

The Lanczos Method in Data Science

Christopher Musco

Massachusetts Institute of Technology.

The Lanczos Method

The Lanczos Method

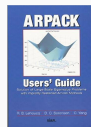
Used for solving linear systems, eigendecomposition, matrix exponentials, and approximating any matrix function.

The Lanczos Method

Used for solving linear systems, eigendecomposition, matrix exponentials, and approximating any matrix function.

- Introduced in 1950, developed through the 70s, ubiquitous in well-developed scientific computing libraries.

L A P A C K
L -A P -A C -K
L A P A -C -K
L -A P -A -C K
L A -P -A C K
L -A -P A C -K



The Lanczos Method

Used for solving linear systems, eigendecomposition, matrix exponentials, and approximating any matrix function.

- Introduced in 1950, developed through the 70s, ubiquitous in well-developed scientific computing libraries.
- Resurgence of interest due to new applications in data science and machine learning.



New applications combine Lanczos with super-scalable stochastic iterative and randomized sketching methods.

New applications combine Lanczos with super-scalable stochastic iterative and randomized sketching methods.

Require understanding of performance with noisy inputs.

New applications combine Lanczos with super-scalable stochastic iterative and randomized sketching methods.

Require understanding of performance with noisy inputs.

Today's results:

1. Lanczos is very noise stable, performing essentially optimally amongst other polynomial methods.

New applications combine Lanczos with super-scalable stochastic iterative and randomized sketching methods.

Require understanding of performance with noisy inputs.

Today's results:

1. Lanczos is very noise stable, performing essentially optimally amongst other polynomial methods.
2. Except when solving linear systems! We provide strong low-bounds that noise can significantly impair Lanczos and the closely related conjugate gradient method.

Stability of the Lanczos Method for Matrix Function
Approximation [SODA 2018]



Aaron Sidford
(Stanford)



Cameron Musco
(MIT)

Stability of the Lanczos Method for Matrix Function
Approximation [SODA 2018]



Aaron Sidford
(Stanford)



Cameron Musco
(MIT)



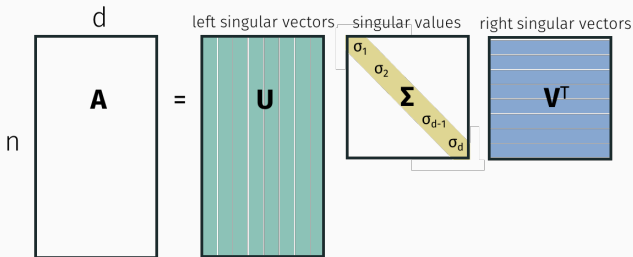
Roy Frostig
(Google)

Principal Component Projection Without Principal Component
Analysis [ICML 2016]

WHAT IS A **MATRIX FUNCTION**?

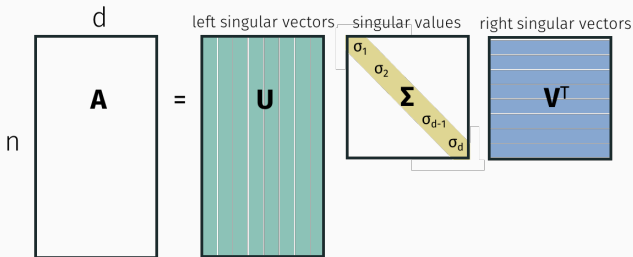
WHAT IS A MATRIX FUNCTION?

Every matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ has a singular value decomposition:



WHAT IS A MATRIX FUNCTION?

Every matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ has a singular value decomposition:



\mathbf{U}, \mathbf{V} are orthogonal, $\mathbf{\Sigma}$ is diagonal, $\sigma_1 \geq \dots \geq \sigma_d \in \mathbb{R}^+$.

WHAT IS A MATRIX FUNCTION?

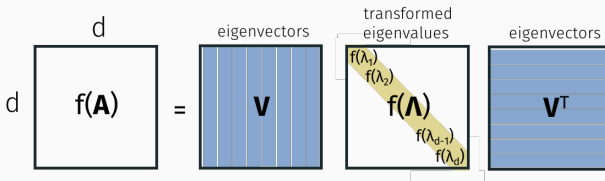
Every symmetric matrix $\mathbf{A} \in \mathbb{R}^{d \times d}$ has an orthogonal eigendecomposition:

The diagram illustrates the eigendecomposition of a symmetric matrix \mathbf{A} . It shows the equation $\mathbf{A} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^T$. The matrix \mathbf{A} is a white square with a black border, labeled with 'd' above and to its left. The matrix \mathbf{V} is a blue square with vertical yellow lines, labeled 'eigenvectors' above. The matrix $\mathbf{\Lambda}$ is a white square with a black border and a yellow diagonal, labeled 'eigenvalues' above. The diagonal elements are labeled $\lambda_1, \lambda_2, \dots, \lambda_{d-1}, \lambda_d$. The matrix \mathbf{V}^T is a blue square with horizontal yellow lines, labeled 'eigenvectors' above. The equals sign is placed between the matrices.

$$\begin{matrix} d \\ \square \\ \mathbf{A} \end{matrix} = \begin{matrix} \text{eigenvectors} \\ \square \\ \mathbf{V} \end{matrix} \begin{matrix} \text{eigenvalues} \\ \square \\ \mathbf{\Lambda} \end{matrix} \begin{matrix} \text{eigenvectors} \\ \square \\ \mathbf{V}^T \end{matrix}$$

WHAT IS A MATRIX FUNCTION?

For any scalar function $f: \mathbb{R} \rightarrow \mathbb{R}$ define $f(\mathbf{A})$:



Cost to compute $f(A)$:

Cost to compute $f(\mathbf{A})$:

$$\underbrace{O(n^3)}_{\text{eigendecompose } \mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T}$$

Cost to compute $f(\mathbf{A})$:

$$\underbrace{O(n^3)}_{\text{eigendecompose } \mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T} + \underbrace{O(n)}_{\text{compute } f(\mathbf{\Lambda})}$$

Cost to compute $f(\mathbf{A})$:

$$\underbrace{O(n^3)}_{\text{eigendecompose } \mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T} + \underbrace{O(n)}_{\text{compute } f(\mathbf{\Lambda})} + \underbrace{O(n^3)}_{\text{form } \mathbf{V}f(\mathbf{\Lambda})\mathbf{V}^T}$$

Cost to compute $f(\mathbf{A})$:

$$\underbrace{O(n^3)}_{\text{eigendecompose } \mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T} + \underbrace{O(n)}_{\text{compute } f(\mathbf{\Lambda})} + \underbrace{O(n^3)}_{\text{form } \mathbf{V}f(\mathbf{\Lambda})\mathbf{V}^T}$$

$= O(n^3)$ in practice

Cost to compute $f(\mathbf{A})$:

$$\underbrace{O(n^3)}_{\text{eigendecompose } \mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T} + \underbrace{O(n)}_{\text{compute } f(\mathbf{\Lambda})} + \underbrace{O(n^3)}_{\text{form } \mathbf{V}f(\mathbf{\Lambda})\mathbf{V}^T}$$

$$= O(n^3) \text{ in practice}$$

In theory can be improved to $O(n^\omega) \approx O(n^{2.3728639})$.
(but this is still slow)

Typically only interested in computing $f(\mathbf{A})\mathbf{x}$ for some $\mathbf{x} \in \mathbb{R}^n$.

$$f\left(\begin{bmatrix} & \mathbf{A} & \end{bmatrix}\right) \cdot \begin{bmatrix} \mathbf{x} \end{bmatrix}$$

Typically only interested in computing $f(\mathbf{A})\mathbf{x}$ for some $\mathbf{x} \in \mathbb{R}^n$.

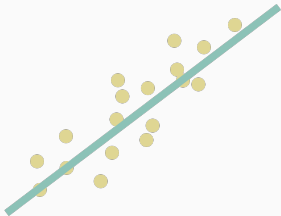
$$f\left(\begin{bmatrix} & \\ & \mathbf{A} \\ & \end{bmatrix}\right) \cdot \begin{bmatrix} \\ \\ \mathbf{x} \end{bmatrix}$$

Often much cheaper than computing $f(\mathbf{A})$ explicitly!

(this is what Lanczos and other algorithms target)

APPLICATIONS IN DATA PROBLEMS

Least squares regression



Least squares regression

$$\| \begin{matrix} \mathbf{A} \\ \mathbf{a}_i \end{matrix} \mathbf{w} - \begin{matrix} \mathbf{b} \\ \mathbf{b}_i \end{matrix} \|_2$$

Find \mathbf{w} that minimizes $\sum_{i=1}^n |b_i - \mathbf{a}_i^T \mathbf{w}|^2 = \|\mathbf{Aw} - \mathbf{b}\|_2^2$

Least squares regression

$$\|Aw - b\|_2$$

Find \mathbf{w} that minimizes $\sum_{i=1}^n |b_i - \mathbf{a}_i^T \mathbf{w}|^2 = \|\mathbf{Aw} - \mathbf{b}\|_2^2$

$$\text{Solution: } \mathbf{w} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b}$$

$$f\left(\begin{bmatrix} & \mathbf{A}^T \mathbf{A} & \end{bmatrix}\right) \cdot \begin{bmatrix} \mathbf{x} \end{bmatrix}$$

Where $f(\lambda) = 1/\lambda$ and $\mathbf{x} = \mathbf{A}^T \mathbf{b}$.

$$f\left(\begin{bmatrix} & \mathbf{A}^T \mathbf{A} & \end{bmatrix}\right) \cdot \begin{bmatrix} \mathbf{x} \end{bmatrix}$$

Where $f(\lambda) = 1/\lambda$ and $\mathbf{x} = \mathbf{A}^T \mathbf{b}$.

Since $\mathbf{V}^T \mathbf{V} = \mathbf{W} \mathbf{W}^T = \mathbf{I}$:

$$\overbrace{\left(\begin{bmatrix} \mathbf{V} \end{bmatrix} \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} \begin{bmatrix} \mathbf{V}^T \end{bmatrix}\right)}^{\mathbf{A}^T \mathbf{A}} \overbrace{\left(\begin{bmatrix} \mathbf{V} \end{bmatrix} \begin{bmatrix} \frac{1}{\lambda_1} & & \\ & \ddots & \\ & & \frac{1}{\lambda_n} \end{bmatrix} \begin{bmatrix} \mathbf{V}^T \end{bmatrix}\right)}^{(\mathbf{A}^T \mathbf{A})^{-1}} = \mathbf{I}$$

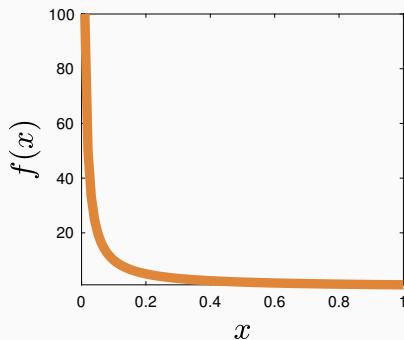
MATRIX INVERSE

Example

Linear system solving, $A^{-1}\mathbf{x}$

Function

$$f(x) = 1/x$$



Countless applications...

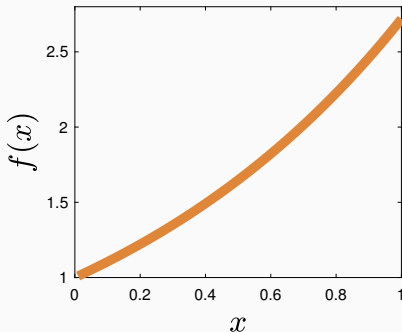
MATRIX EXPONENTIAL

Example

Matrix exponential, e^{Ax}

Function

$$f(x) = e^x$$



Applications in semidefinite programming, graph algorithms
(balanced separator), differential equations.

[Arora, Hazan, Kale, '05], [Iyengar, Phillips, and Stein '11],
[Orecchia, Sachdeva, Vishnoi, '12], [Higham '08] (very complete survey)

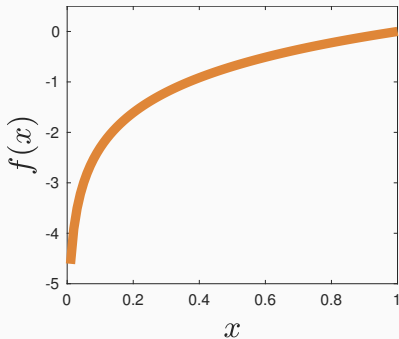
MATRIX LOG

Example

Matrix log, $\log(\mathbf{A})\mathbf{x}$

Function

$$f(x) = \log(x)$$



Used to estimate $\log(\det(\mathbf{A})) = \text{tr}(\log(\mathbf{A}))$.

Appears in log-likelihood equation for multivariate Gaussian.
Applications in Gaussian process regression, learning distance kernels, Markov random fields.

[Dhillon, et al '06, '07,'08], [Han, Malioutov, Shin '15], [Saibaba, Alexanderian, Ipsen '17]

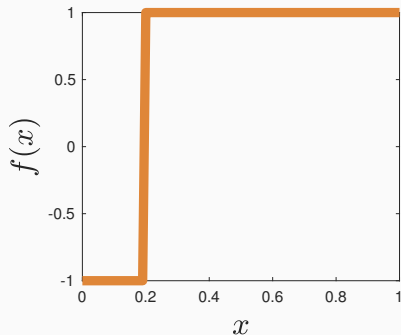
MATRIX STEP FUNCTION

Example

Step function, $\text{step}_\lambda(\mathbf{A})\mathbf{x}$

Function

$$f(x) = \begin{cases} 1, & x \geq \lambda \\ 0, & x < \lambda \end{cases}$$



Projection to top eigenvectors, eigenvalue counting, computing matrix norms, spectral filtering, many more...

[Frostig, Musco, Musco, Sidford '16], [Saad, Ubaru '16], [Allen-Zhu, Li '17], [Tremblay, Puy, Gribonval, Vandergheynst '16], [Musco, Netrapalli, Sidford, Ubaru and Woodruff '18]

Standard Regression:

Given: \mathbf{A} , \mathbf{b}

Solve: $\mathbf{x}^* = \arg \min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|^2$

Standard Regression:

Given: \mathbf{A} , \mathbf{b}

Solve: $\mathbf{x}^* = \arg \min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|^2$

Principal Component Regression:

Given: \mathbf{A} , \mathbf{b} , λ

Solve: $\mathbf{x}^* = \arg \min_{\mathbf{x}} \|\mathbf{A}_{\lambda}\mathbf{x} - \mathbf{b}\|^2$

PRINCIPAL COMPONENT REGRESSION

Standard Regression:

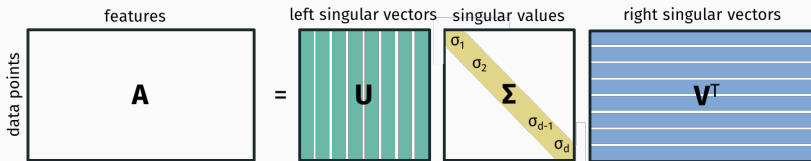
Given: \mathbf{A} , \mathbf{b}

Solve: $\mathbf{x}^* = \arg \min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|^2$

Principal Component Regression:

Given: \mathbf{A} , \mathbf{b} , λ

Solve: $\mathbf{x}^* = \arg \min_{\mathbf{x}} \|\mathbf{A}_{\lambda} \mathbf{x} - \mathbf{b}\|^2$



PRINCIPAL COMPONENT REGRESSION

Standard Regression:

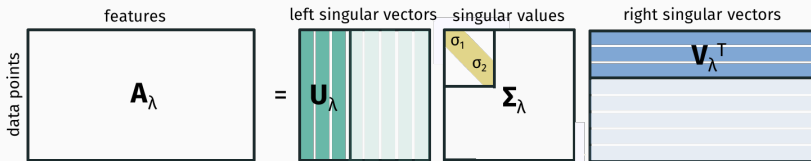
Given: \mathbf{A} , \mathbf{b}

Solve: $\mathbf{x}^* = \arg \min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|^2$

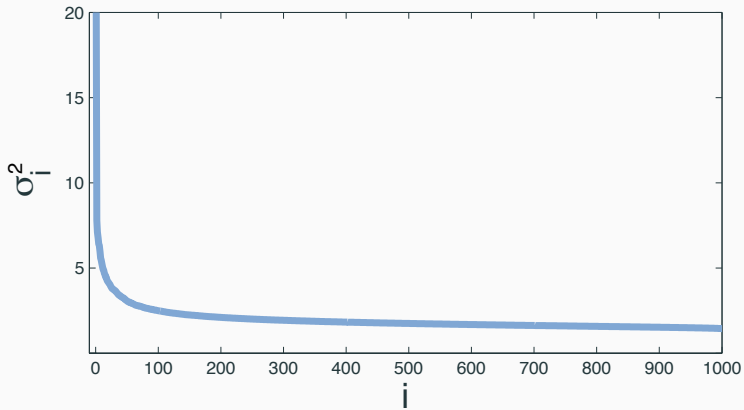
Principal Component Regression:

Given: \mathbf{A} , \mathbf{b} , λ

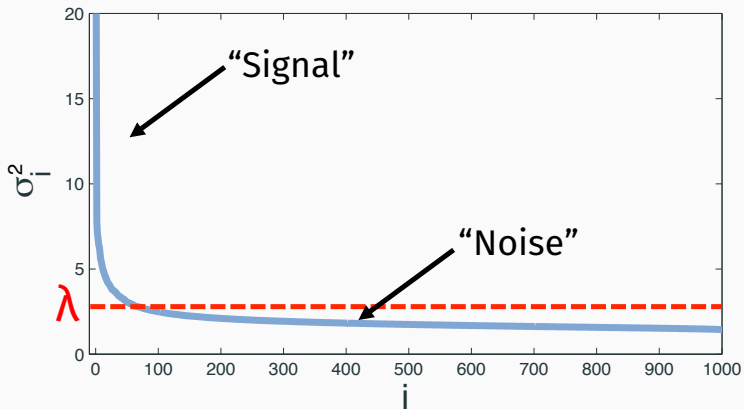
Solve: $\mathbf{x}^* = \arg \min_{\mathbf{x}} \|\mathbf{A}_{\lambda} \mathbf{x} - \mathbf{b}\|^2$



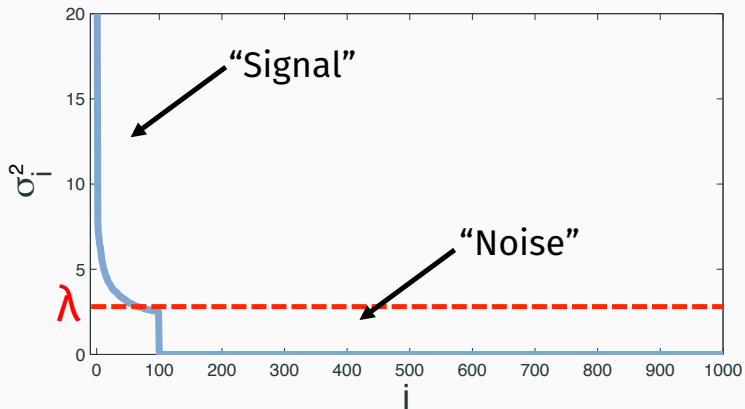
Singular values of A



Singular values of A



Singular values of A_λ



Principal Component Regression (PCR):

$$\text{Goal: } \mathbf{x}^* = \arg \min_{\mathbf{x}} \|\mathbf{A}_{\lambda} \mathbf{x} - \mathbf{b}\|^2$$

$$\text{Solution: } \mathbf{x} = (\mathbf{A}_{\lambda}^T \mathbf{A}_{\lambda})^{-1} \mathbf{A}_{\lambda}^T \mathbf{b}$$

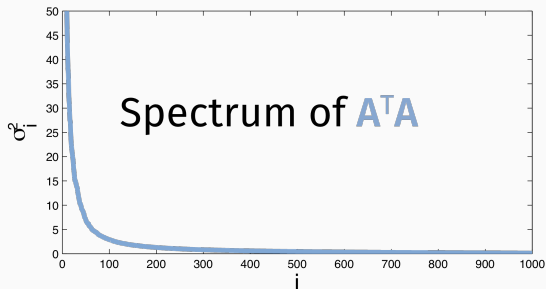
Principal Component Regression (PCR):

$$\text{Goal: } \mathbf{x}^* = \arg \min_{\mathbf{x}} \|\mathbf{A}_{\lambda} \mathbf{x} - \mathbf{b}\|^2$$

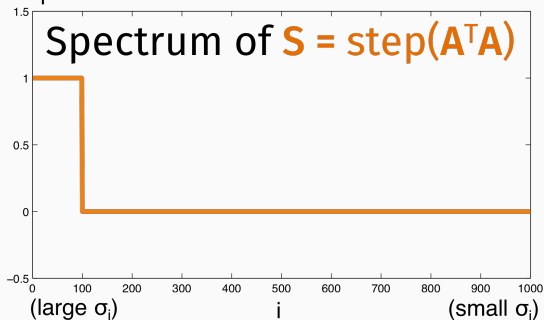
$$\text{Solution: } \mathbf{x} = (\mathbf{A}_{\lambda}^T \mathbf{A}_{\lambda})^{-1} \mathbf{A}_{\lambda}^T \mathbf{b}$$

Fastest way to apply $\mathbf{A}_{\lambda}^T \mathbf{A}_{\lambda}$ and $(\mathbf{A}_{\lambda}^T \mathbf{A}_{\lambda})^{-1}$ to a vector is with a matrix step function.

PRINCIPAL COMPONENT REGRESSION



$$A_{\lambda}^T A_{\lambda} = S A^T A$$



How many eigenvalues does **A** have that are greater than λ ?

How many eigenvalues does \mathbf{A} have that are greater than λ ?

$$\sum_{i=1}^d \mathbb{I}[\lambda_i > \lambda] = \sum_{i=1}^d \text{step}_{\lambda}(\lambda_i(\mathbf{A}))$$

How many eigenvalues does \mathbf{A} have that are greater than λ ?

$$\sum_{i=1}^d \mathbb{I}[\lambda_i > \lambda] = \sum_{i=1}^d \text{step}_{\lambda}(\lambda_i(\mathbf{A})) = \text{trace}(\text{step}_{\lambda}(\mathbf{A}))$$

EIGENVALUE COUNTING

How many eigenvalues does \mathbf{A} have that are greater than λ ?

$$\sum_{i=1}^d \mathbb{I}[\lambda_i > \lambda] = \sum_{i=1}^d \text{step}_{\lambda}(\lambda_i(\mathbf{A})) = \text{trace}(\text{step}_{\lambda}(\mathbf{A}))$$

Hutchinson's estimator:

$$\mathbb{E}_{\mathbf{x} \sim \mathcal{N}_d} [\mathbf{x}^T f(\mathbf{A}) \mathbf{x}] = \text{trace}(f(\mathbf{A}))$$

The diagram illustrates the Hutchinson's estimator formula. It shows a vector \mathbf{x}^T (represented as a row vector with values $-.11, -.14, 1.4, -.91, .2$) multiplied by a function f applied to a matrix \mathbf{A} (represented as a yellow square). The result is a vector \mathbf{x} (represented as a column vector with values $-.11, -.14, 1.4, -.91, .2$).

How many eigenvalues does \mathbf{A} have that are greater than λ ?

$$\sum_{i=1}^d \mathbb{I}[\lambda_i > \lambda] = \sum_{i=1}^d \text{step}_{\lambda}(\lambda_i(\mathbf{A})) = \text{trace}(\text{step}_{\lambda}(\mathbf{A}))$$

Hutchinson's estimator:

$$\mathbb{E}_{\mathbf{x} \sim \mathcal{N}_d} [\mathbf{x}^T f(\mathbf{A}) \mathbf{x}] = \text{trace}(f(\mathbf{A}))$$

The diagram shows the computation of the Hutchinson's estimator. On the left, a row vector \mathbf{x}^T is shown with values $[-.11, -.14, 1.4, -.91, .2]$. In the center, a function f is applied to a square matrix \mathbf{A} , which is represented by a yellow box. On the right, the resulting vector \mathbf{x} is shown with values $[-.11, -.14, 1.4, -.91, .2]$.

Same method used for estimating log-determinants and matrix norms.

FAST ALGORITHMS FOR
MATRIX FUNCTIONS

$$f\left(\begin{bmatrix} & A & \end{bmatrix}\right) \cdot \begin{bmatrix} x \end{bmatrix}$$

Matrix **polynomials** can be computed iteratively.

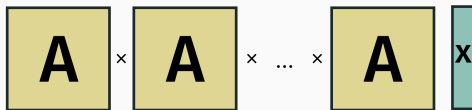
$$p\left(\begin{bmatrix} & \\ & A \end{bmatrix}\right) \cdot \begin{bmatrix} x \end{bmatrix}$$

MATRIX POLYNOMIALS

Matrix **polynomials** can be computed iteratively.

$$p\left(\begin{bmatrix} A \end{bmatrix}\right) \cdot \begin{bmatrix} x \end{bmatrix}$$

$$A^k x = V \Lambda V^T V \Lambda V^T \dots V \Lambda V^T x = V \Lambda^k V^T x$$

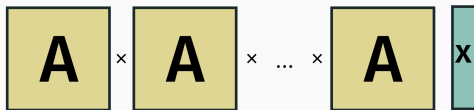

$$\boxed{A} \times \boxed{A} \times \dots \times \boxed{A} \boxed{x}$$

MATRIX POLYNOMIALS

Matrix **polynomials** can be computed iteratively.

$$p\left(\begin{bmatrix} A \end{bmatrix}\right) \cdot \begin{bmatrix} x \end{bmatrix}$$

$$A^k x = V \Lambda V^T V \Lambda V^T \dots V \Lambda V^T x = V \Lambda^k V^T x$$


$$\boxed{A} \times \boxed{A} \times \dots \times \boxed{A} \boxed{x}$$

Total time to compute $p(A)x = c_0x + c_1Ax + c_2A^2x + \dots + c_kA^kx$:

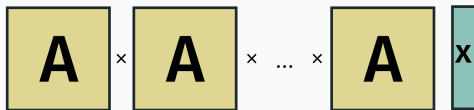
$$O(k \cdot \text{nnz}(A))$$

MATRIX POLYNOMIALS

Matrix **polynomials** can be computed iteratively.

$$p\left(\begin{bmatrix} A \end{bmatrix}\right) \cdot \begin{bmatrix} x \end{bmatrix}$$

$$A^k x = V \Lambda V^T V \Lambda V^T \dots V \Lambda V^T x = V \Lambda^k V^T x$$



The diagram illustrates the iterative computation of $A^k x$ as a sequence of matrix multiplications. It consists of three yellow square boxes, each containing the letter **A**, followed by a vertical teal rectangle containing the letter **x**. These boxes are connected by multiplication symbols (\times) and an ellipsis (\dots), representing the sequence $A \times A \times \dots \times A \times x$.

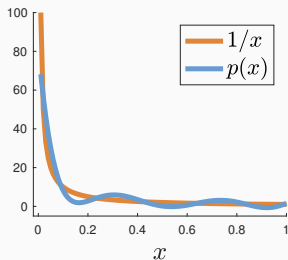
Total time to compute $p(A)x = c_0x + c_1Ax + c_2A^2x + \dots + c_kA^kx$:

$$O(k \cdot \text{nnz}(A)) \leq O(k \cdot n^2) \ll O(n^3)$$

For general matrix functions:
approximate $f(x)$ with low-degree polynomial $p(x)$.

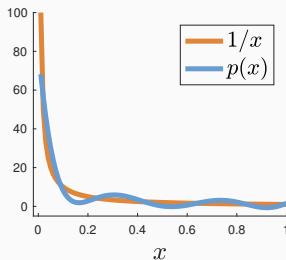
POLYNOMIAL APPROXIMATION

For general matrix functions:
approximate $f(x)$ with low-degree polynomial $p(x)$.



$$f(A)x \approx p(A)x$$

For general matrix functions:
approximate $f(x)$ with low-degree polynomial $p(x)$.



$$f(\mathbf{A})\mathbf{x} \approx p(\mathbf{A})\mathbf{x}$$

How does error in approximating scalar function $f(\cdot)$
translate to error on matrix function?

$$\|f(\mathbf{A})\mathbf{x} - p(\mathbf{A})\mathbf{x}\| \leq \|f(\mathbf{A}) - p(\mathbf{A})\| \cdot \|\mathbf{x}\|$$

$$\|f(A)x - p(A)x\| \leq \|f(A) - p(A)\| \cdot \|x\| \leq \epsilon \cdot \|x\|$$

where

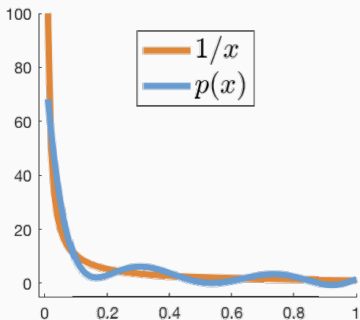
$$\epsilon = \max_{i=1,\dots,n} |f(\lambda_i) - p(\lambda_i)|.$$

POLYNOMIAL APPROXIMATION

$$\|f(A)x - p(A)x\| \leq \|f(A) - p(A)\| \cdot \|x\| \leq \epsilon \cdot \|x\|$$

where

$$\epsilon = \max_{i=1,\dots,n} |f(\lambda_i) - p(\lambda_i)|.$$

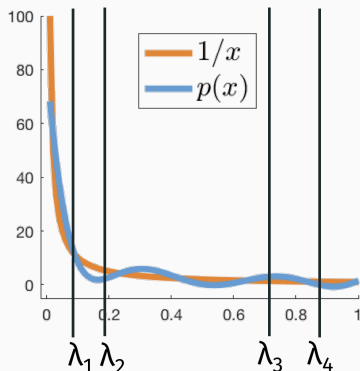


POLYNOMIAL APPROXIMATION

$$\|f(A)x - p(A)x\| \leq \|f(A) - p(A)\| \cdot \|x\| \leq \epsilon \cdot \|x\|$$

where

$$\epsilon = \max_{i=1,\dots,n} |f(\lambda_i) - p(\lambda_i)|.$$

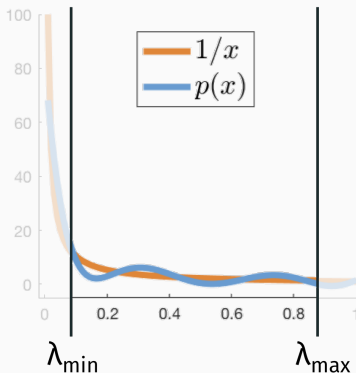


POLYNOMIAL APPROXIMATION

$$\|f(A)x - p(A)x\| \leq \|f(A) - p(A)\| \cdot \|x\| \leq \epsilon \cdot \|x\|$$

where

$$\epsilon = \max_{i=1,\dots,n} |f(\lambda_i) - p(\lambda_i)|.$$



If we know $\lambda_{\min}(\mathbf{A})$ and $\lambda_{\max}(\mathbf{A})$ we can explicitly compute a near optimal polynomial p via Chebyshev interpolation.

If we know $\lambda_{\min}(\mathbf{A})$ and $\lambda_{\max}(\mathbf{A})$ we can explicitly compute a near optimal polynomial p via Chebyshev interpolation.

$$\delta_k = \min_{p \text{ a degree } k \text{ polynomial}} \left(\max_{x \in [\lambda_{\min}(\mathbf{A}), \lambda_{\max}(\mathbf{A})]} |f(x) - p(x)| \right)$$

Final bound: Output \mathbf{y} such that

$$\|f(\mathbf{A})\mathbf{x} - \mathbf{y}\| \leq O(\log k) \cdot \delta_k \cdot \|\mathbf{x}\|.$$

FINDING GOOD APPROXIMATING POLYNOMIALS

If we know $\lambda_{\min}(\mathbf{A})$ and $\lambda_{\max}(\mathbf{A})$ we can explicitly compute a near optimal polynomial p via Chebyshev interpolation.

$$\delta_k = \min_{p \text{ a degree } k \text{ polynomial}} \left(\max_{x \in [\lambda_{\min}(\mathbf{A}), \lambda_{\max}(\mathbf{A})]} |f(x) - p(x)| \right)$$

Final bound: Output \mathbf{y} such that

$$\|f(\mathbf{A})\mathbf{x} - \mathbf{y}\| \leq O(\log k) \cdot \delta_k \cdot \|\mathbf{x}\|.$$

Example bounds:

- Linear systems in $O\left(\sqrt{\lambda_{\max} / \lambda_{\min}}\right)$ iterations.
- Matrix exponential in $O(\|\mathbf{A}\|)$ iterations.
- Matrix sign function in $O(1/\epsilon)$ iterations.
- Top eigenvector in $O(\log(n)/\sqrt{\epsilon})$ iterations.

Example bounds:

- Linear systems in $O\left(\sqrt{\lambda_{\max} / \lambda_{\min}}\right)$ iterations.
- Matrix exponential in $O(\|\mathbf{A}\|)$ iterations.
- Matrix sign function in $O(1/\epsilon)$ iterations.
- Top eigenvector in $O(\log(n)/\sqrt{\epsilon})$ iterations.

No one actually uses Chebyshev interpolation!

THE LANCZOS METHOD
FOR MATRIX FUNCTIONS

THE LANCZOS METHOD



Cornelius Lanczos, 1950



Cornelius Lanczos, 1950

- Simple to implement.
- No need to know $\lambda_{\min}(\mathbf{A})$ and $\lambda_{\max}(\mathbf{A})$.
- Much better convergence in practice (for many reasons).



Cornelius Lanczos, 1950

- Simple to implement.
- No need to know $\lambda_{\min}(\mathbf{A})$ and $\lambda_{\max}(\mathbf{A})$.
- Much better convergence in practice (for many reasons).
- Matches optimal uniform approximation up to factor 2.



Cornelius Lanczos, 1950

- Simple to implement.
- No need to know $\lambda_{\min}(\mathbf{A})$ and $\lambda_{\max}(\mathbf{A})$.
- Much better convergence in practice (for many reasons).
- Matches optimal uniform approximation up to factor 2.

Final bound: Output \mathbf{y} such that $\|f(\mathbf{A})\mathbf{x} - \mathbf{y}\| \leq 2\delta_k \cdot \|\mathbf{x}\|$.

Step 1: Form orthogonal matrix $\mathbf{Q} = [\mathbf{q}_0, \mathbf{q}_1, \dots, \mathbf{q}_k]$ that spans the Krylov subspace

$$\mathcal{K} = \{\mathbf{x}, \mathbf{A}\mathbf{x}, \mathbf{A}^2\mathbf{x}, \dots, \mathbf{A}^k\mathbf{x}\}.$$

LANCZOS METHOD FOR MATRIX FUNCTIONS

Step 1: Form orthogonal matrix $\mathbf{Q} = [\mathbf{q}_0, \mathbf{q}_1, \dots, \mathbf{q}_k]$ that spans the Krylov subspace

$$\mathcal{K} = \{\mathbf{x}, \mathbf{A}\mathbf{x}, \mathbf{A}^2\mathbf{x}, \dots, \mathbf{A}^k\mathbf{x}\}.$$

Step 2: Compute

$$\mathbf{T} = \mathbf{Q}^T \mathbf{A} \mathbf{Q}$$

LANCZOS METHOD FOR MATRIX FUNCTIONS

Step 1: Form orthogonal matrix $\mathbf{Q} = [\mathbf{q}_0, \mathbf{q}_1, \dots, \mathbf{q}_k]$ that spans the Krylov subspace

$$\mathcal{K} = \{\mathbf{x}, \mathbf{A}\mathbf{x}, \mathbf{A}^2\mathbf{x}, \dots, \mathbf{A}^k\mathbf{x}\}.$$

Step 2: Compute

$$\mathbf{T} = \mathbf{Q}^T \mathbf{A} \mathbf{Q}$$

Step 3: Approximate $f(\mathbf{A})\mathbf{x}$ by

$$\mathbf{Q}f(\mathbf{T})\mathbf{Q}^T\mathbf{x}$$

LANCZOS METHOD FOR MATRIX FUNCTIONS

$$\overset{k}{\subseteq} Q = \text{span} \left(\begin{array}{c} x \quad Ax \quad \dots \quad A^{k-1}x \end{array} \right)$$

LANCZOS METHOD FOR MATRIX FUNCTIONS

$$\overset{k}{\subseteq} \mathbf{Q} = \text{span} \left(\begin{array}{c} \text{---} \\ \mathbf{x} \quad \mathbf{Ax} \quad \dots \quad \mathbf{A}^k \mathbf{x} \\ \text{---} \end{array} \right)$$

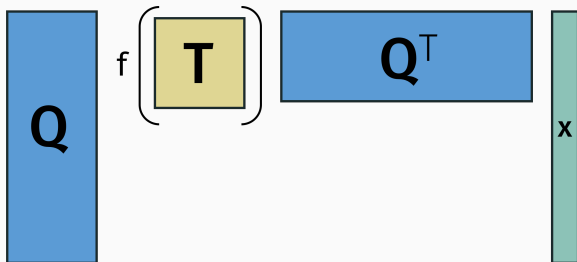
Runtime: $O(k \cdot \text{nnz}(\mathbf{A})) + O(nk^2)$

LANCZOS METHOD FOR MATRIX FUNCTIONS

$$\begin{matrix} & k \\ \begin{matrix} \downarrow \\ k \end{matrix} & \mathbf{T} \end{matrix} = \begin{matrix} & n \\ \mathbf{Q}^T & \end{matrix} \begin{matrix} n \\ \mathbf{A} \\ n \end{matrix} \begin{matrix} \mathbf{Q} \\ k \end{matrix}$$

Runtime: $O(k \cdot \text{nnz}(\mathbf{A})) + O(nk^2)$

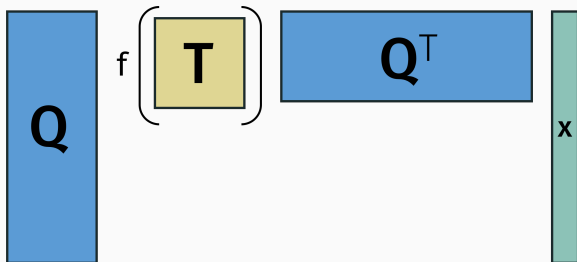
LANCZOS METHOD FOR MATRIX FUNCTIONS



Runtime: $O(k \cdot \text{nnz}(A)) + O(nk^2) + O(k^3)$

Reduce the problem to the cost of computing a matrix function for a $k \times k$ matrix.

LANCZOS METHOD FOR MATRIX FUNCTIONS

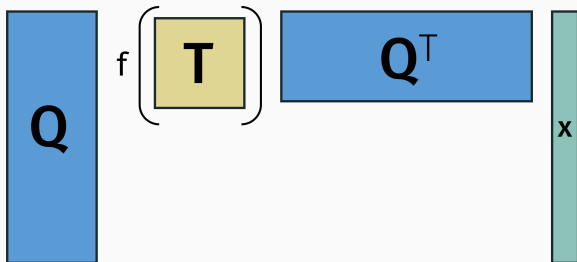


Runtime: $O(k \cdot \text{nnz}(A)) + O(nk^2) + O(k^3)$

Runtime: $O(k \cdot \text{nnz}(A) + nk)$

Reduce the problem to the cost of computing a matrix function for a $k \times k$ matrix.

LANCZOS METHOD FOR MATRIX FUNCTIONS



Runtime: $O(k \cdot \text{nnz}(A)) + O(nk^2) + O(k^3)$

Runtime: $O(k \cdot \text{nnz}(A) + nk)$

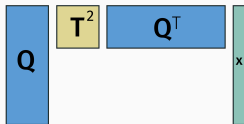
Reduce the problem to the cost of computing a matrix function for a $k \times k$ matrix.

Final bound: Output y such that $\|f(A)x - y\| \leq 2\delta_k \cdot \|x\|$.

Claim: Lanczos applies degree k polynomials exactly.

Claim: Lanczos applies degree k polynomials exactly.

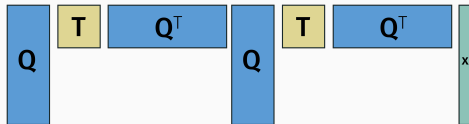
Proof:



The diagram illustrates the Lanczos polynomial equation. It consists of four rectangular blocks arranged horizontally. The first block is a tall blue rectangle labeled Q . The second block is a small yellow square labeled T^2 . The third block is a wide blue rectangle labeled Q^T . The fourth block is a tall, narrow light green rectangle labeled x . The blocks are connected by plus signs, representing the equation $Q(T^2 - Q^T)x = 0$.

Claim: Lanczos applies degree k polynomials exactly.

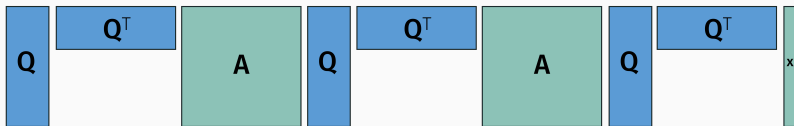
Proof:



QUICK ANALYSIS OF LANCZOS

Claim: Lanczos applies degree k polynomials exactly.

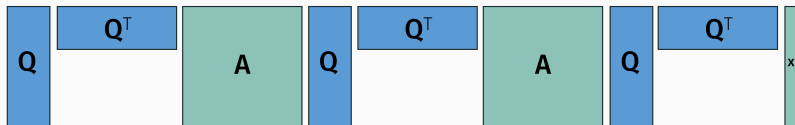
Proof:



QUICK ANALYSIS OF LANCZOS

Claim: Lanczos applies degree k polynomials exactly.

Proof:

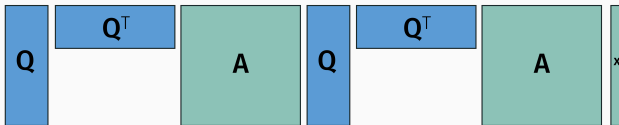


x, Ax, A^2x all lie in the span of Q .

QUICK ANALYSIS OF LANCZOS

Claim: Lanczos applies degree k polynomials exactly.

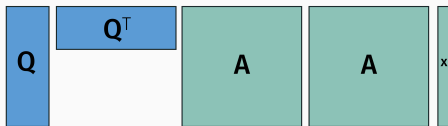
Proof:



x, Ax, A^2x all lie in the span of Q .

Claim: Lanczos applies degree k polynomials exactly.

Proof:



x, Ax, A^2x all lie in the span of Q .

Claim: Lanczos applies degree k polynomials exactly.

Proof:



x, Ax, A^2x all lie in the span of Q .

How about for a general functions $f(x)$?

How about for a general functions $f(x)$?

Lanczos automatically applies the polynomial “part” of f .
(simple application of triangle inequality)

How about for a general functions $f(x)$?

Lanczos automatically applies the polynomial “part” of f .
(simple application of triangle inequality)

For any degree k polynomial p ,

$$\begin{aligned}\|f(A)x - Qf(T)Q^T x\| &\leq \|f(A)x - p(A)x\| \\ &\quad + \|p(A)x - Qp(T)Q^T x\| \\ &\quad + \|Qp(T)Q^T x - Qf(T)Q^T x\|\end{aligned}$$

How about for a general functions $f(x)$?

Lanczos automatically applies the polynomial “part” of f .
(simple application of triangle inequality)

For any degree k polynomial p ,

$$\begin{aligned}\|f(A)x - Qf(T)Q^T x\| &\leq \|f(A)x - p(A)x\| \\ &\quad + \|p(A)x - Qp(T)Q^T x\| \\ &\quad + \|Qp(T)Q^T x - Qf(T)Q^T x\| \\ &\leq \delta_k \|x\|\end{aligned}$$

How about for a general functions $f(x)$?

Lanczos automatically applies the polynomial “part” of f .
(simple application of triangle inequality)

For any degree k polynomial p ,

$$\begin{aligned}\|f(A)x - Qf(T)Q^T x\| &\leq \|f(A)x - p(A)x\| \\ &\quad + \|p(A)x - Qp(T)Q^T x\| \\ &\quad + \|Qp(T)Q^T x - Qf(T)Q^T x\| \\ &\leq \delta_k \|x\| + 0\end{aligned}$$

How about for a general functions $f(x)$?

Lanczos automatically applies the polynomial “part” of f .
(simple application of triangle inequality)

For any degree k polynomial p ,

$$\begin{aligned}\|f(A)x - Qf(T)Q^T x\| &\leq \|f(A)x - p(A)x\| \\ &\quad + \|p(A)x - Qp(T)Q^T x\| \\ &\quad + \|Qp(T)Q^T x - Qf(T)Q^T x\| \\ &\leq \delta_k \|x\| + 0 + \|p(T) - f(T)\| \cdot \|Q^T x\|\end{aligned}$$

Since $T = Q^T A Q$, $[\lambda_{\min}(T), \lambda_{\max}(T)] \subseteq [\lambda_{\min}(A), \lambda_{\max}(A)]$.

How about for a general functions $f(x)$?

Lanczos automatically applies the polynomial “part” of f .
(simple application of triangle inequality)

For any degree k polynomial p ,

$$\begin{aligned}\|f(A)x - Qf(T)Q^T x\| &\leq \|f(A)x - p(A)x\| \\ &\quad + \|p(A)x - Qp(T)Q^T x\| \\ &\quad + \|Qp(T)Q^T x - Qf(T)Q^T x\| \\ &\leq \delta_k \|x\| + 0 + \delta_k \|x\|.\end{aligned}$$

Since $T = Q^T A Q$, $[\lambda_{\min}(T), \lambda_{\max}(T)] \subseteq [\lambda_{\min}(A), \lambda_{\max}(A)]$.

POLYNOMIAL METHODS
WITH NOISE

In many data applications, we do not multiply by A exactly!

In many data applications, we do not multiply by A exactly!



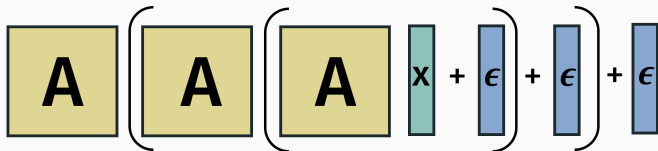
A diagram illustrating the equation $Ax + \epsilon$. It consists of three main components arranged horizontally: a yellow square containing the letter 'A', a teal vertical rectangle containing the letter 'x', and a blue vertical rectangle containing the Greek letter 'epsilon' (ϵ). A plus sign '+' is positioned between the teal rectangle and the blue rectangle. All elements have a thin black border.

In many data applications, we do not multiply by A exactly!

$$A \left(A x + \epsilon \right) + \epsilon$$

The diagram illustrates the equation $A(Ax + \epsilon) + \epsilon$ using colored boxes to represent different components. The first A is in a yellow square box. The second A is in a yellow square box. The variable x is in a teal vertical rectangle. The noise term ϵ is in a blue vertical rectangle. The entire expression is enclosed in large parentheses, with a plus sign and another ϵ term (in a blue vertical rectangle) added outside the parentheses.

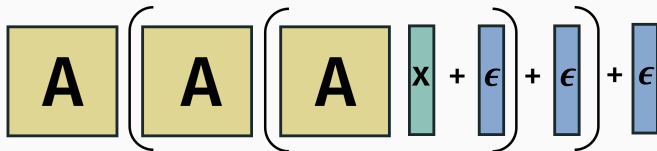
In many data applications, we do not multiply by A exactly!



The diagram illustrates a sequence of matrix multiplications with added noise. It consists of three yellow square blocks labeled 'A' and three blue vertical rectangular blocks labeled 'ε'. A green vertical rectangular block labeled 'x' is also present. The expression is structured as follows: the first 'A' block is followed by a large right parenthesis '}', then the second 'A' block is followed by a large right parenthesis '}', then the third 'A' block is followed by a large right parenthesis '}'. Inside the first parenthesis is the 'x' block followed by a '+' sign and the first 'ε' block. Inside the second parenthesis is the second 'ε' block followed by a '+' sign. Inside the third parenthesis is the third 'ε' block followed by a '+' sign. The final 'ε' block is outside the third parenthesis. The entire expression is enclosed in a large left parenthesis '(' at the beginning.

$$A \left(A \left(A x + \epsilon \right) + \epsilon \right) + \epsilon$$

In many data applications, we do not multiply by A exactly!



The diagram illustrates a matrix function with noise model. It consists of three yellow square blocks labeled 'A' representing matrix multiplications. The first 'A' is followed by a large right parenthesis '('. Inside this parenthesis is a second yellow square block 'A' followed by another large right parenthesis '('. Inside this second parenthesis is a third yellow square block 'A' followed by a teal vertical rectangle labeled 'x'. To the right of 'x' is a plus sign '+', followed by a blue vertical rectangle labeled with the Greek letter epsilon 'ε'. This inner parenthesis is closed by a large right parenthesis ')'. To the right of this is a plus sign '+', followed by a blue vertical rectangle labeled 'ε'. This middle parenthesis is closed by a large right parenthesis ')'. Finally, to the right of this is a plus sign '+', followed by a blue vertical rectangle labeled 'ε'. The entire expression is enclosed in a large right parenthesis '(', which is closed by a large right parenthesis ')' at the end.

$$A \left(A \left(A x + \epsilon \right) + \epsilon \right) + \epsilon$$

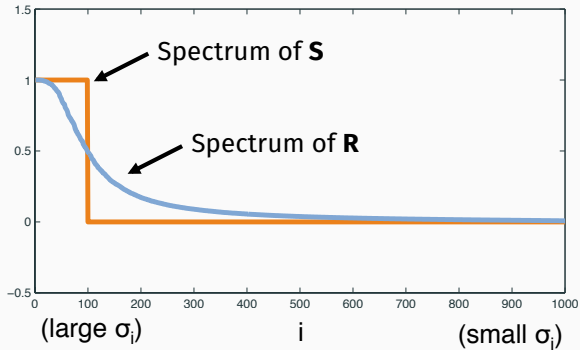
Natural model when Lanczos is combined with super-scalable randomized methods.

Powerful paradigm:

- $A = B^{-1}$ for some matrix B .
- Apply B^{-1} to vectors very quickly and approximately.

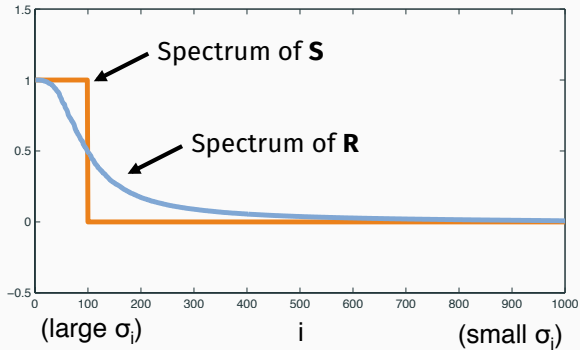
MATRIX STEP FUNCTION

Fastest algorithms for computing $\mathbf{S} = \text{step}_\lambda(\mathbf{A}^T \mathbf{A})$ actually compute $\text{step}_{1/2}(\mathbf{R})$ where $\mathbf{R} = (\mathbf{A}^T \mathbf{A} + \lambda \mathbf{I})^{-1} \mathbf{A}^T \mathbf{A}$.



MATRIX STEP FUNCTION

Fastest algorithms for computing $\mathbf{S} = \text{step}_\lambda(\mathbf{A}^T \mathbf{A})$ actually compute $\text{step}_{1/2}(\mathbf{R})$ where $\mathbf{R} = (\mathbf{A}^T \mathbf{A} + \lambda \mathbf{I})^{-1} \mathbf{A}^T \mathbf{A}$.



Most of the work is computing $\mathbf{R}\mathbf{x}$.

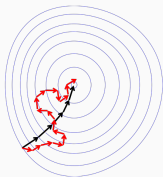
$Rx = (A^T A + \lambda I)^{-1} A^T A x$ is a convex optimization problem.

The diagram illustrates the mathematical expression $\|Aw - b\|_2^2 + \lambda \|w\|_2^2$ using colored rectangles and mathematical symbols. On the left, a tall yellow rectangle labeled 'A' is flanked by two vertical double lines. To its right is a small teal rectangle labeled 'w', followed by a minus sign and another tall yellow rectangle labeled 'b', which is also flanked by two vertical double lines. A superscript '2' is placed above the second set of double lines. To the right of this is a plus sign, followed by a Greek letter lambda, and then a teal rectangle labeled 'w' flanked by two vertical double lines. A superscript '2' is placed above the second set of double lines, and another '2' is placed below the second set of double lines.

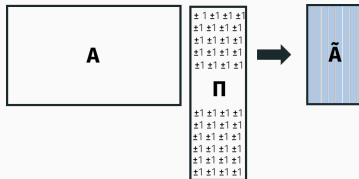
$$\|Aw - b\|_2^2 + \lambda \|w\|_2^2$$

LANCZOS AND RANDOMIZED METHODS

Lots of recent interest and new algorithms for convex problems on massive datasets (i.e. when \mathbf{A} does not fit in memory).



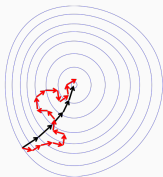
Stochastic Iterative
Methods



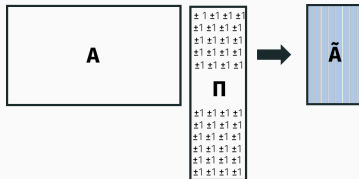
Randomized Sketching

LANCZOS AND RANDOMIZED METHODS

Lots of recent interest and new algorithms for convex problems on massive datasets (i.e. when \mathbf{A} does not fit in memory).



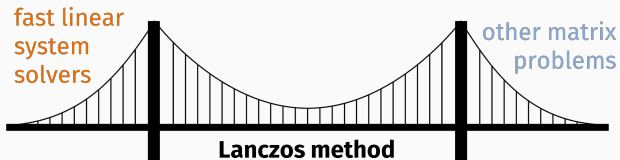
Stochastic Iterative
Methods



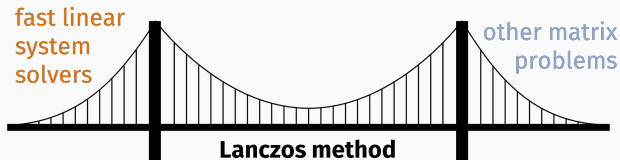
Randomized Sketching

Runtimes scale roughly as $O(\text{nnz}(\mathbf{A}) \cdot \log(1/\epsilon))$.
(for ϵ approximate solution)

LANCZOS AND RANDOMIZED METHODS



LANCZOS AND RANDOMIZED METHODS



- Faster eigenvector algorithms (in many regimes).
- Faster eigenvalue counting algorithms.
- Faster log-determinant and matrix norm algorithms.
- Faster balanced separator algorithms for graphs (via Laplacian matrix exponential).

We need to understand how the performance of our algorithms change when we replace every matrix-vector multiplication \mathbf{Ax} with an approximate solution.

We need to understand how the performance of our algorithms change when we replace every matrix-vector multiplication \mathbf{Ax} with an approximate solution.

Are matrix function algorithms stable?

We need to understand how the performance of our algorithms change when we replace every matrix-vector multiplication \mathbf{Ax} with an approximate solution.

Are matrix function algorithms stable?

Same stability questions were asked decades ago to understand roundoff error when computing \mathbf{Ax} !

$$fl(x \circ y) = (1 \pm \epsilon)(x \circ y) \text{ for } \circ = +, -, \times, \div$$

It is very easy to design iterative methods that converge very slowly when \mathbf{Ax} is computed approximately. But the Lanczos method (with no modifications) continues to perform well.

It is very easy to design iterative methods that converge very slowly when \mathbf{Ax} is computed approximately. But the Lanczos method (with no modifications) continues to perform well.

Can we explain this phenomena?

How can we apply polynomials in a stable way?

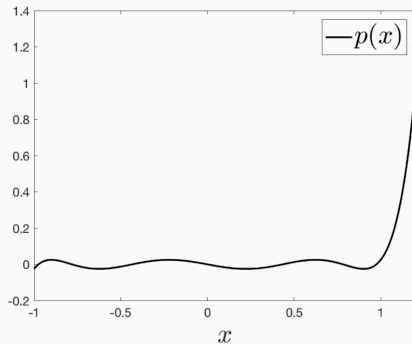
How can we apply polynomials in a stable way?

1. Want to compute $p(x) = c_0 + c_1x + \dots + c_kx^k$.
2. We do not know x , but we have access to a function `approxMult` that for any input z outputs:

$$\text{approxMult}(z) = z \cdot x + \epsilon.$$

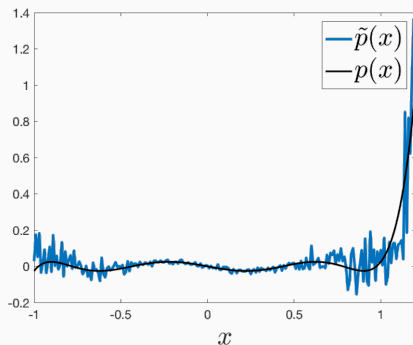
Goal: Compute $p(x) = 64x^7 - 112x^5 + 56x^3 - 7x$.

Using `approxMult` with $\epsilon = .05$.



Goal: Compute $p(x) = 64x^7 - 112x^5 + 56x^3 - 7x$.

Using `approxMult` with $\epsilon = .05$.

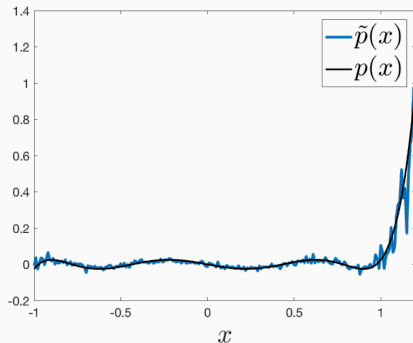


Directly compute and sum monomials.

$x^i = \text{approxMult}(\text{approxMult}(\dots \text{approxMult}(1)\dots))$

Goal: Compute $p(x) = 64x^7 - 112x^5 + 56x^3 - 7x$.

Using `approxMult` with $\epsilon = .05$.

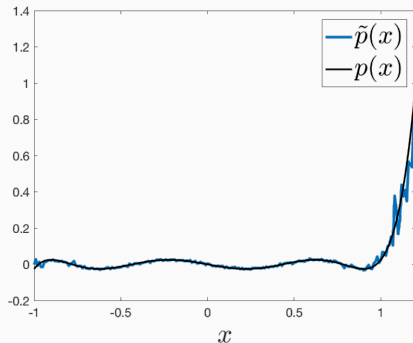


Factor $p(x) = (x - .98)(x - .78) \dots (x - .43)$.

$t_1 = (\text{approxMult}(1) - .98), t_2 = \text{approxMult}(t_1) - .78 \cdot t_1, \dots$

Goal: Compute $p(x) = 64x^7 - 112x^5 + 56x^3 - 7x$.

Using `approxMult` with $\epsilon = .05$.



Use special recurrence relation for this polynomial.

$$t_i = 2 \cdot \text{approxMult}(t_{i-1}) - t_{i-2}$$

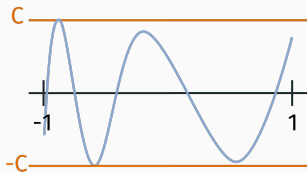
Assume we want to approximate $p(x)$ for $x \in [-1, 1]$.

Assume $|p(x)| \leq C$.

STABLE POLYNOMIAL COMPUTATION

Assume we want to approximate $p(x)$ for $x \in [-1, 1]$.

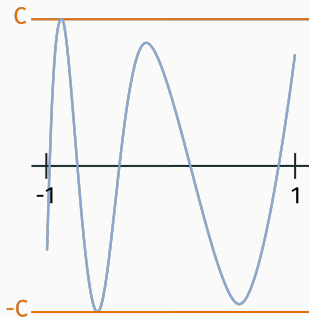
Assume $|p(x)| \leq C$.



STABLE POLYNOMIAL COMPUTATION

Assume we want to approximate $p(x)$ for $x \in [-1, 1]$.

Assume $|p(x)| \leq C$.



Claim

We can compute any $p(x)$ to accuracy $\epsilon \cdot Ck^3$ if `approxMult` has accuracy ϵ .

Compute monomials:

$$(X + \epsilon_1)$$

Compute monomials:

$$(X(X + \epsilon_1) + \epsilon_2)$$

Compute monomials:

$$(X(X(X + \epsilon_1) + \epsilon_2) + \epsilon_3)$$

Compute monomials:

$$X^i + X^{i-1}\epsilon_1 + X^{i-2}\epsilon_2 + \dots + \epsilon_j.$$

Compute monomials:

$$x^i + x^{i-1}\epsilon_1 + x^{i-2}\epsilon_2 + \dots + \epsilon_j.$$

Since $|x| \leq 1$, error on x^i bounded by $\epsilon_1 + \epsilon_2 + \dots + \epsilon_j \leq \epsilon i$.

Compute monomials:

$$x^i + x^{i-1}\epsilon_1 + x^{i-2}\epsilon_2 + \dots + \epsilon_i.$$

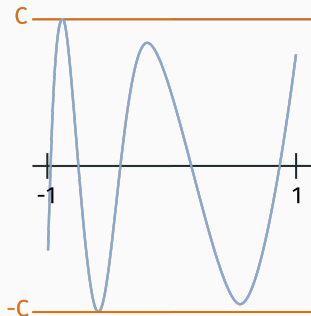
Since $|x| \leq 1$, error on x^i bounded by $\epsilon_1 + \epsilon_2 + \dots + \epsilon_i \leq \epsilon i$.

We can then compute $p(x) = c_0 + c_1x + \dots c_kx^k$ up to error:

$$c_1\epsilon + 2 \cdot c_2\epsilon + \dots + k \cdot c_k\epsilon \leq \epsilon k \cdot \sum_{i=1}^k |c_k|$$

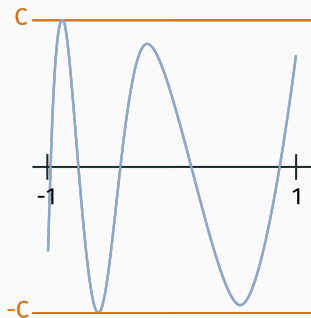
FIRST ATTEMPT

$\sum_{i=1}^k |c_k|$ can be far larger than our goal of $\epsilon \cdot Ck^3$.



FIRST ATTEMPT

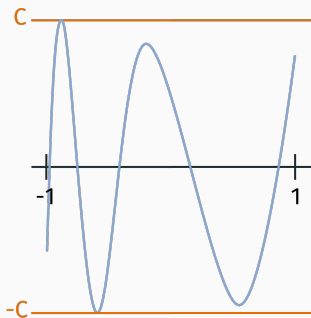
$\sum_{i=1}^k |c_k|$ can be far larger than our goal of $\epsilon \cdot Ck^3$.



There are polynomials with $C = 1$ but $\sum_{i=1}^k |c_k| = O(2^k)$.

FIRST ATTEMPT

$\sum_{i=1}^k |c_k|$ can be far larger than our goal of $\epsilon \cdot Ck^3$.

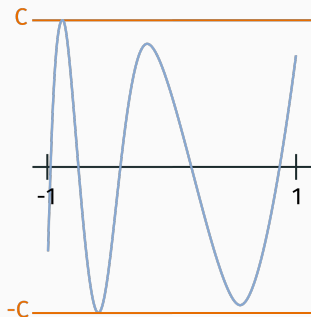


There are polynomials with $C = 1$ but $\sum_{i=1}^k |c_k| = O(2^k)$.

Exponential instead of polynomial loss in k .

FIRST ATTEMPT

$\sum_{i=1}^k |c_k|$ can be far larger than our goal of $\epsilon \cdot Ck^3$.



There are polynomials with $C = 1$ but $\sum_{i=1}^k |c_k| = O(2^k)$.

Exponential instead of polynomial loss in k .

Runtimes of randomized system solvers depended on $\log(1/\epsilon)$.

What are those polynomials?

“BAD” POLYNOMIALS

What are those polynomials?

Chebyshev polynomials of the first kind.

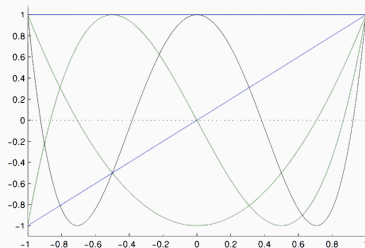
$$T_0(x) = 1$$

$$T_1(x) = x$$

$$T_2(x) = 2x^2 - 1$$

$$\vdots$$

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$$



What are those polynomials?

Chebyshev polynomials of the first kind.

$$T_0(x) = 1$$

$$T_1(x) = x$$

$$T_2(x) = 2x^2 - 1$$

$$T_3(x) = 4x^3 - 3x$$

$$T_4(x) = 8x^4 - 8x^2 + 1$$

$$T_5(x) = 16x^5 - 20x^3 + 5x$$

$$T_6(x) = 32x^6 - 48x^4 + 18x^2 - 1$$

$$T_7(x) = 64x^7 - 112x^5 + 56x^3 - 7x$$

$$T_8(x) = 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1$$

$$T_9(x) = 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x$$

$$T_{10}(x) = 512x^{10} - 1280x^8 + 1120x^6 - 400x^4 + 50x^2 - 1$$

$$T_{11}(x) = 1024x^{11} - 2816x^9 + 2816x^7 - 1232x^5 + 220x^3 - 11x$$

What are those polynomials?

Chebyshev polynomials of the first kind.

$$T_0(x) = 1$$

$$T_1(x) = x$$

$$T_2(x) = 2x^2 - 1$$

$$T_3(x) = 4x^3 - 3x$$

$$T_4(x) = 8x^4 - 8x^2 + 1$$

$$T_5(x) = 16x^5 - 20x^3 + 5x$$

$$T_6(x) = 32x^6 - 48x^4 + 18x^2 - 1$$

$$T_7(x) = 64x^7 - 112x^5 + 56x^3 - 7x$$

$$T_8(x) = 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1$$

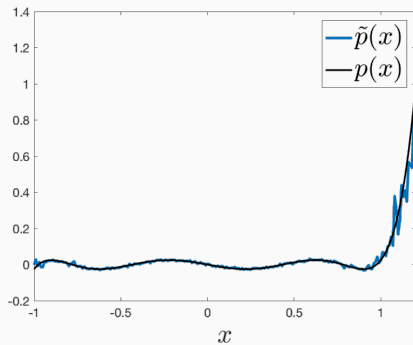
$$T_9(x) = 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x$$

$$T_{10}(x) = 512x^{10} - 1280x^8 + 1120x^6 - 400x^4 + 50x^2 - 1$$

$$T_{11}(x) = 1024x^{11} - 2816x^9 + 2816x^7 - 1232x^5 + 220x^3 - 11x$$

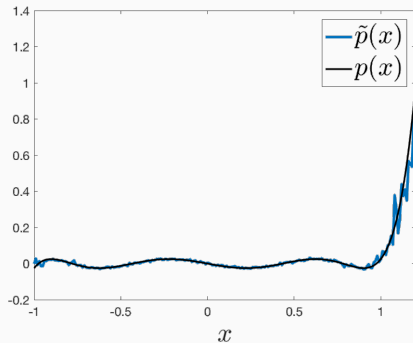
We can apply these in a stable way, using their recurrence!

“GOOD” POLYNOMIALS?



$$t_i = 2 \cdot \text{approxMult}(t_{i-1}) - t_{i-2}$$

“GOOD” POLYNOMIALS?



$$t_i = 2 \cdot \text{approxMult}(t_{i-1}) - t_{i-2}$$

Not hard to show that when computing $T_k(x)$ the error $\leq \epsilon k^2$.

Chebyshev polynomials are the only hard case.

Chebyshev polynomials are the only hard case.

Property: If a degree k polynomial $p(x)$ is bounded by C on $[-1, 1]$, it can be written as

$$p(x) = c_0 T_0(x) + c_1 T_1(x) + \dots + c_k T_k(x)$$

where every $c_i \leq C$.

Chebyshev polynomials are the only hard case.

Property: If a degree k polynomial $p(x)$ is bounded by C on $[-1, 1]$, it can be written as

$$p(x) = c_0 T_0(x) + c_1 T_1(x) + \dots + c_k T_k(x)$$

where every $c_i \leq C$.

Total error of sum $p(x)$ is bounded by

$$C \cdot 1^2 \epsilon + C \cdot 2^2 \epsilon + \dots + C \cdot k^2 \epsilon \leq C k^3 \epsilon.$$

Same arguments extends from scalar polynomials to matrix polynomials.

Same arguments extends from scalar polynomials to matrix polynomials. Framework allows us to analyze Lanczos as well.

Same arguments extends from scalar polynomials to matrix polynomials. Framework allows us to analyze Lanczos as well.

Step 1: Lanczos stably applies Chebyshev polynomials (building on results of Paige ['71, '76, '80]).

Same arguments extends from scalar polynomials to matrix polynomials. **Framework allows us to analyze Lanczos as well.**

Step 1: Lanczos stably applies Chebyshev polynomials (building on results of Paige ['71, '76, '80]).

Step 2: By linearity, Lanczos stably applies polynomials bounded by C .

Same arguments extends from scalar polynomials to matrix polynomials. **Framework allows us to analyze Lanczos as well.**

Step 1: Lanczos stably applies Chebyshev polynomials (building on results of Paige ['71, '76, '80]).

Step 2: By linearity, Lanczos stably applies polynomials bounded by C .

Step 3: If $|f(x)| \leq C$, a good approximating polynomial has $|p(x)| \leq O(C)$, so Lanczos is stable for bounded functions.

Same arguments extends from scalar polynomials to matrix polynomials. **Framework allows us to analyze Lanczos as well.**

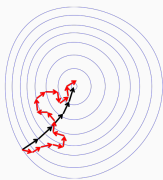
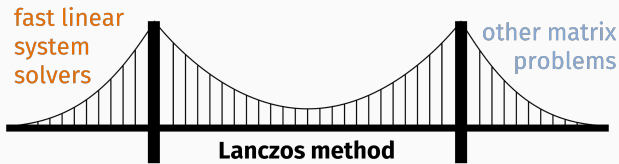
Step 1: Lanczos stably applies Chebyshev polynomials (building on results of Paige ['71, '76, '80]).

Step 2: By linearity, Lanczos stably applies polynomials bounded by C .

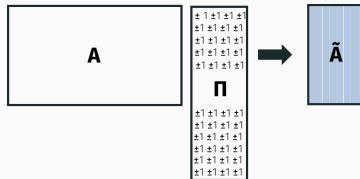
Step 3: If $|f(x)| \leq C$, a good approximating polynomial has $|p(x)| \leq O(C)$, so Lanczos is stable for bounded functions.

Use Lanczos without fear (on bounded functions)!

STABILITY OF LANCZOS



Stochastic Iterative
Methods



Randomized Sketching

See paper for applications to step function, matrix exponential, top eigenvector, etc.

Answer to old question on Lanczos in finite precision:

Theorem (Lanczos is stable for any bounded function)

If $|f(x)| \leq C$ for $x \in [\lambda_{\min}(\mathbf{A}), \lambda_{\max}(\mathbf{A})]$, then if Lanczos is run for k iterations on a computer with $O(\log(nC\kappa))$ bits of precision, it outputs a vector \mathbf{y} such that

$$\|f(\mathbf{A})\mathbf{x} - \mathbf{y}\| \leq 7k \cdot \delta_k \cdot \|\mathbf{x}\|$$

where δ_k is the error of the best degree k uniform approximation to f .

Answer to old question on Lanczos in finite precision:

Theorem (Lanczos is stable for any bounded function)

If $|f(x)| \leq C$ for $x \in [\lambda_{\min}(\mathbf{A}), \lambda_{\max}(\mathbf{A})]$, then if Lanczos is run for k iterations on a computer with $O(\log(nC\kappa))$ bits of precision, it outputs a vector \mathbf{y} such that

$$\|f(\mathbf{A})\mathbf{x} - \mathbf{y}\| \leq 7k \cdot \delta_k \cdot \|\mathbf{x}\|$$

where δ_k is the error of the best degree k uniform approximation to f .

- Compare to $\|f(\mathbf{A})\mathbf{x} - \mathbf{y}\| \leq 2 \cdot \delta_k \cdot \|\mathbf{x}\|$ in exact arithmetic.

Answer to old question on Lanczos in finite precision:

Theorem (Lanczos is stable for any bounded function)

If $|f(x)| \leq C$ for $x \in [\lambda_{\min}(\mathbf{A}), \lambda_{\max}(\mathbf{A})]$, then if Lanczos is run for k iterations on a computer with $O(\log(nC\kappa))$ bits of precision, it outputs a vector \mathbf{y} such that

$$\|f(\mathbf{A})\mathbf{x} - \mathbf{y}\| \leq 7k \cdot \delta_k \cdot \|\mathbf{x}\|$$

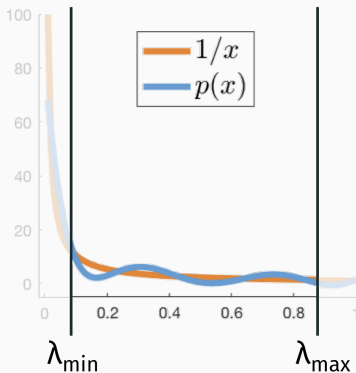
where δ_k is the error of the best degree k uniform approximation to f .

- Compare to $\|f(\mathbf{A})\mathbf{x} - \mathbf{y}\| \leq 2 \cdot \delta_k \cdot \|\mathbf{x}\|$ in exact arithmetic.
- Matches known bound for $\mathbf{A}^{-1}\mathbf{x}$ (Greenbaum, '89).

NEGATIVE RESULT FOR
LINEAR SYSTEMS

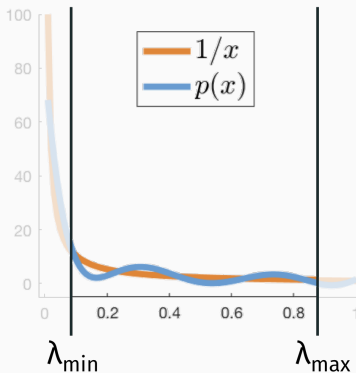
LANCZOS FOR LINEAR SYSTEMS

We proved earlier that Lanczos always matches the best uniform approximating polynomial for $f(x)$:



LANCZOS FOR LINEAR SYSTEMS

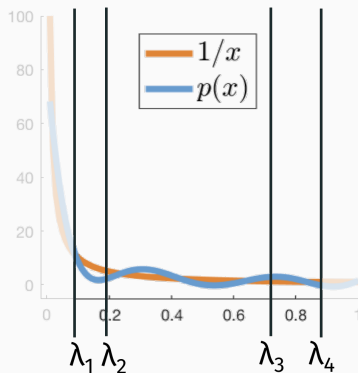
We proved earlier that Lanczos always matches the best uniform approximating polynomial for $f(x)$:



For linear systems it actually does better than that.

LANCZOS FOR LINEAR SYSTEMS

We proved earlier that Lanczos always matches the best uniform approximating polynomial for $f(x)$:



For linear systems it actually does better than that.

- The best **uniform** approximation to $1/x$ has degree $\sqrt{\lambda_{\max} / \lambda_{\min}} \cdot \log(1/\epsilon)$.

- The best **uniform** approximation to $1/x$ has degree $\sqrt{\lambda_{\max} / \lambda_{\min}} \cdot \log(1/\epsilon)$.
- $1/x$ can be represented exactly by a degree $n - 1$ polynomial if **A** only has n eigenvalues.

- The best **uniform** approximation to $1/x$ has degree $\sqrt{\lambda_{\max} / \lambda_{\min}} \cdot \log(1/\epsilon)$.
- $1/x$ can be represented exactly by a degree $n - 1$ polynomial if **A** only has n eigenvalues.

Claim: On exact arithmetic computers, linear systems can be solved in $O(\text{nnz}(\mathbf{A}) \cdot n)$ time (i.e. n iterations of Lanczos)

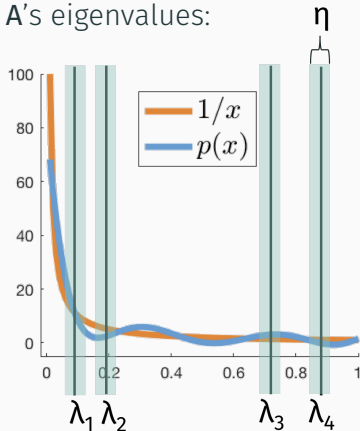
- The best **uniform** approximation to $1/x$ has degree $\sqrt{\lambda_{\max} / \lambda_{\min}} \cdot \log(1/\epsilon)$.
- $1/x$ can be represented exactly by a degree $n - 1$ polynomial if **A** only has n eigenvalues.

Claim: On exact arithmetic computers, linear systems can be solved in $O(\text{nnz}(\mathbf{A}) \cdot n)$ time (i.e. n iterations of Lanczos)

Research question: To what extent does this bound hold true in finite precision? Are $n \log n$ iterations sufficient? n^2 ?

LINEAR SYSTEMS IN FINITE PRECISION

Greenbaum (1989): Finite precision Lanczos and conjugate gradient match the best polynomial approximating $1/x$ in **tiny** intervals around \mathbf{A} 's eigenvalues:



η is on the order of machine precision!

Theorem (Stable polynomial lower bound.)

For any n , there is a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ with condition number $\lambda_{\max} / \lambda_{\min}$ such that no k degree polynomial satisfies Greenbaum's condition with error $\leq 1/3$ for all

$$k \leq (\lambda_{\max} / \lambda_{\min})^{1/5}$$

even when $\eta \leq \frac{1}{2^{n/\log \kappa}}$.

Theorem (Stable polynomial lower bound.)

For any n , there is a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ with condition number $\lambda_{\max} / \lambda_{\min}$ such that no k degree polynomial satisfies Greenbaum's condition with error $\leq 1/3$ for all

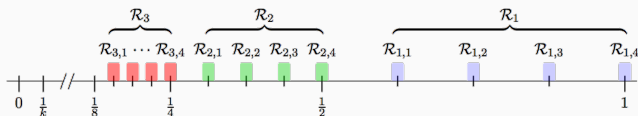
$$k \leq (\lambda_{\max} / \lambda_{\min})^{1/5}$$

even when $\eta \leq \frac{1}{2^{n/\log \kappa}}$.

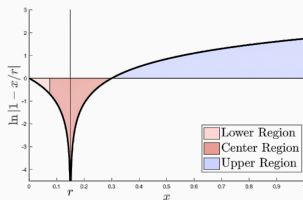
In other words, we cannot avoid polynomial dependence on condition number unless we have nearly n bits of precision.

LOWER BOUND

Construction: Eigenvalues roughly uniform on geometric scale.



Proof: Simple potential function argument.



OPEN QUESTIONS

- Can $(\lambda_{\max} / \lambda_{\min})^{1/5}$ be tightened to $(\lambda_{\max} / \lambda_{\min})^{1/2}$
- Does Greenbaum's estimate fully characterize Lanczos?
Can the lower bound be extended to an actual runtime lower bound?
- How about for a more general class of algorithms? Any method accessing **A** only through noisy matrix-vector products?

THANK YOU!