmetric.

singular values equal magnitudes of eigenvalues

$X = U \Sigma V^T$

$X = V \Lambda V^T$

if $\lambda_i(X) \geq 0$, Λ is non negative

3 Multiple Answer 1 point

Which of the follow properties of the graph Laplacian for an undirected, unweighted graph always hold? Select all that apply.

- [x] It is symmetric.
- [ ] All if its entries are non-negative.
- [x] For any vector $v, v^T L v \geq 0$.
- [x] All if its eigenvalues are non-negative.
- [ ] It has at most two entries per row and column.

$D - A$ (adjacency matrix) $A_{ij} = A_{ji}$

$v = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$ $v^T L v = 0$

$v^T L v = \sum_{(i,j) \in E} [v(i) - v(j)]^2$

(A) $\Rightarrow$ (B)

v be an eigenvector of L

$Lv = \lambda \cdot v$

$v^T L v = v^T (\lambda \cdot v) = \lambda \cdot v^T v = \lambda$

$\|v\|_2^2 = 1$

$v^T L v = \lambda$

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.

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

Quiz Q7:
which vectors can be used to recover communities in SBM:

↳ $v_{n-1}(L)$ (second smallest e.v. of L)

$v_2(A)$ (second largest e.v. of A)

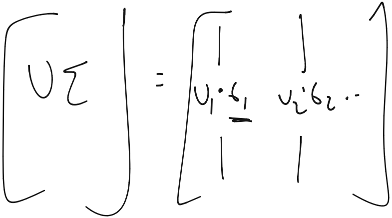
$$\mathbb{E}[L] = \mathbb{E}[D] - \mathbb{E}[A]$$

$$= \left(\frac{p+q}{2}\right) n I - \mathbb{E}[A]$$

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. $n \geq d$)



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!

Computing the SVD

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

- Compute $\mathbf{X}^T\mathbf{X}$ – $O(nd^2)$ runtime.
- Find eigendecomposition $\mathbf{X}^T\mathbf{X} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T$ – $O(d^3)$ runtime.
- Compute $\mathbf{L} = \mathbf{X}\mathbf{V}$ – $O(nd^2)$ runtime. Note that $\mathbf{L} = \mathbf{U}\boldsymbol{\Sigma}$.
- 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 $\approx$ 100 petaFLOPS = 10^{17} FLOPS (floating point operations per second).
- This is a relatively easy task for them – but no one else.

$X = U\Sigma V^T$

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 $\mathbf{U}_k \boldsymbol{\Sigma}_k = \mathbf{X}\mathbf{V}_k$.
- Use an *iterative 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 $O(nd^2)$.

input size

Faster Algorithms

Oja's algorithm

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 $\mathbf{U}_k \boldsymbol{\Sigma}_k = \mathbf{X}\mathbf{V}_k$.
- Use an *iterative 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`.

"exact" eig eigs

iterative
fast
take k as input

idea behind "Krylov methods" ← SVDs, eigs

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 $\mathbf{A}$.

$Av_1 = \lambda_1 v_1$

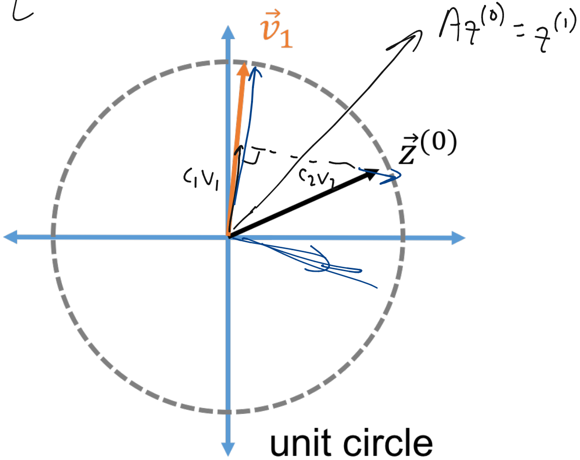
Power Method

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

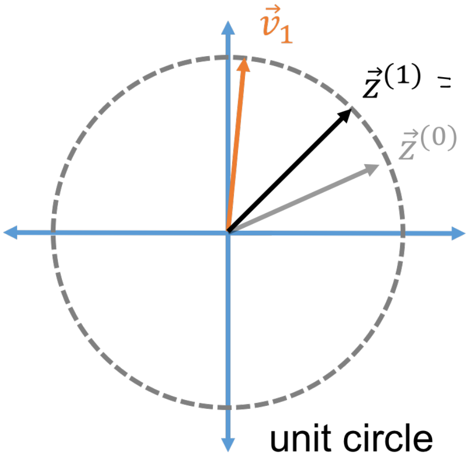
Goal: Given symmetric $\mathbf{A} \in \mathbb{R}^{d \times d}$, with eigendecomposition $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$, find $\vec{z} \approx \vec{v}_1$. I.e., the top eigenvector of $\mathbf{A}$.

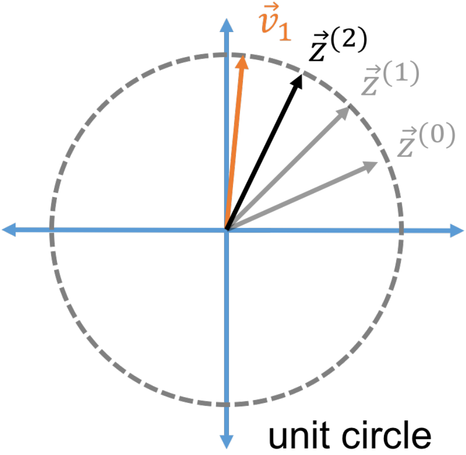
- **Initialize:** Choose $\vec{z}^{(0)}$ randomly. E.g. $\vec{z}^{(0)}(i) \sim \mathcal{N}(0, 1)$.
- For $i = 1, \ldots, t$
  - $\vec{z}^{(i)} := \mathbf{A} \cdot \vec{z}^{(i-1)}$
  - $\vec{z}^{(i)} := \frac{\vec{z}^{(i)}}{\|\vec{z}^{(i)}\|_2}$
- Return $\vec{z}^{(t)}$

A is 2x2



Power Method




$\vec{v}_1$
$\vec{z}^{(2)}$
$\vec{z}^{(1)}$
$\vec{z}^{(0)}$
unit circle

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 $\vec{z}_t$.

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:

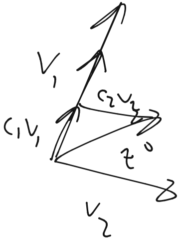
- For $i = 1, \ldots, t$
  - $\vec{z}^{(i)} := \mathbf{A} \cdot \vec{z}^{(i-1)}$
- $\vec{z}_i := \frac{\vec{z}^{(i)}}{\|\vec{z}^{(i)}\|_2}$.
- Return $\vec{z}_t$.

} in practice don't do this due to round off/ overflow error.

Power Method Analysis

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

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



$\mathbf{A} \in \mathbb{R}^{d\times d}$: 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 Analysis

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

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

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

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

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

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

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

Power Method Analysis

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

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

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

Power Method Analysis

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

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

$$\vec{z}^{(2)} = \mathbf{A}\vec{z}^{(1)} = \mathbf{V\Lambda V}^T\vec{z}^{(1)} = c_1\lambda_1^2 v_1 + c_2\lambda_2^2 v_2 + \ldots$$

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

Power Method Analysis

Claim 1 : Writing $\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d$, $\simeq V_1, -V_1$

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

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

Claim 2:

very big — relatively small

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

$$|\lambda_1^t| >> |\lambda_i^t|$$

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

Power Method Convergence

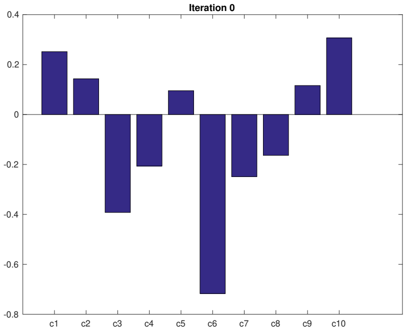
After t iterations, we have 'powered' up the eigenvalues, making the component in the direction of v_1 much larger, relative to the other components.

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

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.

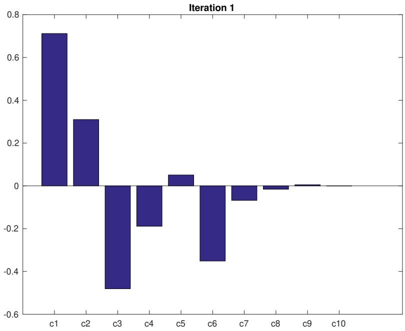
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



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.

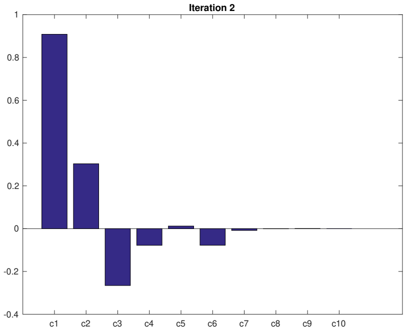
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



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.

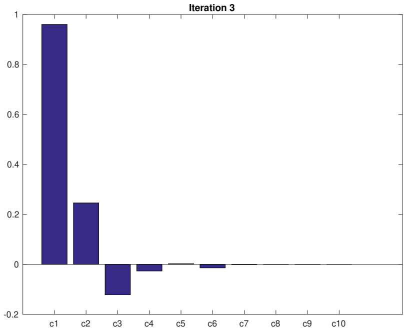
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



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.

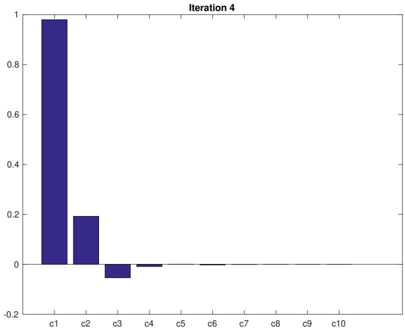
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



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.

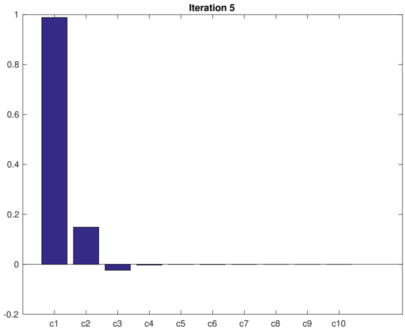
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



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.

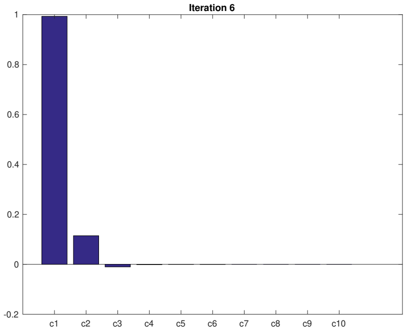
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



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.

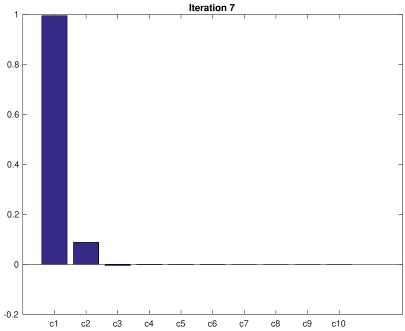
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



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.

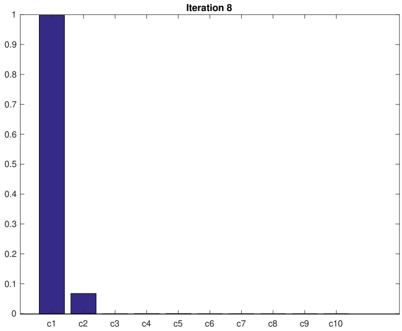
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



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.

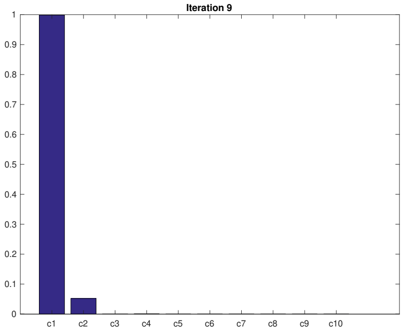
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



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.

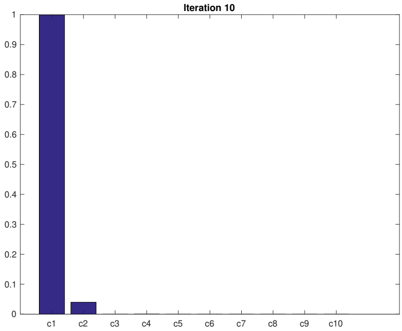
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



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?

$\lambda_1 \approx \lambda_2$

Power Method Slow Convergence

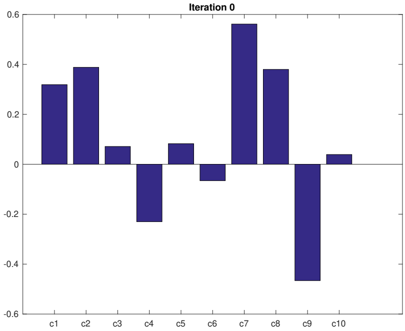
Slow Case: A has eigenvalues: $\lambda_1 = 1, \lambda_2 = .99, \lambda_3 = .9, \lambda_4 = .8, \ldots$

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

Power Method Slow Convergence

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

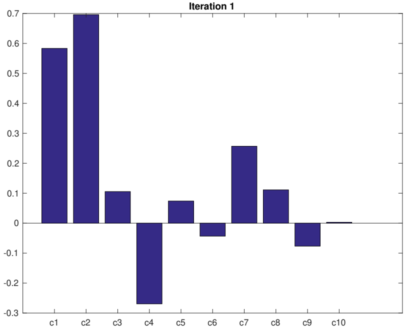
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



Power Method Slow Convergence

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

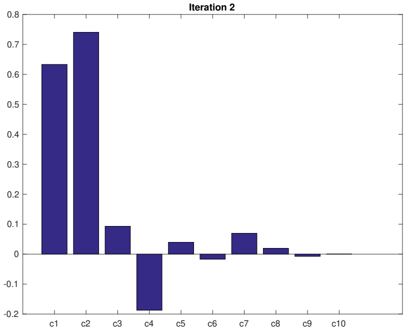
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



Power Method Slow Convergence

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

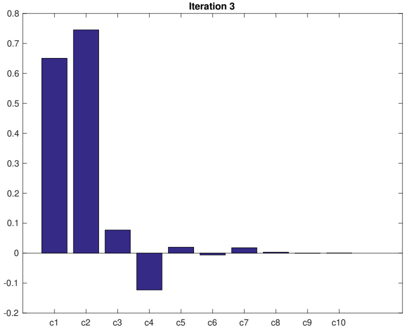
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



Power Method Slow Convergence

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

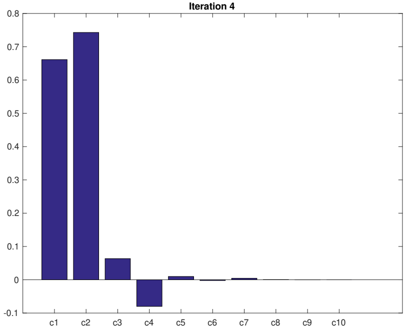
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



Power Method Slow Convergence

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

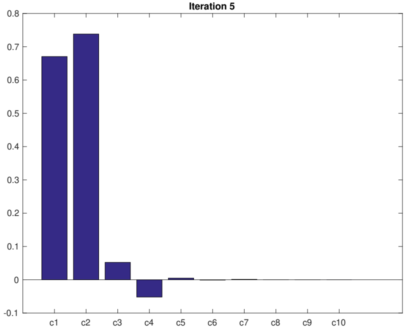
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



Power Method Slow Convergence

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

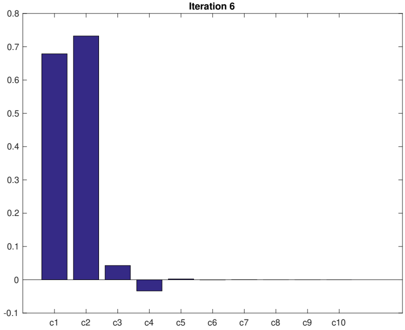
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



Power Method Slow Convergence

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

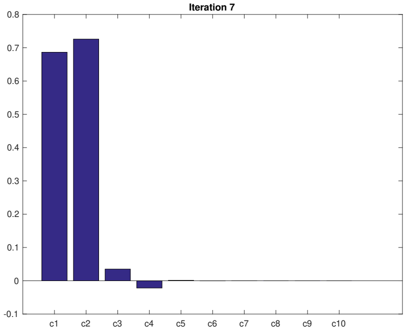
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



Power Method Slow Convergence

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

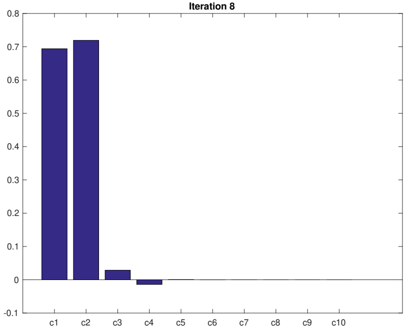
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



Power Method Slow Convergence

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

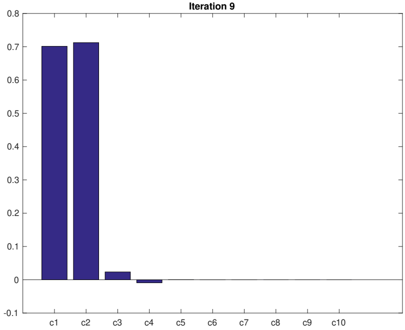
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



Power Method Slow Convergence

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

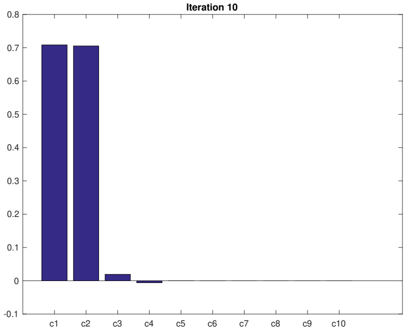
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



Power Method Slow Convergence

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

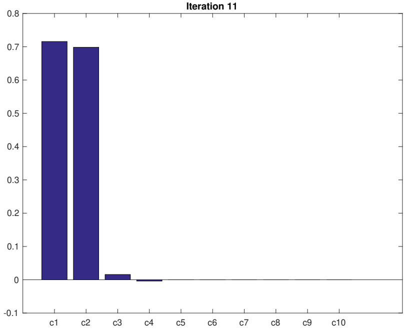
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



Power Method Slow Convergence

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

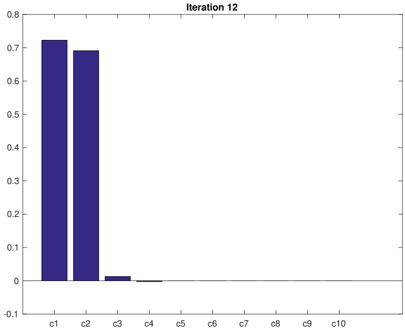
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



Power Method Slow Convergence

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

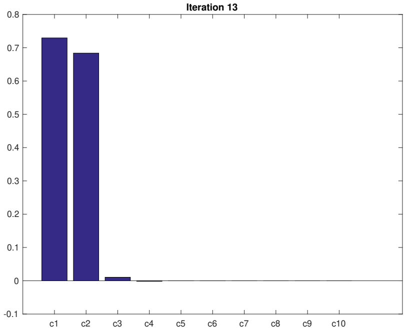
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



Power Method 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 $\delta > 0$?

$\vec{v}_1$: 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