COMPSCI 514: ALGORITHMS FOR DATA SCIENCE

Cameron Musco

University of Massachusetts Amherst. Fall 2021.

Lecture 21

LOGISTICS

due date office hours link broken

- I released Problem Set 4 this am. Due Wednesday 12/1.
- At least one group is looking to add a third teammate. if you are working alone and would like to join a group for Problem Set 4, let me know.
- Problem Set 5 will be released right after break and will be optimal extra credit.
- No quiz this week.
- This is the last day of our spectral unit. Then will have 3-4 classes on optimization + a possible review class before the end of the semester.
- I am holding hybrid office hours today at 5-6pm, so feel free to come by in person if you like.

CS 234.

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.
- Idealized analysis in class. See slides for 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.

PAP^T Pv_2 v s.t $\mathbb{E}[A]v = \lambda v$ $\mathbb{E}[L]v = \left(\frac{p+q}{2} \cdot n - \lambda\right)v$

Upshot: The second smallest eigenvector of $\mathbb{E}[\mathbf{L}]$ is $\chi_{B,C}$ – the indicator vector for the cut between the communities.

- If the random graph G (equivalently $\mathbf{A}$ and $\mathbf{L}$) were exactly equal to its expectation, partitioning using this eigenvector (i.e., spectral clustering) would exactly recover the two communities B and C.

$$\mathbb{E}[A] = \begin{bmatrix} p & q \\ q & p \end{bmatrix} \quad (n/2)$$

$$v_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \quad \text{second smallest eigenvector of } L.$$

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

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

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

v eigenvector of $\mathbb{E}[A]$ with eigenvalue λ $\mathbb{E}[A]v = \lambda v$

Upshot: The second smallest eigenvector of $\mathbb{E}[\mathbf{L}]$ is $\chi_{B,C}$ – the indicator vector for the cut between the communities.

- If the random graph G (equivalently $\mathbf{A}$ and $\mathbf{L}$) were exactly equal to its expectation, partitioning using this eigenvector (i.e., spectral clustering) would exactly recover the two communities B and C.

How do we show that a matrix (e.g., $\mathbf{A}$) is close to its expectation? Matrix concentration inequalities.

- Analogous to scalar concentration inequalities like Markovs, Chebyshevs, Bernsteins.
- Random matrix theory is a very recent and cutting edge subfield of mathematics that is being actively applied in computer science, statistics, and ML.

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 massive datasets?

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

X^TX $d \times d$ XX^T $n \times n$

- Compute $\mathbf{X}^T\mathbf{X}$ – $O(nd^2)$ runtime.
- Find eigendecomposition $\mathbf{X}^T\mathbf{X} = \mathbf{V\Lambda V}^T$ – $O(d^3)$ runtime.

  $n \times d$ $d \times d$ Σ^2
- Compute $\mathbf{L} = \mathbf{XV}$ – $O(nd^2)$ runtime. Note that $\mathbf{L} = \mathbf{U\Sigma}$.

  $XV = U\Sigma V^T V = U\Sigma$
- 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. $n \geq d$)

SVD(X)

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!

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

- Compute $\mathbf{X}^T\mathbf{X}$ – $O(nd^2)$ runtime.
- Find eigendecomposition $\mathbf{X}^T\mathbf{X} = \mathbf{V\Lambda V}^T$ – $O(d^3)$ runtime.
- Compute $\mathbf{L} = \mathbf{XV}$ – $O(nd^2)$ runtime. Note that $\mathbf{L} = \mathbf{U\Sigma}$.
- 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.

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 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 $O(nd^2)$.

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 $\mathbf{X} \in \mathbb{R}^{n\times k}$ 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 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.

eig eigs

nd^2

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.

$X^T X$

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

(first column

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 $\vec{z} \approx \vec{v}_1$ – the top eigenvector of $\mathbf{A}$.

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

  Return $z^{(t)}$

normalize

$z^{(i)} \in \mathbb{R}^d$ is the solution at step i

$d = 2$

$z^{(1)} = \underbrace{Az^{(0)}}$

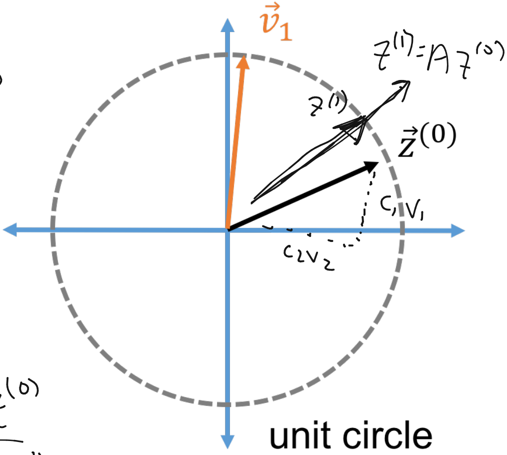
$z^{(1)} = A\underbrace{z^{(0)}}$

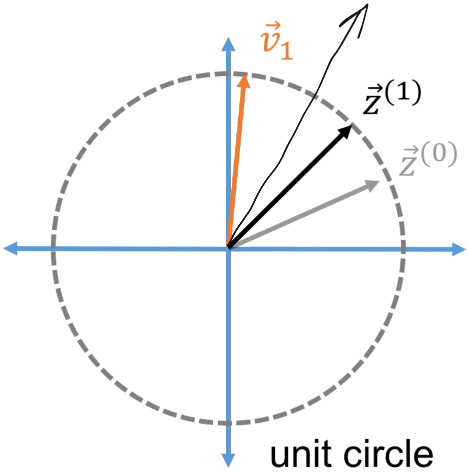
$z^{(2)} = \underline{A z^{(1)}}$

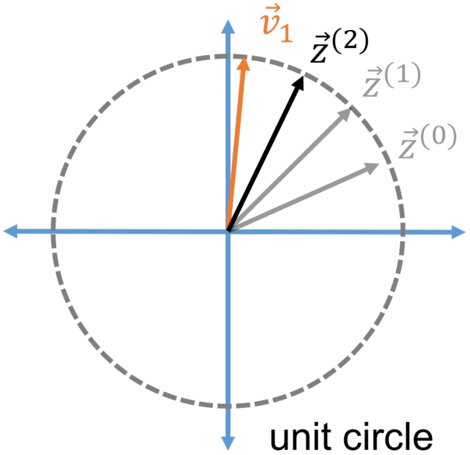
$\vdots$

$z^t = A z^{(t-1)}$

$$z^{(t)} = \frac{A^t \cdot z^{(0)}}{\|A^t \cdot z^{(0)}\|}$$




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


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

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 $\vec{v}_1$.

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

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

$V_1^T(c_1 V_1)$

$c_1 V_1^T V_1 = c_1 \|V_1\|_2^2$

c_1

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)

$d \times d \quad d \times 1$

$$\mathbf{V}^T\vec{z}^{(0)} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_d \end{bmatrix} \qquad \begin{bmatrix} - & V_1^T & - \\ & V_2^T & \\ & \vdots & \\ & V_d^T & \end{bmatrix} \left(c_1 \begin{bmatrix} V_1 \end{bmatrix} + c_2 \begin{bmatrix} V_2 \end{bmatrix} + \ldots \right)$$

$$\mathbf{\Lambda}\mathbf{V}^T\vec{z}^{(0)} = \begin{bmatrix} c_1\lambda_1 \\ c_2\lambda_2 \\ \vdots \\ c_d\lambda_d \end{bmatrix}$$

$A z^{(0)}$

$$\vec{z}^{(1)} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T \cdot \vec{z}^{(0)} = c_1\lambda_1\vec{V}_1 + c_2\lambda_2\vec{V}_2 + \ldots c_d\lambda_d\vec{V}_d$$

$\mathbf{A} \in \mathbb{R}^{d\times d}$: 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 $\vec{v}_1$.

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 $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$. $\vec{v}_1$: top eigenvector, being computed, $\vec{z}^{(i)}$: iterate at step i, converging to $\vec{v}_1$.

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

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

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

$\mathbf{A} \in \mathbb{R}^{d\times d}$: input matrix with eigendecomposition $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$. $\vec{v}_1$: 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:

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

$\approx$

v_1

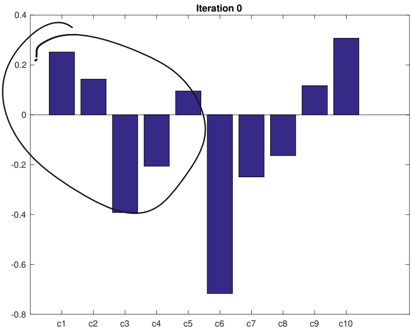
$\mathbf{A} \in \mathbb{R}^{d\times d}$: input matrix with eigendecomposition $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$. $\vec{v}_1$: top eigenvector, being computed, $\vec{z}^{(i)}$: iterate at step i, converging to $\vec{v}_1$.

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.

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

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.

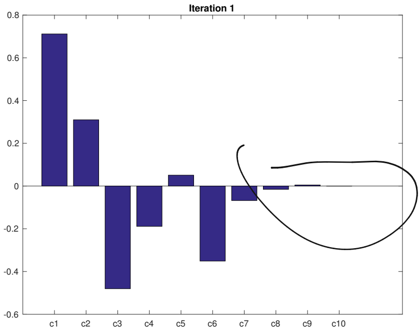
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



C_1 C_2 C_3 . . . .

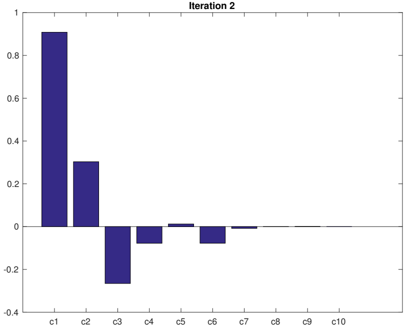
After t iterations, we have 'powered' up the eigenvalues, making the component in the direction of v_1 much larger, relative to the other components.

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



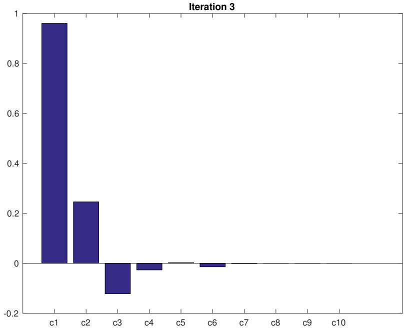
After t iterations, we have 'powered' up the eigenvalues, making the component in the direction of v_1 much larger, relative to the other components.

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



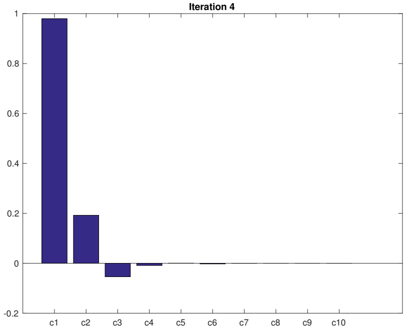
After t iterations, we have 'powered' up the eigenvalues, making the component in the direction of v_1 much larger, relative to the other components.

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



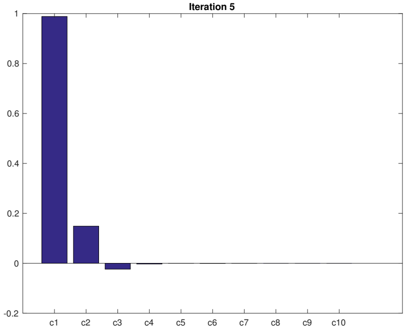
After t iterations, we have 'powered' up the eigenvalues, making the component in the direction of v_1 much larger, relative to the other components.

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



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.

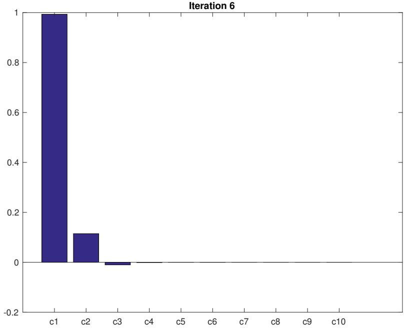
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



POWER METHOD CONVERGENCE

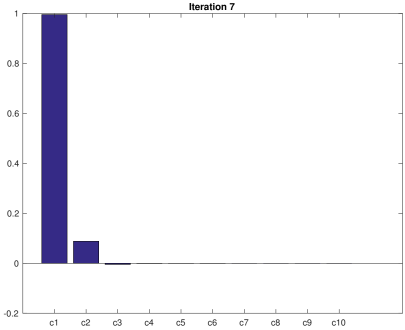
After t iterations, we have 'powered' up the eigenvalues, making the component in the direction of v_1 much larger, relative to the other components.

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



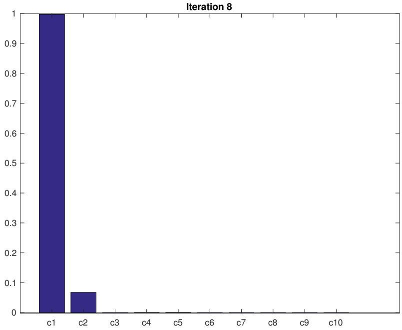
After t iterations, we have 'powered' up the eigenvalues, making the component in the direction of v_1 much larger, relative to the other components.

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



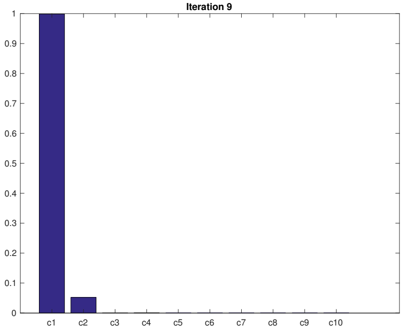
After t iterations, we have 'powered' up the eigenvalues, making the component in the direction of v_1 much larger, relative to the other components.

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



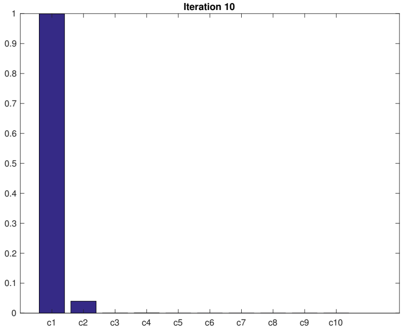
After t iterations, we have 'powered' up the eigenvalues, making the component in the direction of v_1 much larger, relative to the other components.

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



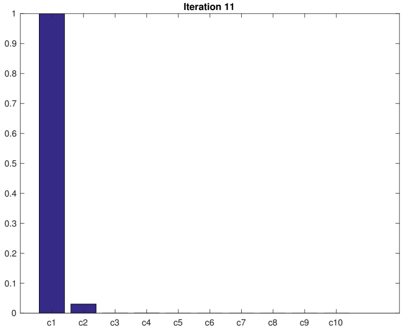
After t iterations, we have 'powered' up the eigenvalues, making the component in the direction of v_1 much larger, relative to the other components.

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



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.

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



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.

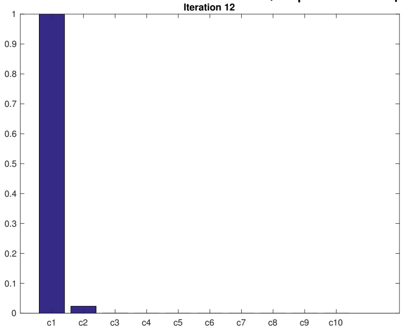
$\lambda_1 > \lambda_i$ for all

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

$\lambda_1 = 1.1 \quad \lambda_2 = 1$

$\lambda_1^t > \lambda_i^t$

λ_1^{20} vs λ_2^{20}

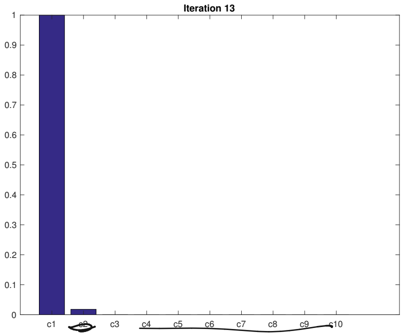


After t iterations, we have 'powered' up the eigenvalues, making the component in the direction of v_1 much larger, relative to the other components.

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

$\lambda_1 = 1.001$

$\lambda_2 = 1$



When will convergence be slow?

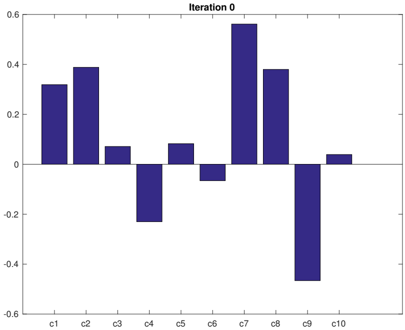
Slow Case: A has eigenvalues: $\lambda_1 = 1, \lambda_2 = .99, \lambda_3 = .9, \lambda_4 = .8, \ldots$

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

POWER METHOD SLOW CONVERGENCE

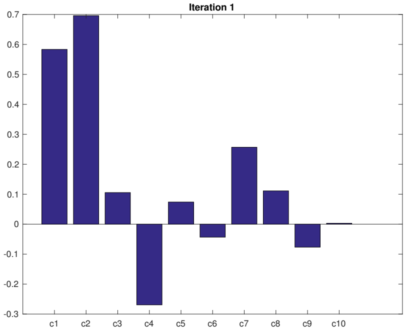
Slow Case: A has eigenvalues: $\lambda_1 = 1, \lambda_2 = .99, \lambda_3 = .9, \lambda_4 = .8, \ldots$

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



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

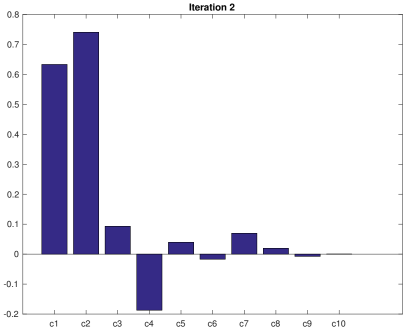
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



POWER METHOD SLOW CONVERGENCE

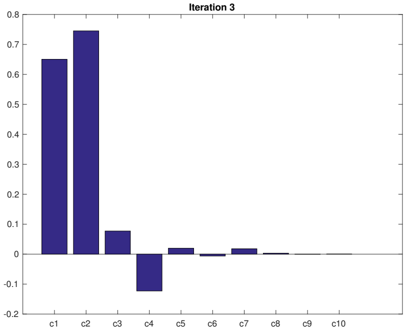
Slow Case: A has eigenvalues: $\lambda_1 = 1, \lambda_2 = .99, \lambda_3 = .9, \lambda_4 = .8, \ldots$

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



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

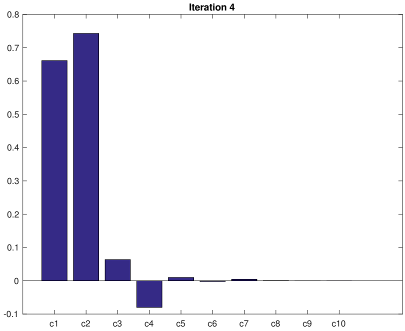
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



POWER METHOD SLOW CONVERGENCE

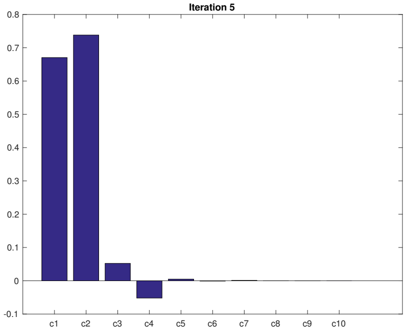
Slow Case: A has eigenvalues: $\lambda_1 = 1, \lambda_2 = .99, \lambda_3 = .9, \lambda_4 = .8, \ldots$

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



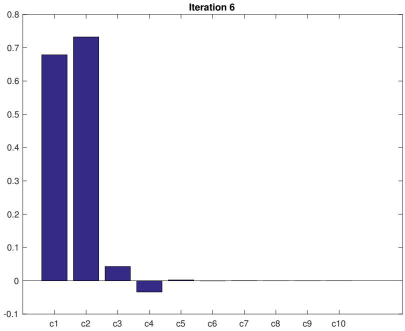
Slow Case: A has eigenvalues: $\lambda_1 = 1, \lambda_2 = .99, \lambda_3 = .9, \lambda_4 = .8, \ldots$

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



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

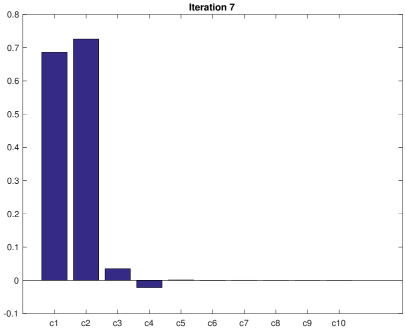
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



POWER METHOD SLOW CONVERGENCE

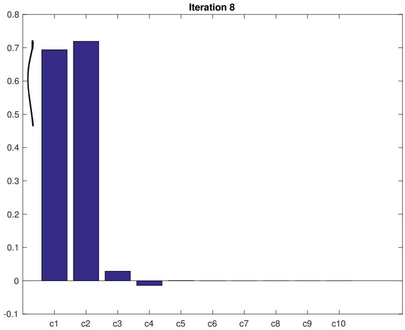
Slow Case: A has eigenvalues: $\lambda_1 = 1, \lambda_2 = .99, \lambda_3 = .9, \lambda_4 = .8, \ldots$

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



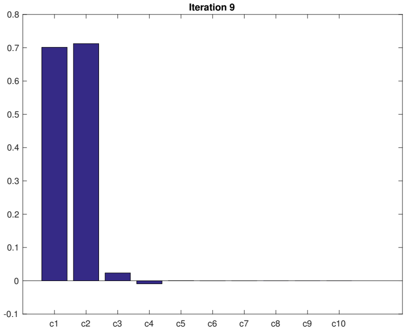
Slow Case: A has eigenvalues: $\lambda_1 = 1, \lambda_2 = .99, \lambda_3 = .9, \lambda_4 = .8, \ldots$

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



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

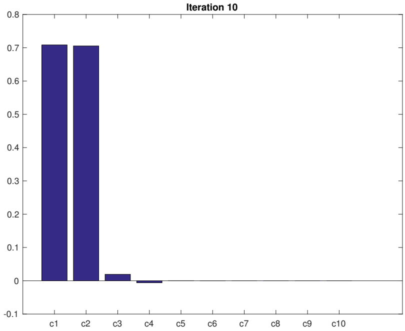
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



POWER METHOD SLOW CONVERGENCE

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

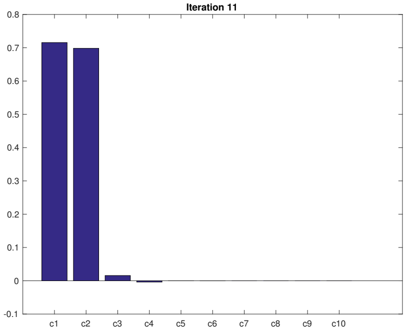
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



POWER METHOD SLOW CONVERGENCE

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

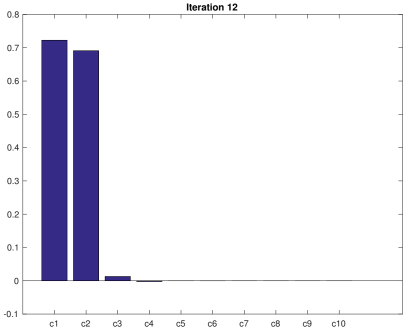
$$\vec{z}^{(0)} = c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d \implies \vec{z}^{(t)} = c_1\lambda_1^t\vec{v}_1 + c_2\lambda_2^t\vec{v}_2 + \ldots + c_d\lambda_d^t\vec{v}_d$$



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: $\lambda_1 = 1, \lambda_2 = .99, \lambda_3 = .9, \lambda_4 = .8, \ldots$

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

$z = .55 v_1 + .45 v_2$

$Az = .55 v_1 + .44 v_2$

$\approx z$