Stability of the Lanczos Method

Christopher Musco (joint with Aaron Sidford and Cameron Musco)

Princeton University.

The Lanczos Method

The Lanczos Method

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

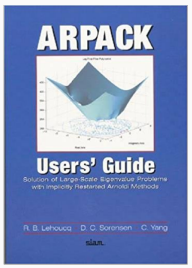
The Lanczos Method

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

Eigen

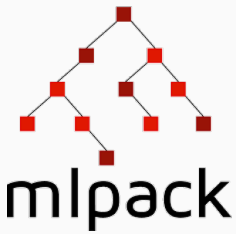


SciPy

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.

LANCZOS IN DATA SCIENCE

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

Require understanding performance under noisy inputs.

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

Require understanding performance under noisy inputs.

Today's results:

1. Lanczos is very noise stable, performing essentially optimally for many matrix functions.

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

Require understanding performance under noisy inputs.

Today's results:

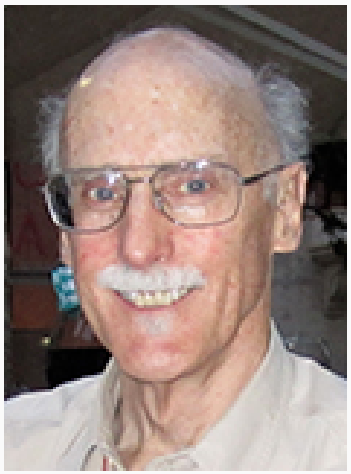
1. Lanczos is very noise stable, performing essentially optimally for many matrix functions.
2. Except when solving linear systems! We provide a strong lower bound showing that noise can significantly impair Lanczos and the closely related conjugate gradient method.

STABILITY OF THE LANCZOS METHOD

The stability of Lanczos methods implemented in finite precision has been studied since the 70s and 80s.

STABILITY OF THE LANCZOS METHOD

The stability of Lanczos methods implemented in finite precision has been studied since the 70s and 80s.



Christopher Paige

Anne Greenbaum

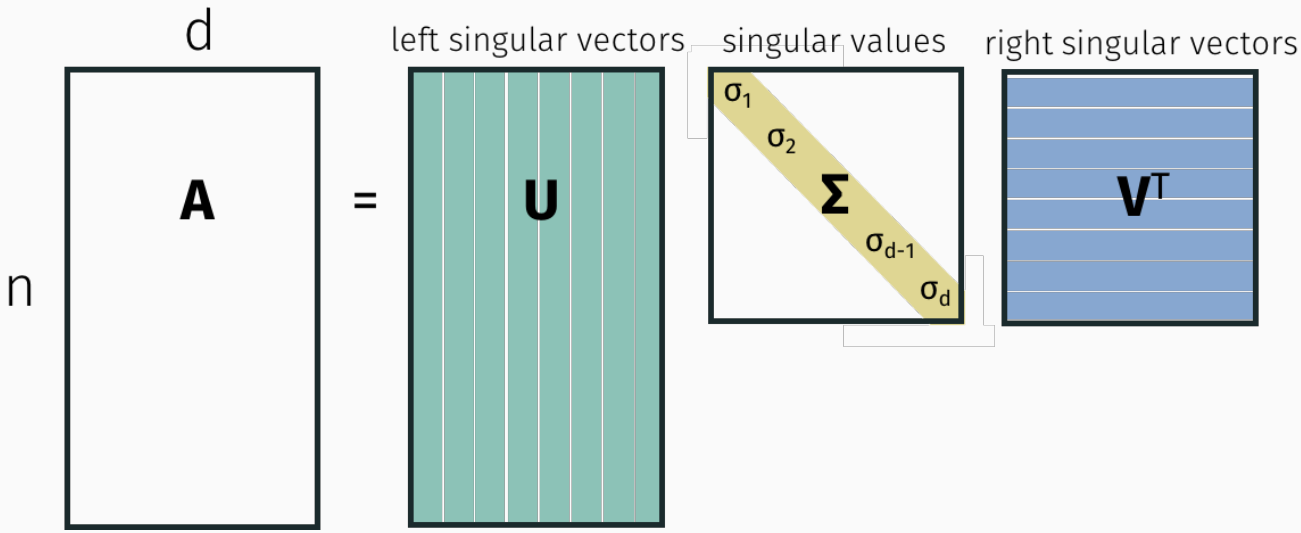
Vladimir Druskin

Leonid Knizhnerman

WHAT IS A MATRIX FUNCTION?

WHAT IS A MATRIX FUNCTION?

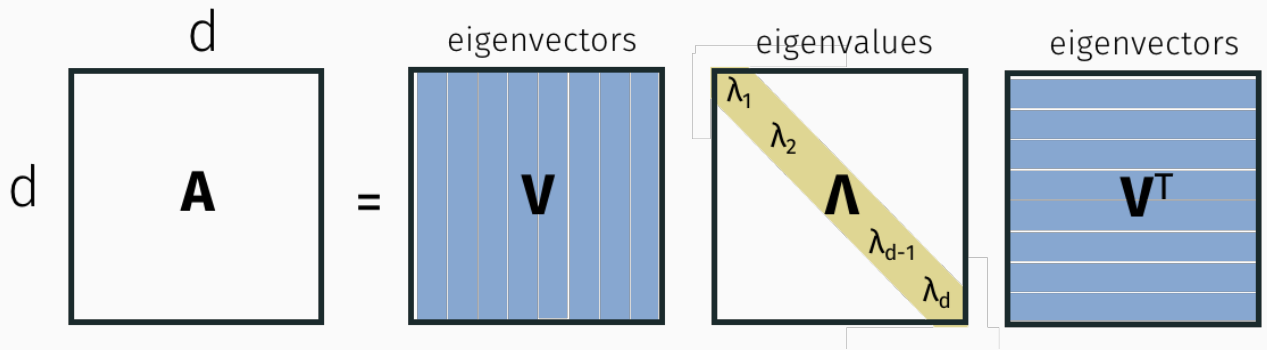
Every matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ has a singular value decomposition:



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

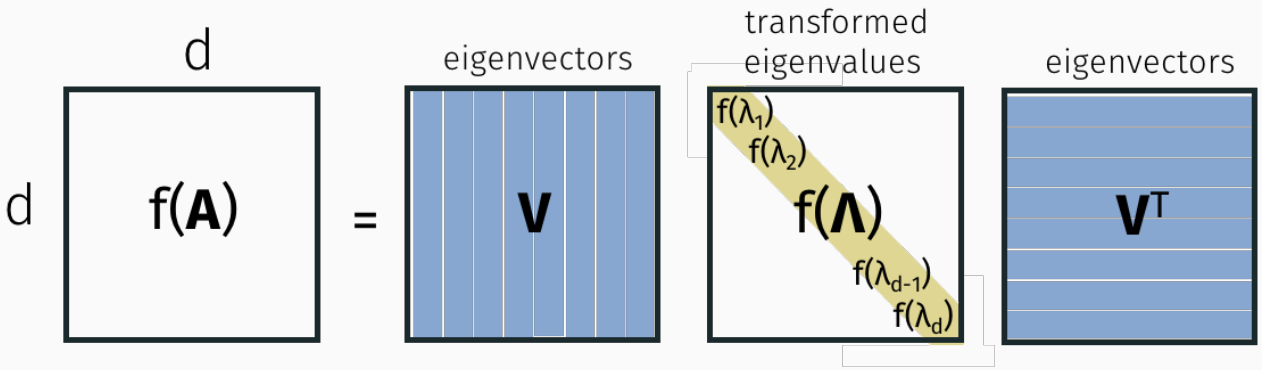
WHAT IS A MATRIX FUNCTION?

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



WHAT IS A MATRIX FUNCTION?

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



EXAMPLE MATRIX FUNCTION

$$\mathbf{A}^{-1} = f(\mathbf{A}) \text{ where } f(\lambda) = 1/\lambda$$

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

Cost to compute $f(A)$:

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

$$\underbrace{O(n^3)}$$

eigendecompose $\mathsf{A} = \mathsf{V}\boldsymbol{\Lambda}\mathsf{V}^T$

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

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

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

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

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

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

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

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

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

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

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)

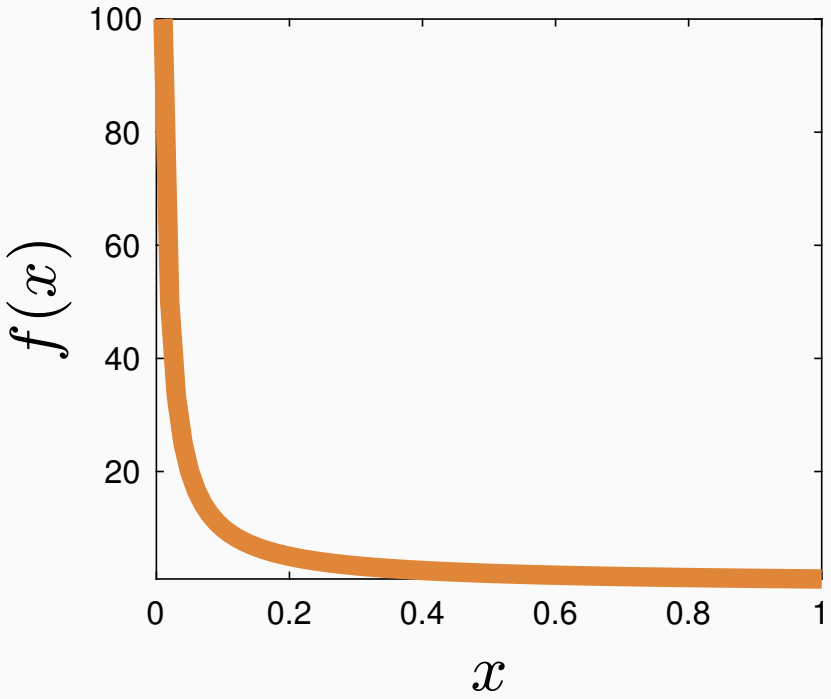
MATRIX INVERSE

Example

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

Function

$f(x) = 1/x$



Countless applications...

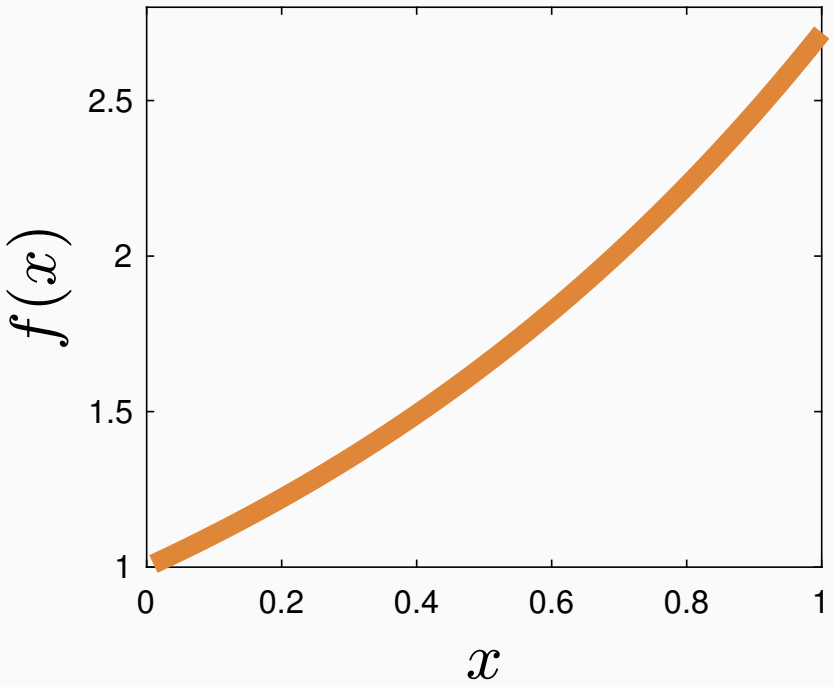
MATRIX EXPONENTIAL

Example

Matrix exponential, $e^{A}x$

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)

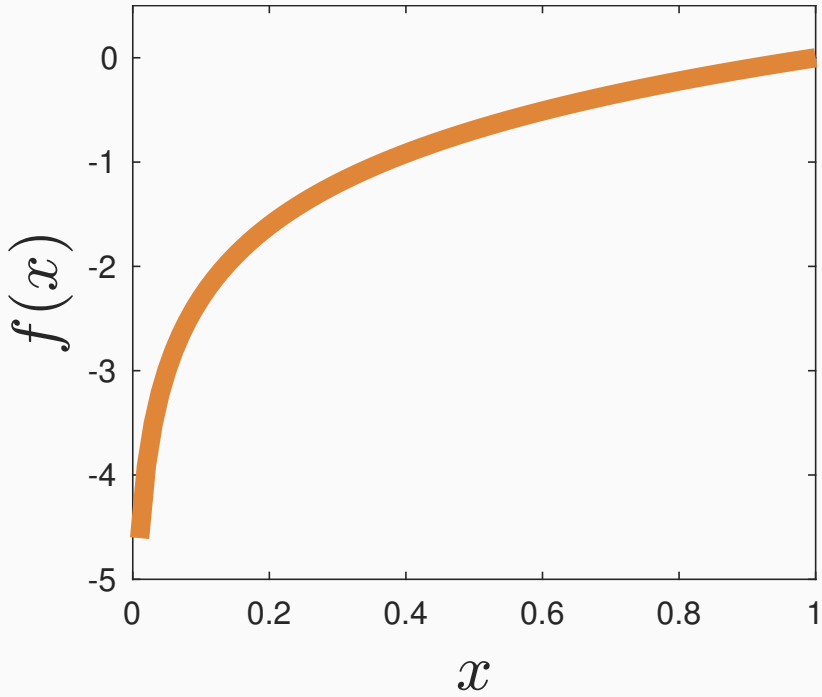
MATRIX LOG

Example

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

Function

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



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

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]

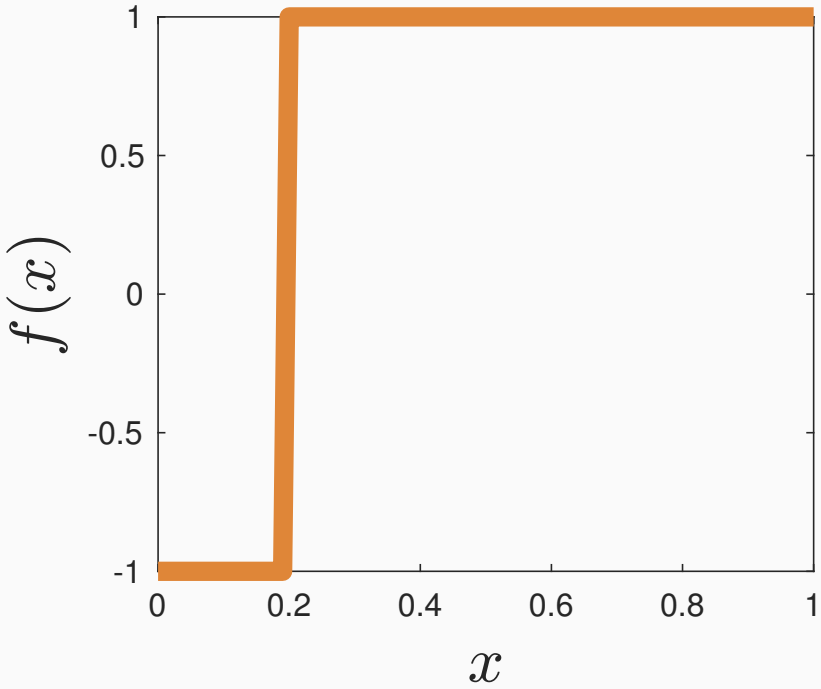
MATRIX STEP FUNCTION

Example

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

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]

PRINCIPAL COMPONENT REGRESSION

Standard Regression:

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

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

Principal Component Regression

Standard Regression:

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

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

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:

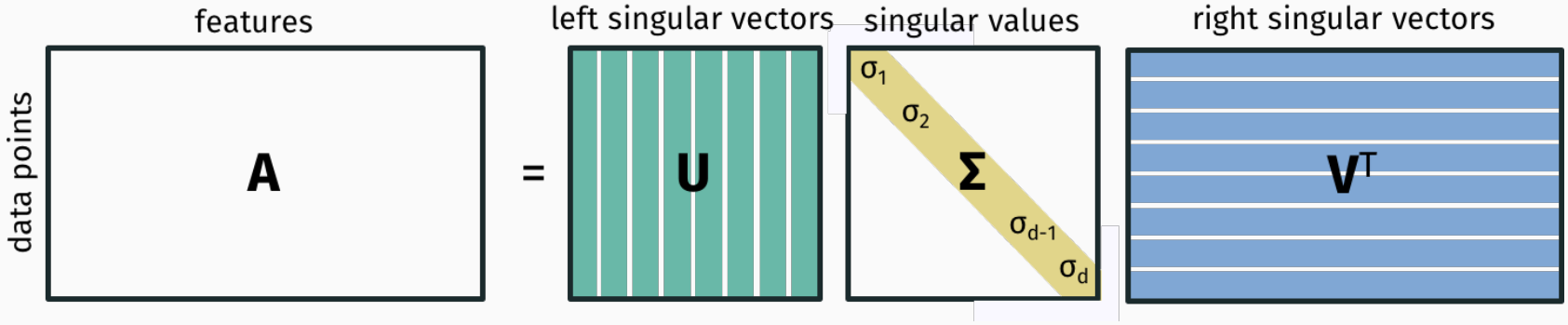
Given: $\mathbf{A}$, $\mathbf{b}$

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

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:

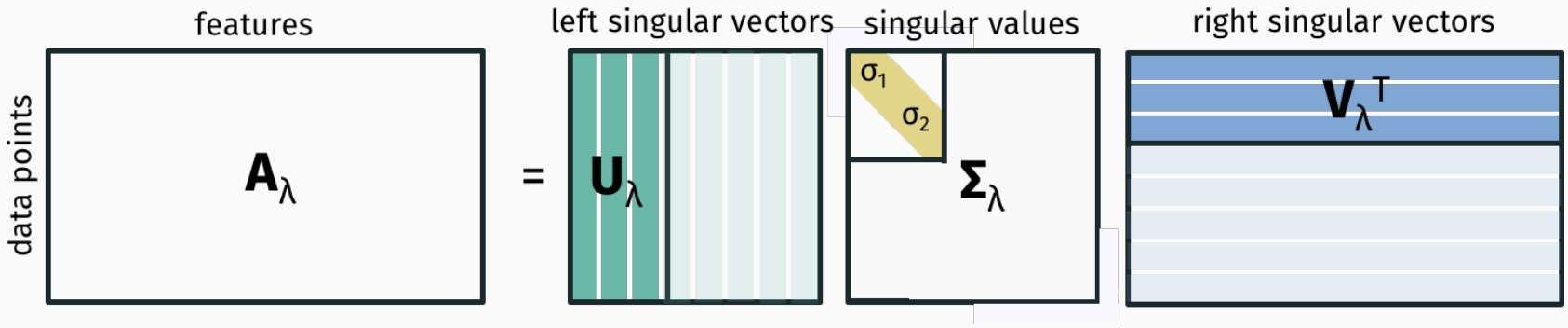
Given: $\mathbf{A}$, $\mathbf{b}$

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

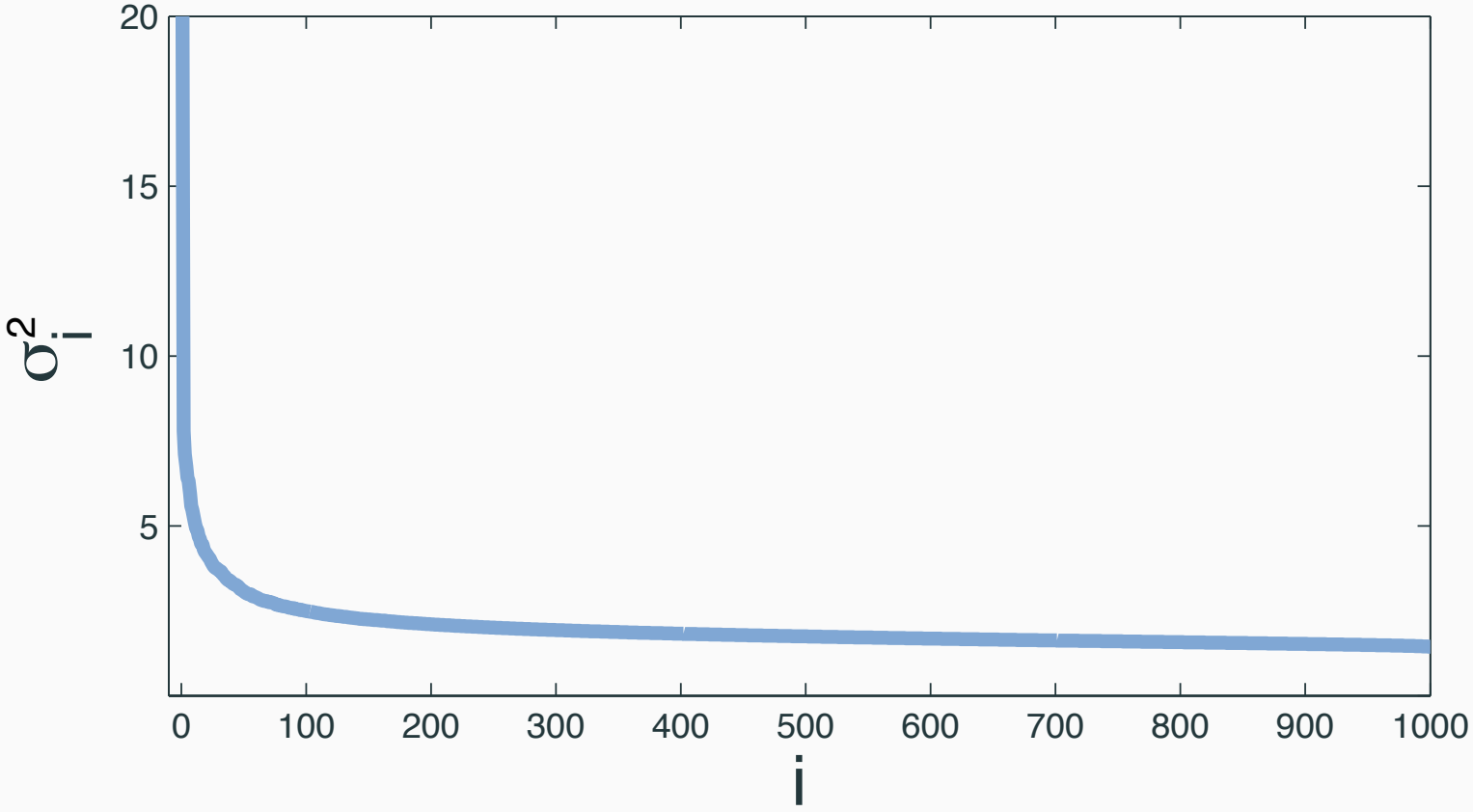
Principal Component Regression:

Given: $\mathbf{A}$, $\mathbf{b}$, λ

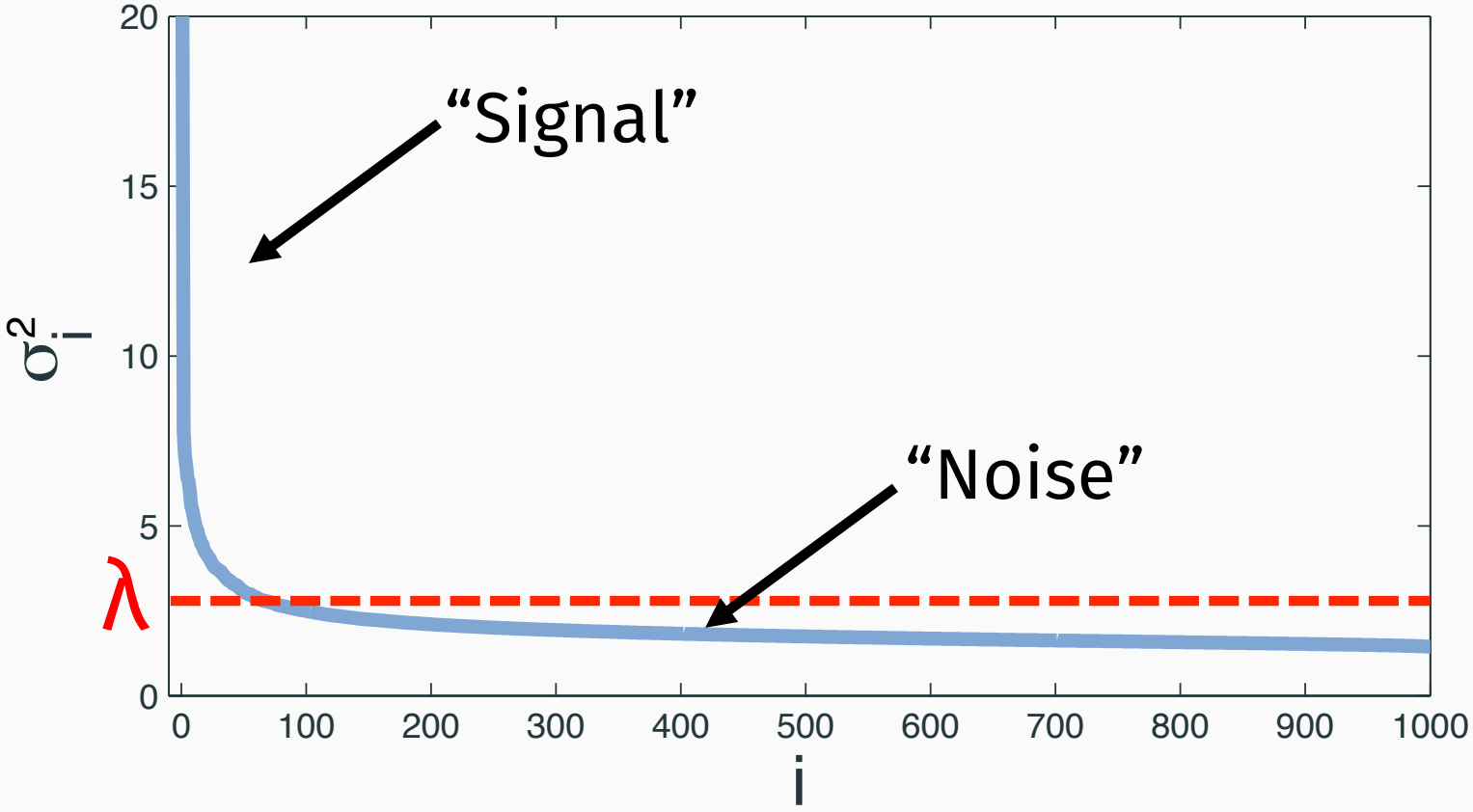
Solve: $\mathbf{x}^* = \arg\min_{\mathbf{x}} \|\mathbf{A}_\lambda \mathbf{x} - \mathbf{b}\|^2$



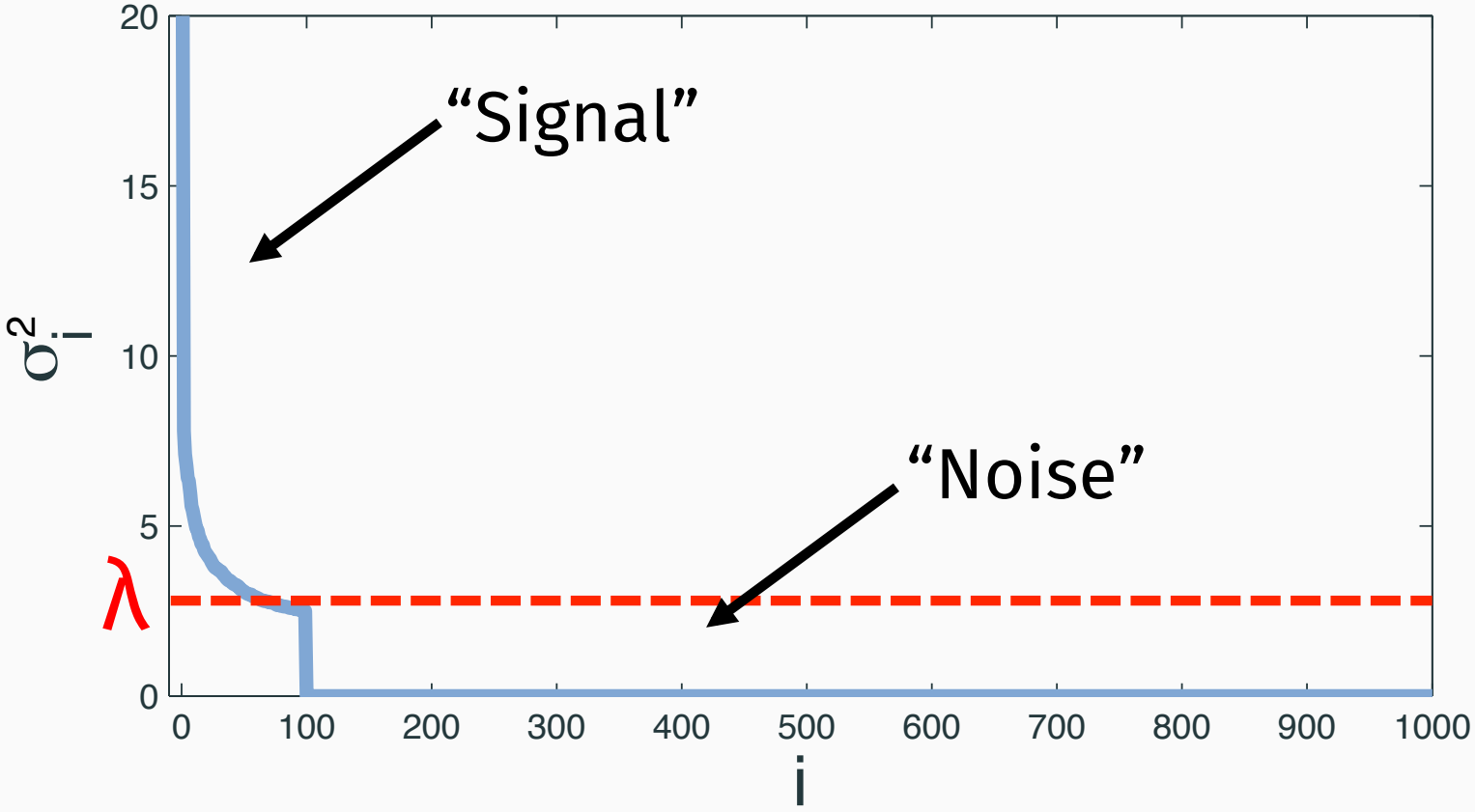
Singular values of **A**



Singular values of **A**



Singular values of $\mathbf{A}_\lambda$



Principal Component Regression (PCR):

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

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

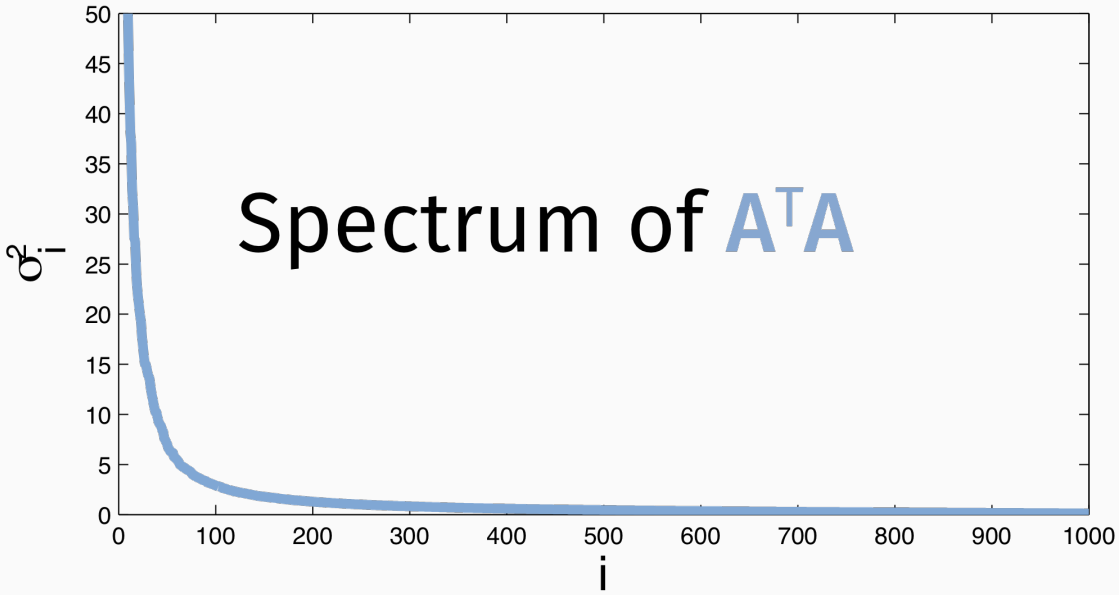
Principal Component Regression (PCR):

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

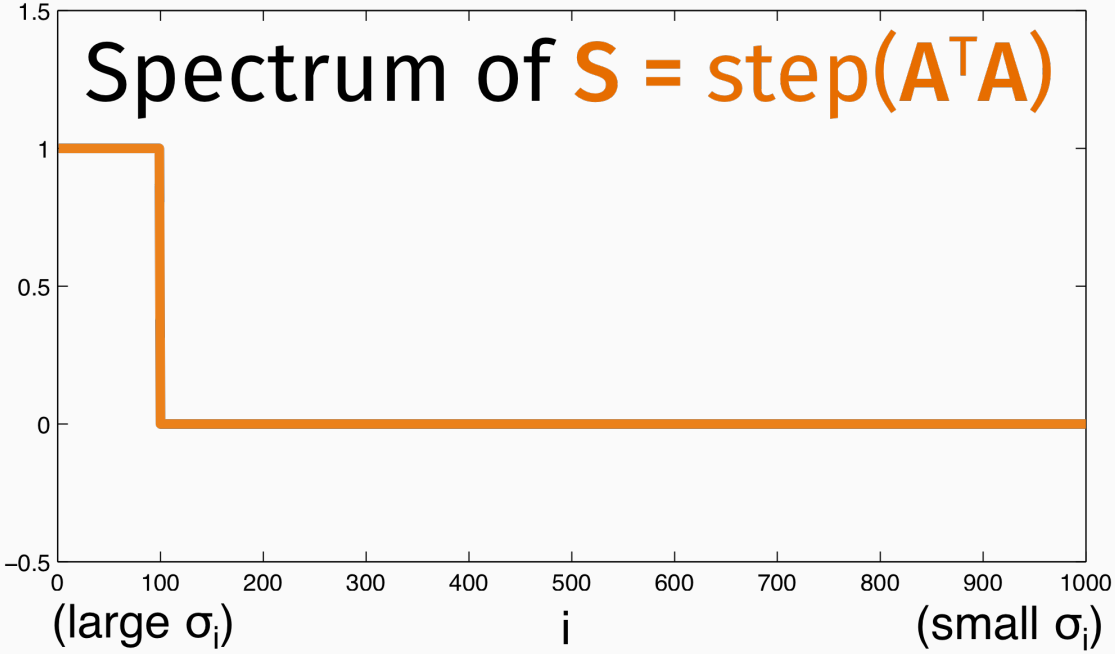
Solution: $\mathbf{x} = (\mathbf{A}_\lambda^T \mathbf{A}_\lambda)^{-1} \mathbf{A}^T \mathbf{b}$

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



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



FAST ALGORITHMS FOR MATRIX FUNCTIONS

MATRIX POLYNOMIALS

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

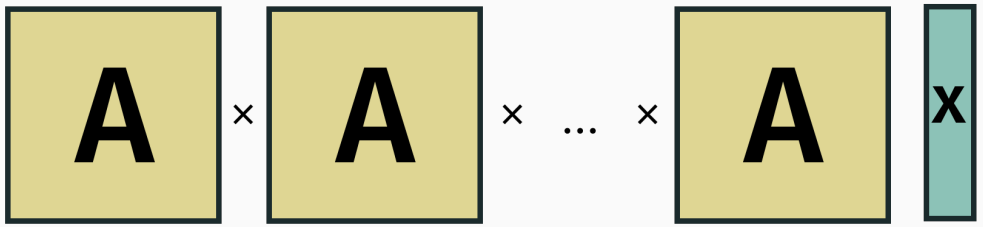
Matrix polynomials can be computed iteratively.

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

Matrix polynomials can be computed iteratively.

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

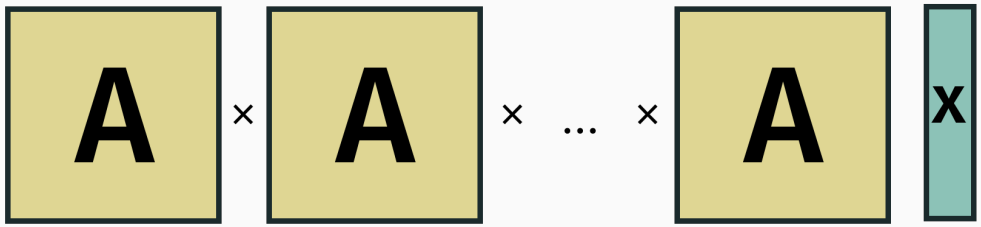
$$A^k x = V\Lambda V^T V\Lambda V^T \cdots V\Lambda V^T x = V\Lambda^k V^T x$$



Matrix **polynomials** can be computed iteratively.

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

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



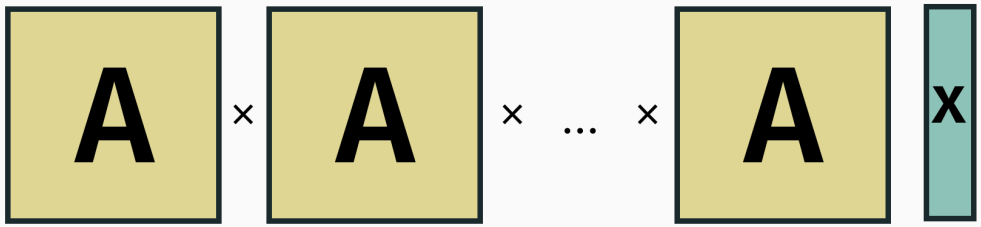
Total time to compute $p(A)x = c_0 x + c_1 Ax + c_2 A^2 x + \ldots + c_k A^k x$:

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

Matrix **polynomials** can be computed iteratively.

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

$$\mathbf{A}^k\mathbf{x} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T\mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T \cdots \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T\mathbf{x} = \mathbf{V}\boldsymbol{\Lambda}^k\mathbf{V}^T\mathbf{x}$$



Total time to compute $p(\mathbf{A})\mathbf{x} = c_0\mathbf{x} + c_1\mathbf{A}\mathbf{x} + c_2\mathbf{A}^2\mathbf{x} + \ldots + c_k\mathbf{A}^k\mathbf{x}$:

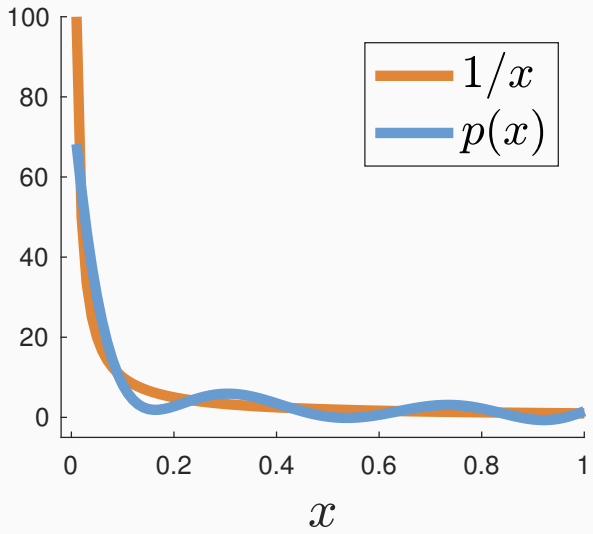
$$O(k \cdot \text{nnz}(\mathbf{A})) \leq O(k \cdot n^2) \ll O(n^3)$$

For general matrix functions:

approximate $f(x)$ with low-degree polynomial $p(x)$.

For general matrix functions:

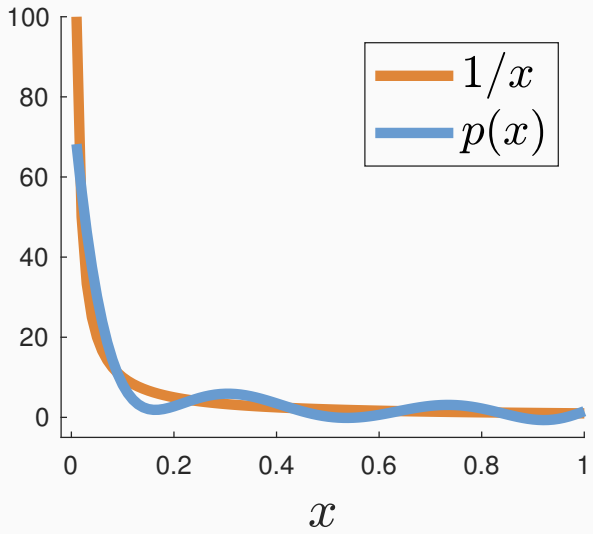
approximate $f(x)$ with low-degree polynomial $p(x)$.



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

For general matrix functions:

approximate $f(x)$ with low-degree polynomial $p(x)$.



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

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

POLYNOMIAL APPROXIMATION

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

POLYNOMIAL APPROXIMATION

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

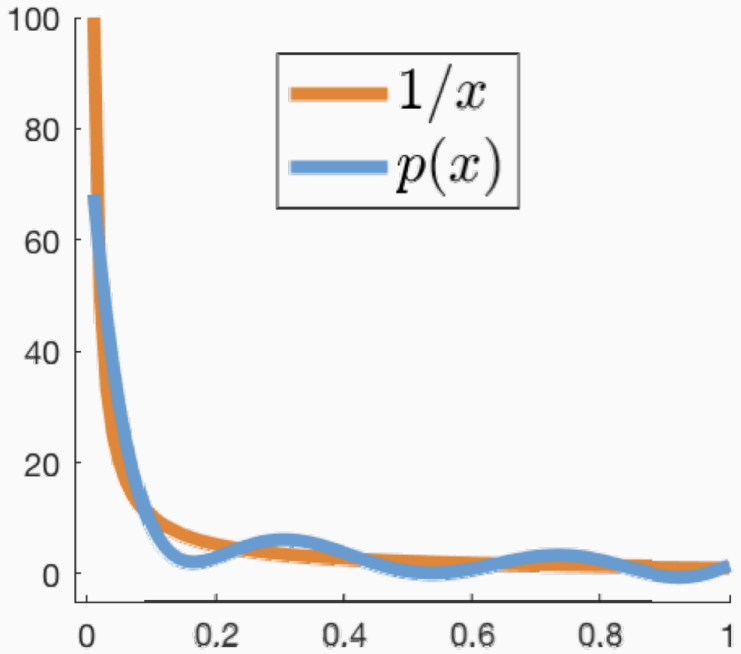
where

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

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

where

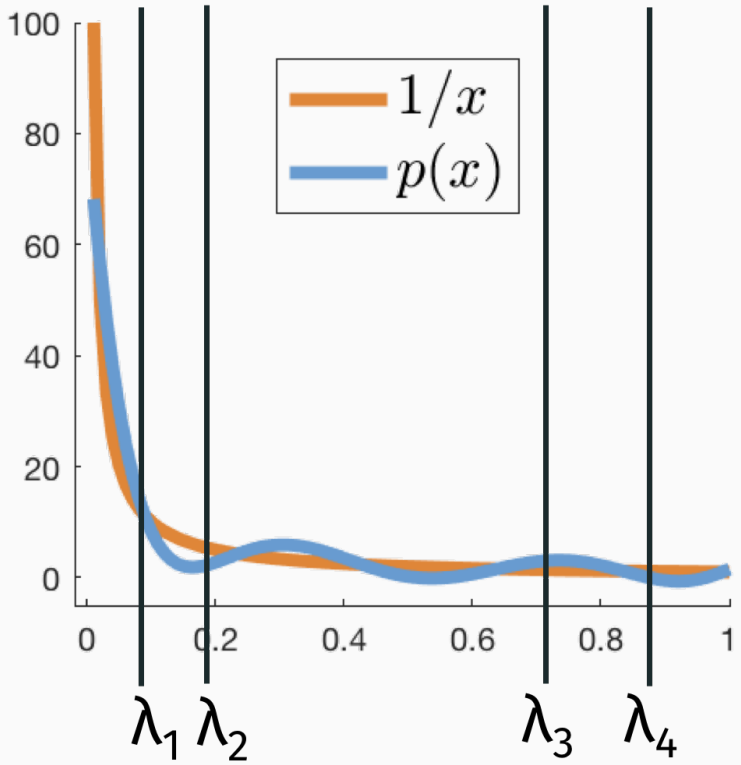
$$\epsilon = \max_{i=1,\ldots,n} |f(\lambda_i) - p(\lambda_i)|.$$



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

where

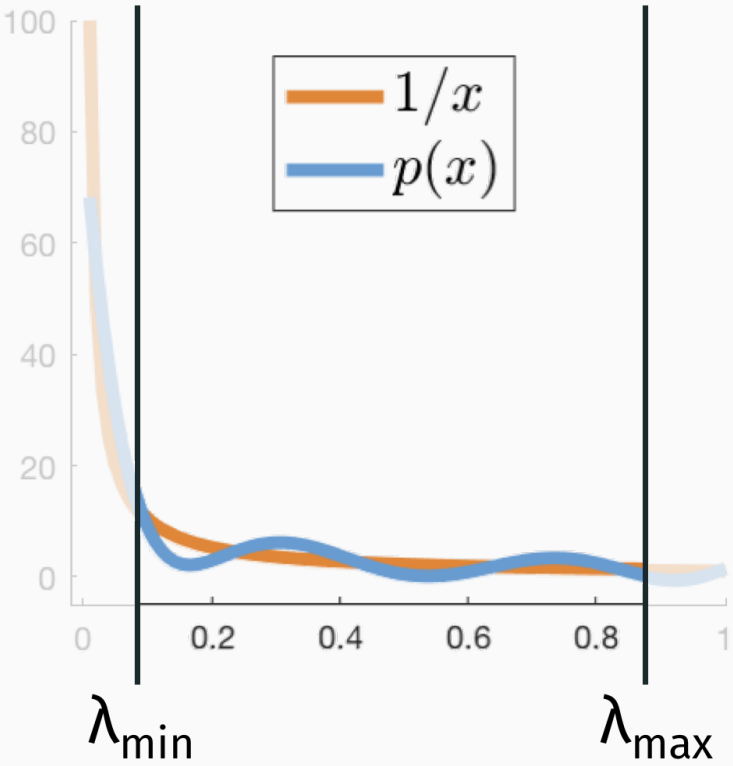
$$\epsilon = \max_{i=1,\ldots,n} |f(\lambda_i) - p(\lambda_i)|.$$



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

where

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



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.

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

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

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.

But no one actually uses Chebyshev interpolation...

THE LANCZOS METHOD

FOR MATRIX FUNCTIONS

THE LANCZOS METHOD

Cornelius Lanczos, 1950

THE LANCZOS METHOD

Cornelius Lanczos, 1950

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

THE LANCZOS METHOD

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.

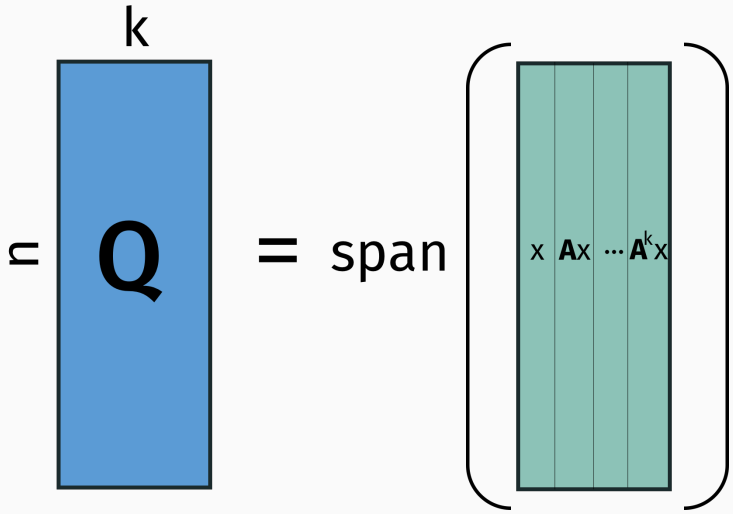
THE LANCZOS METHOD

Cornelius Lanczos, 1950

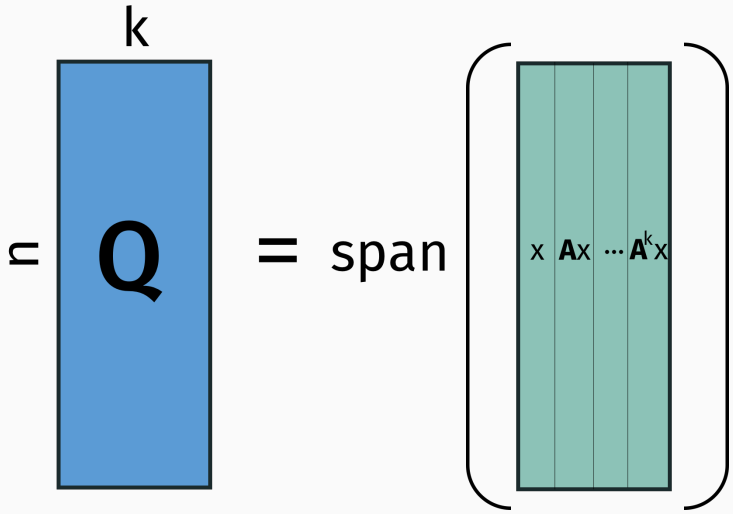
- Simple to implement.
- No need to know $\lambda_{\min}(\mathbf{A})$ and $\lambda_{\max}(\mathbf{A})$.
- Much better convergence in practice (for many reasons).
- Matches optimal uniform approximation up to factor 2.

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

Form orthogonal span for Krylov subspace $\mathcal{K} = \{\mathbf{x}, \mathbf{A}\mathbf{x}, \ldots \mathbf{A}^k\mathbf{x}\}$:

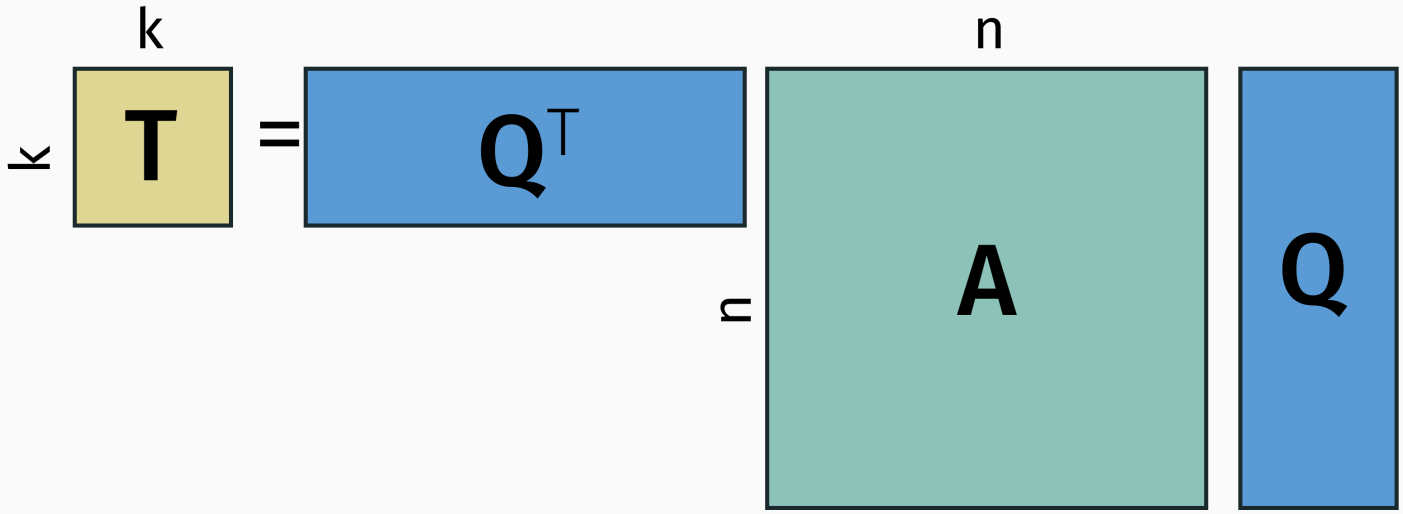


Form orthogonal span for Krylov subspace $\mathcal{K} = \{\mathbf{x}, \mathbf{A}\mathbf{x}, \ldots \mathbf{A}^k\mathbf{x}\}$:



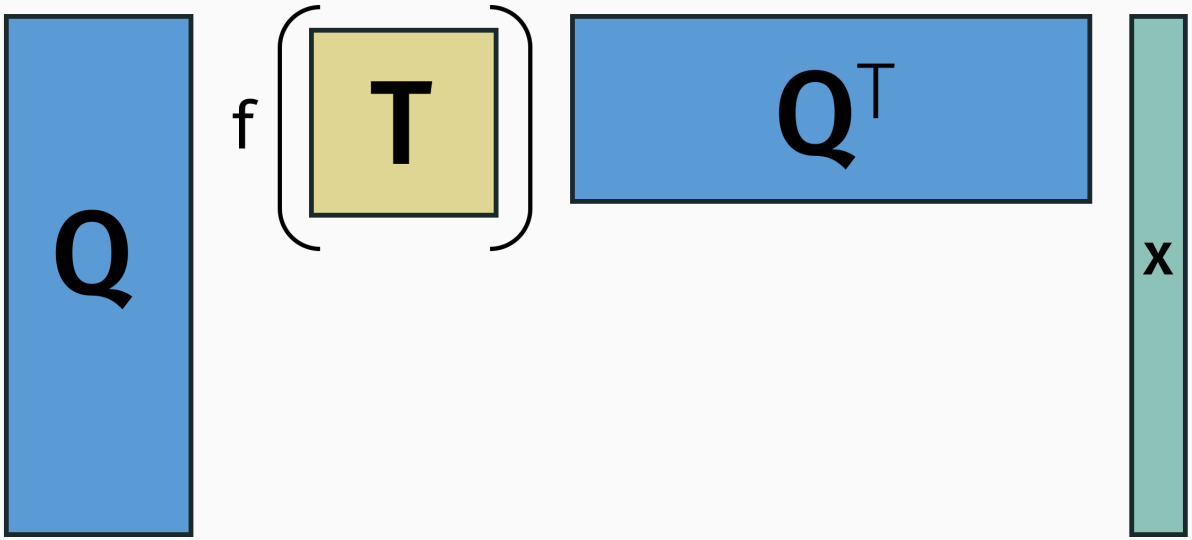
Runtime: $O(k \cdot \text{nnz}(\mathbf{A})) + O(nk^2)$

Compute:



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

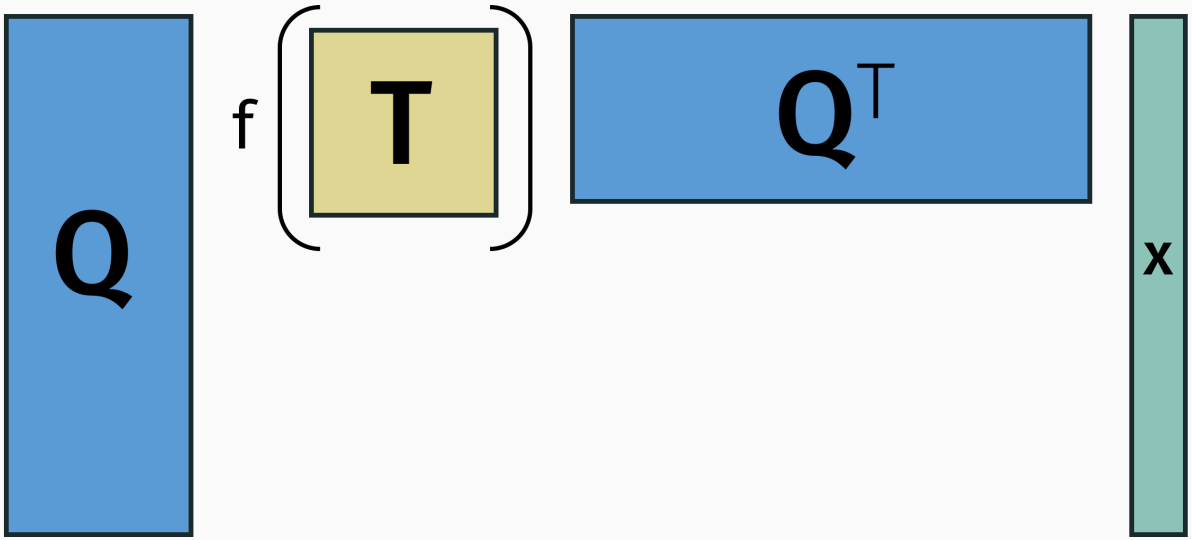
Approximate:



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

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

Approximate:



~~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.

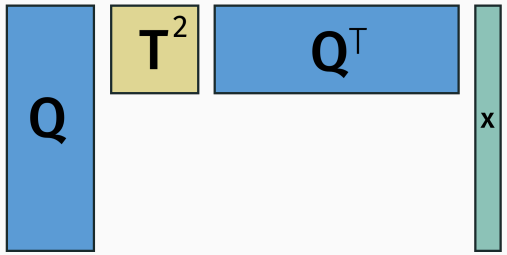
QUICK ANALYSIS OF LANCZOS

Claim: Lanczos applies degree k polynomials exactly.

QUICK ANALYSIS OF LANCZOS

Claim: Lanczos applies degree k polynomials exactly.

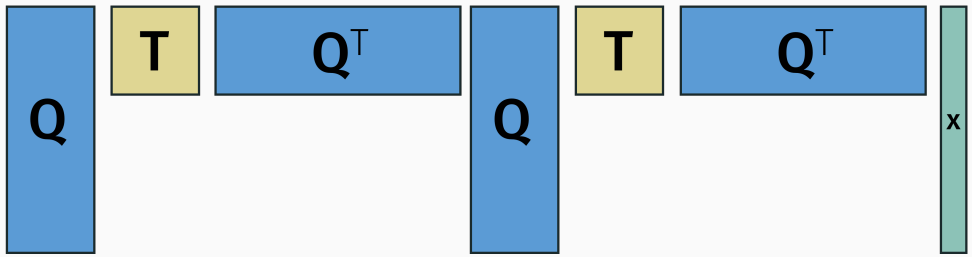
Proof:



QUICK ANALYSIS OF LANCZOS

Claim: Lanczos applies degree k polynomials exactly.

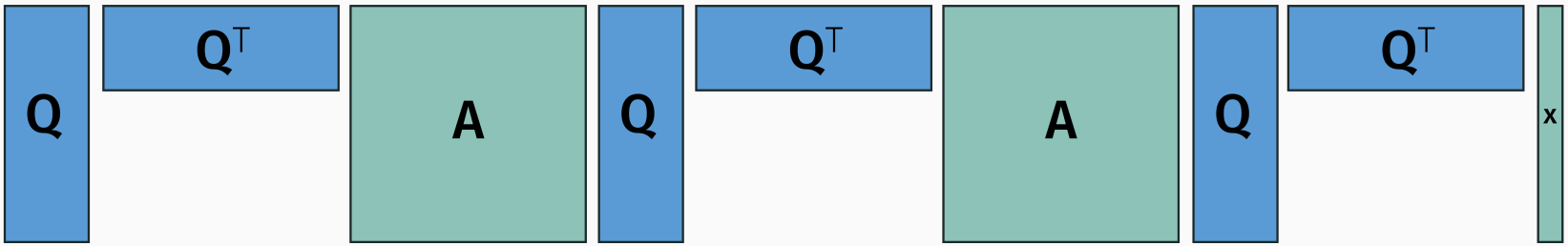
Proof:



QUICK ANALYSIS OF LANCZOS

Claim: Lanczos applies degree k polynomials exactly.

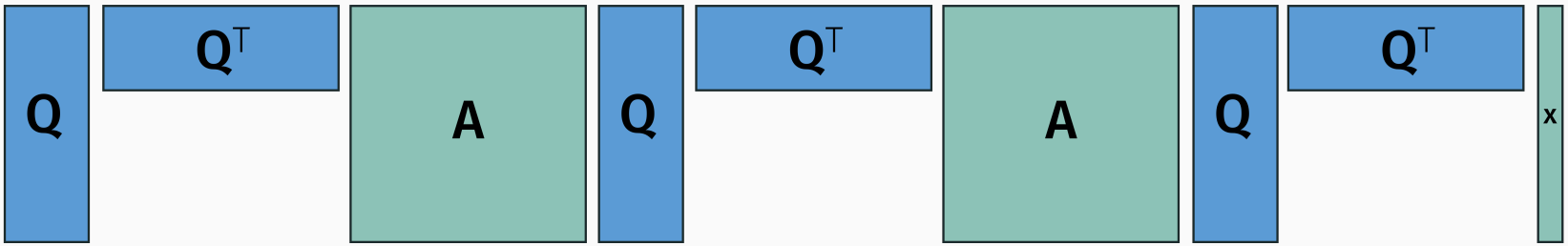
Proof:



QUICK ANALYSIS OF LANCZOS

Claim: Lanczos applies degree k polynomials exactly.

Proof:

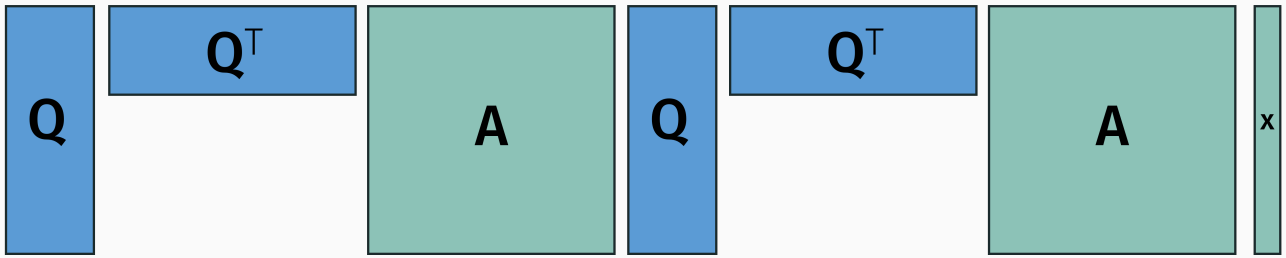


$\mathbf{x}, \mathbf{A}\mathbf{x}, \mathbf{A}^2\mathbf{x}$ all lie in the span of $\mathbf{Q}$.

QUICK ANALYSIS OF LANCZOS

Claim: Lanczos applies degree k polynomials exactly.

Proof:

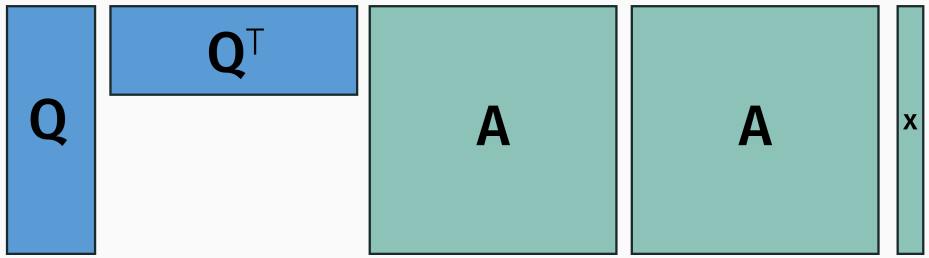


$\mathbf{x}, \mathbf{A}\mathbf{x}, \mathbf{A}^2\mathbf{x}$ all lie in the span of $\mathbf{Q}$.

QUICK ANALYSIS OF LANCZOS

Claim: Lanczos applies degree k polynomials exactly.

Proof:

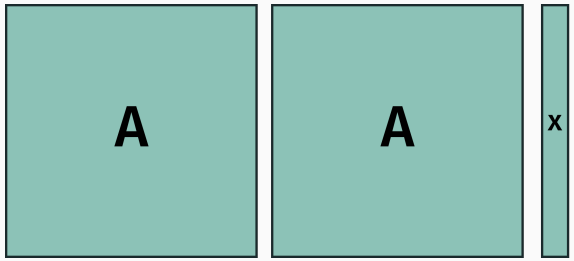


$\mathbf{x}, \mathbf{A}\mathbf{x}, \mathbf{A}^2\mathbf{x}$ all lie in the span of $\mathbf{Q}$.

QUICK ANALYSIS OF LANCZOS

Claim: Lanczos applies degree k polynomials exactly.

Proof:



$\mathbf{x}, \mathbf{A}\mathbf{x}, \mathbf{A}^2\mathbf{x}$ all lie in the span of $\mathbf{Q}$.

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,

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

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,

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

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,

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

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,

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

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,

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

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

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,

$$\begin{aligned}\|f(\mathsf{A})\mathbf{x} - \mathsf{Q}f(\mathsf{T})\mathsf{Q}^T\mathbf{x}\| &\leq \|f(\mathsf{A})\mathbf{x} - p(\mathsf{A})\mathbf{x}\| \\ &\quad + \|p(\mathsf{A})\mathbf{x} - \mathsf{Q}p(\mathsf{T})\mathsf{Q}^T\mathbf{x}\| \\ &\quad + \|\mathsf{Q}p(\mathsf{T})\mathsf{Q}^T\mathbf{x} - \mathsf{Q}f(\mathsf{T})\mathsf{Q}^T\mathbf{x}\| \\ &\leq \delta_k\|\mathbf{x}\| + 0 + \delta_k\|\mathbf{x}\|. \\ &= 2\delta_k \cdot \|\mathbf{x}\|\end{aligned}$$

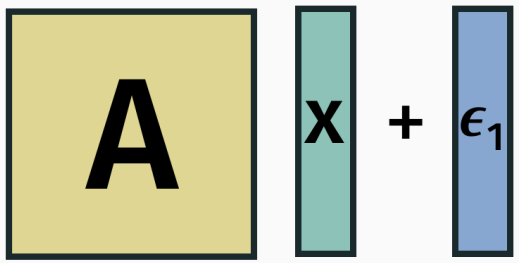
POLYNOMIAL METHODS

WITH NOISE

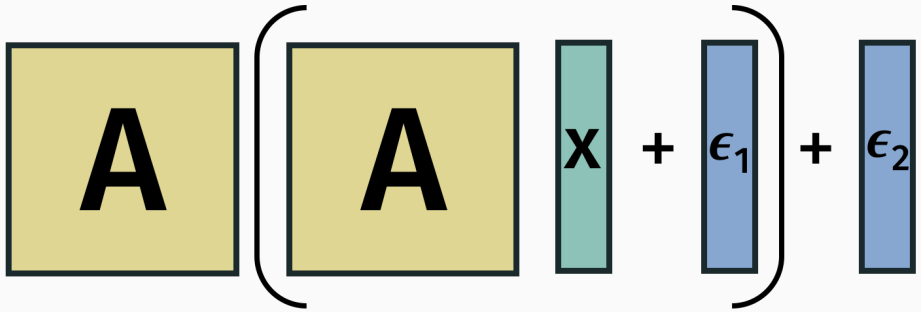
MATRIX FUNCTIONS WITH NOISE

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

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

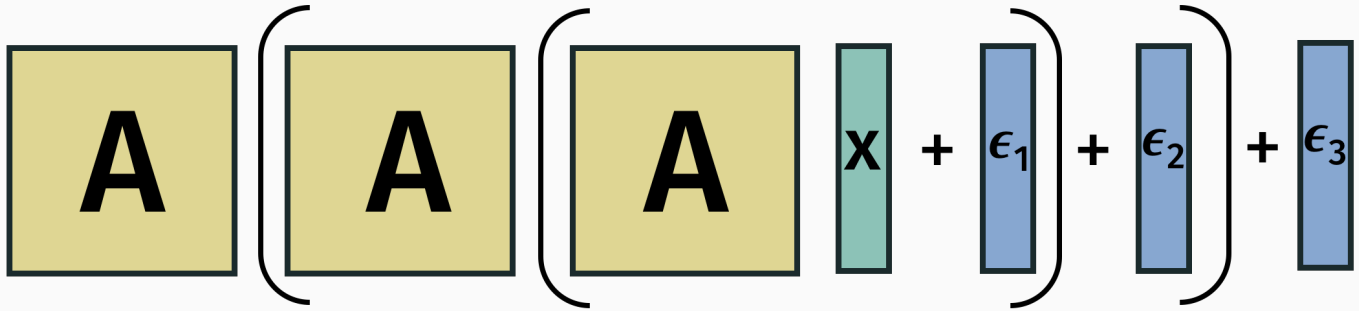


In many applications, we do not multiply by A exactly!

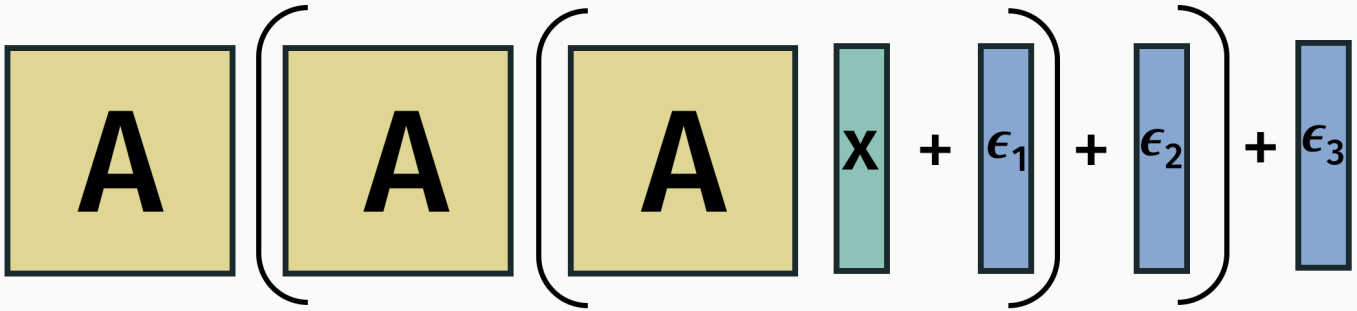


MATRIX FUNCTIONS WITH NOISE

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



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



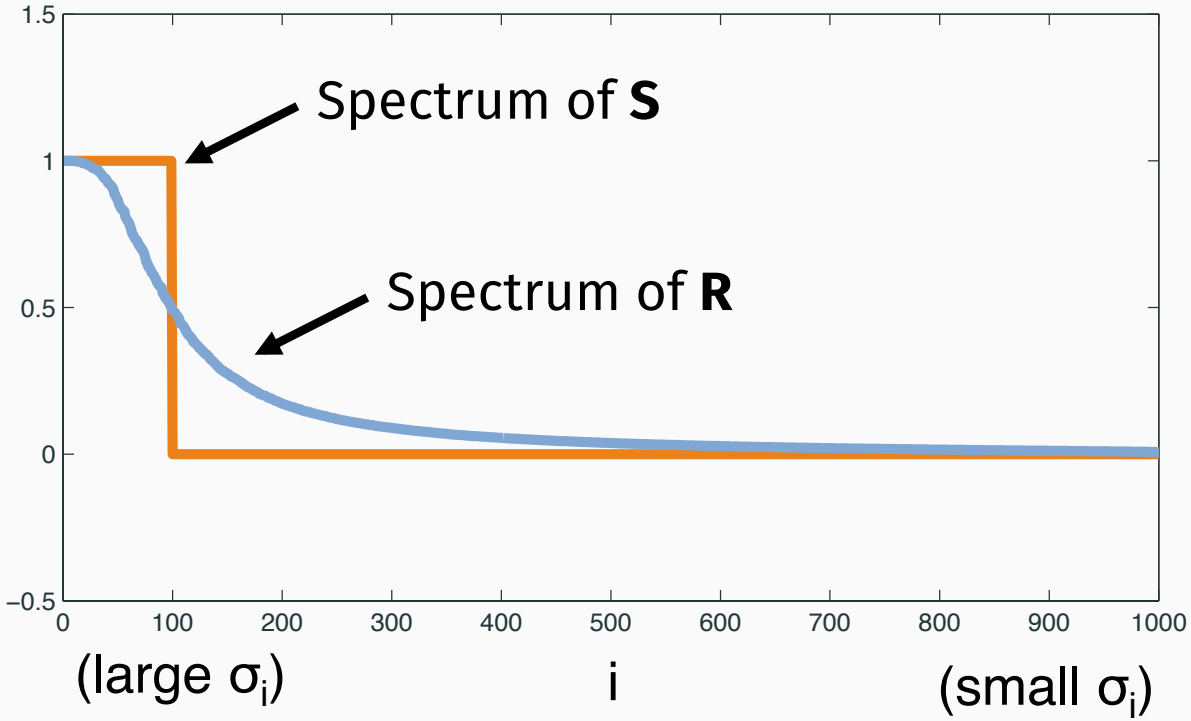
Natural model when Lanczos is combined with scalable randomized or iterative methods.

MATRIX FUNCTIONS WITH NOISE

Powerful paradigm:

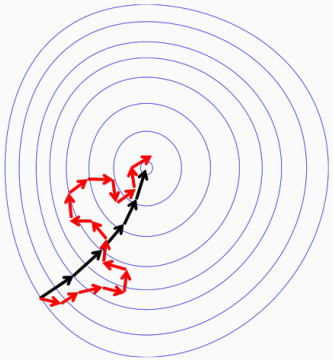
- $\mathsf{A} = \mathsf{B}^{-1}$ for some matrix B.
- Apply B^{-1} to vectors quickly and approximately.

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

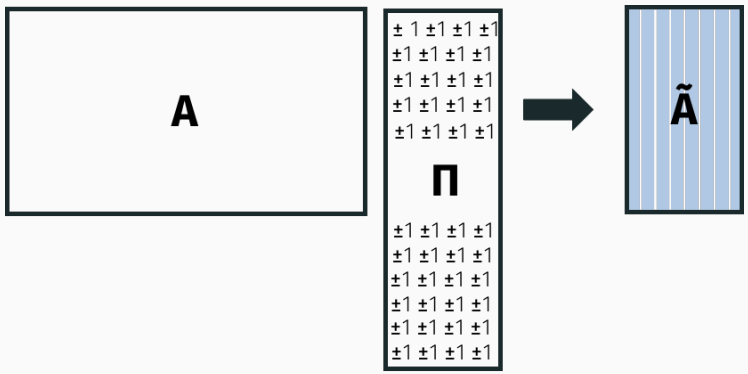


Most of the work is computing $R\mathbf{x}$.

$(A^TA + \lambda I)^{-1}$ can be computed approximately with a variety of highly (often randomized) scalable methods:

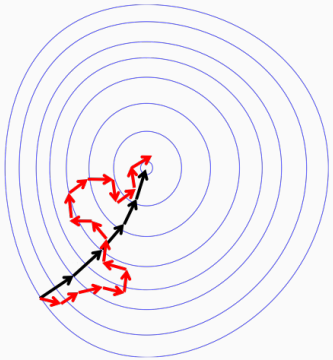


Stochastic Iterative Methods

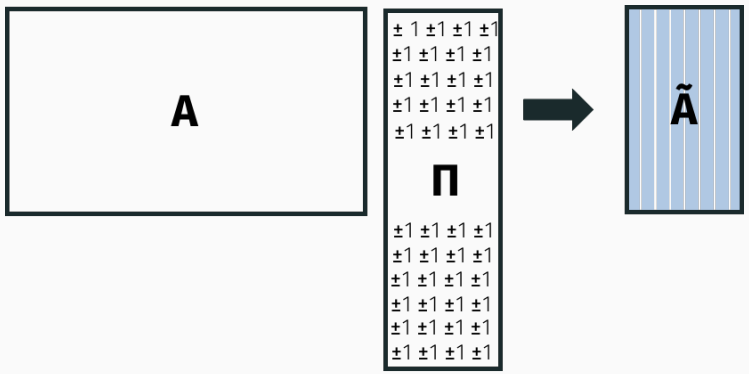


Randomized Sketching

$(\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1}$ can be computed approximately with a variety of highly (often randomized) scalable methods:



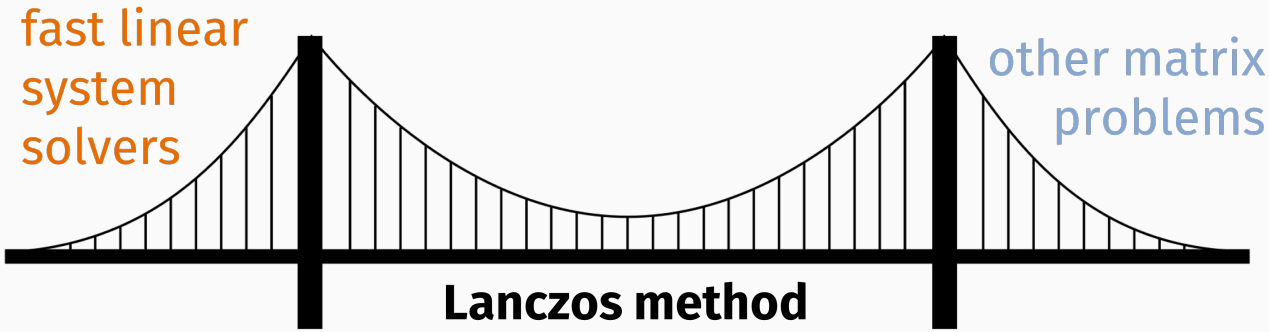
Stochastic Iterative Methods



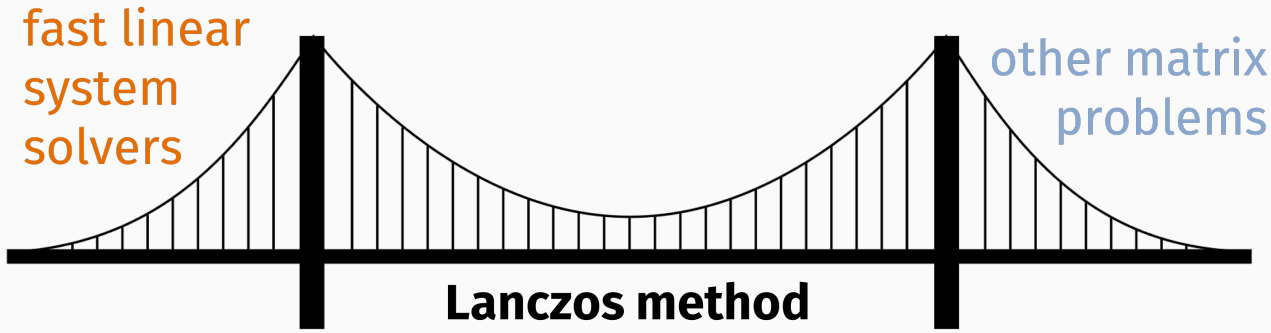
Randomized Sketching

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

LANCZOS AND FAST MATRIX INVERSION



LANCZOS AND FAST MATRIX INVERSION



- 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).

Are matrix function algorithms stable?

We need to understand how the performance of our algorithms change when we replace every matrix-vector multiplication $\mathbf{A}\mathbf{x}$ 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.

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 $\mathbf{A}\mathbf{x}$ is computed approximately. But the Lanczos method (with no modifications) continues to perform well.

Can we explain this phenomena?

STABLE POLYNOMIAL COMPUTATION

Can we apply scalar polynomials in a stable way?

STABLE POLYNOMIAL COMPUTATION

Can we apply scalar polynomials in a stable way?

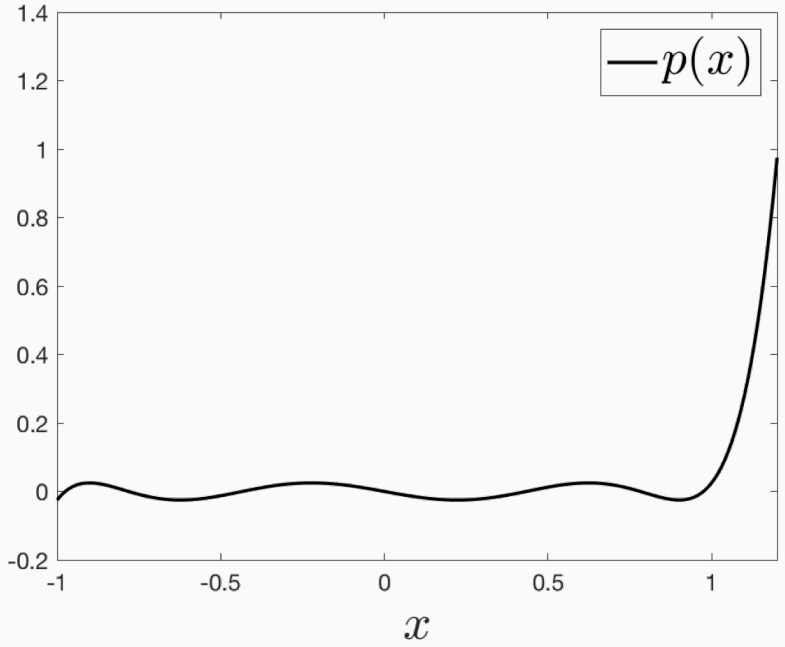
1. Want to compute $p(x) = c_0 + c_1 x + \ldots + c_k x^k$.
2. We do not know x, but we have access to a function `approxMult` that for any input z outputs:

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

STABLE POLYNOMIAL COMPUTATION

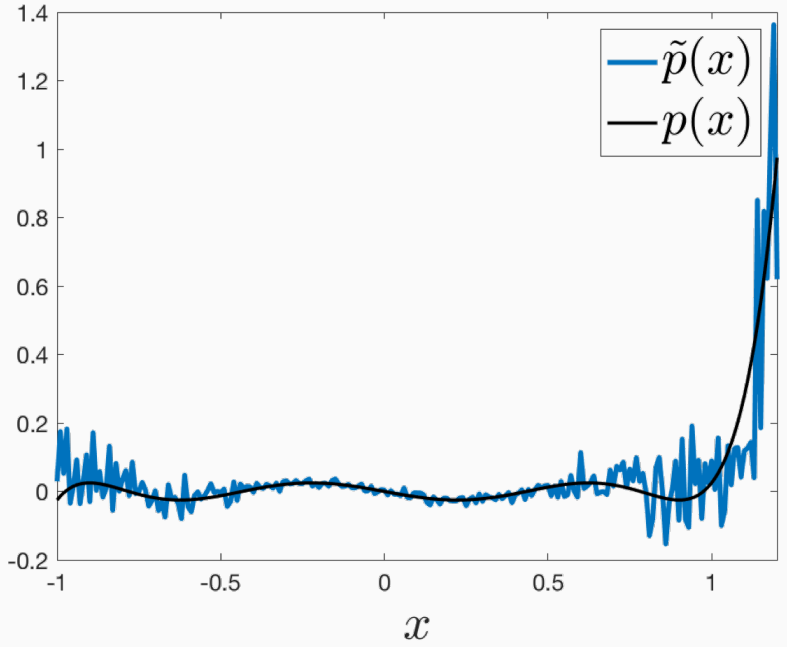
Goal: Compute $p(x) = 64x^7 - 112x^5 + 56x^3 - 7x$.

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



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

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



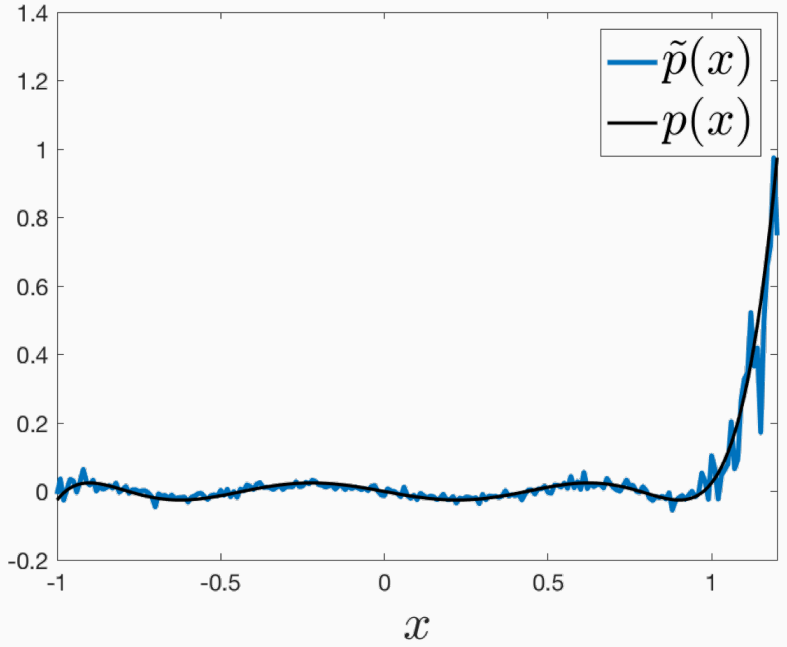
Directly compute and sum monomials.

$x^i = \texttt{approxMult}(\texttt{approxMult}(\ldots\texttt{approxMult}(1)\ldots))$

STABLE POLYNOMIAL COMPUTATION

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

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



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

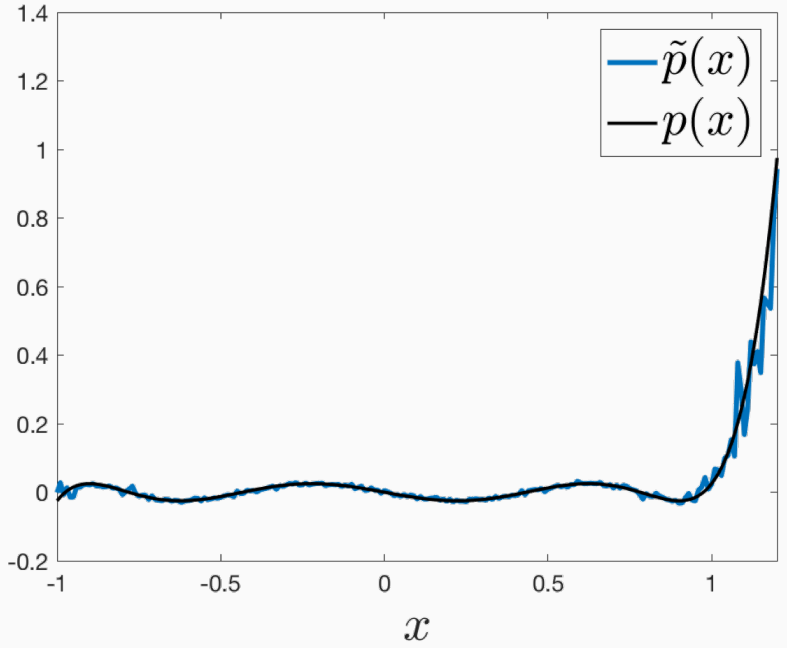
$$t_1 = \texttt{approxMult}(1) - .98$$

$$t_2 = \texttt{approxMult}(t_1) - .78 \cdot t_1$$

$$\ldots$$

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

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



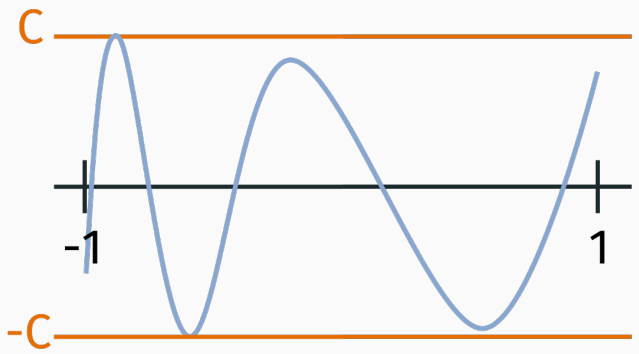
It's possible to do even better.

STABLE POLYNOMIAL COMPUTATION

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

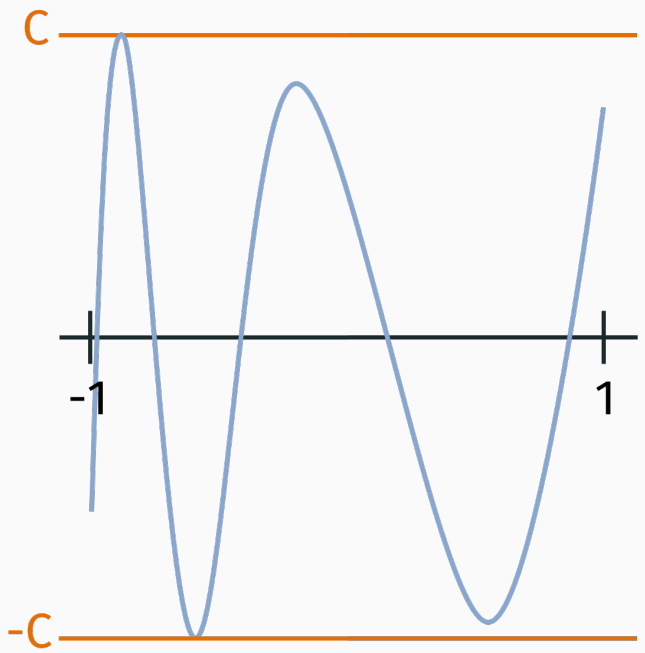
STABLE POLYNOMIAL COMPUTATION

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



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.

Claim

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

When translating to matrices:

- Runtime of linear system solvers scales with $\log(1/\epsilon)$

Claim

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

When translating to matrices:

- Runtime of linear system solvers scales with $\log(1/\epsilon)$
- Set $\epsilon = \epsilon'/Ck^3$ to get error ϵ'.

Claim

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

When translating to matrices:

- Runtime of linear system solvers scales with $\log(1/\epsilon)$
- Set $\epsilon = \epsilon'/Ck^3$ to get error ϵ'.
- Runtime overhead is just $O(\log(kC))$.

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:

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

Compute monomials:

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

Since $|x| \leq 1$, error on x^i bounded by $\epsilon_1 + \epsilon_2 + \ldots + \epsilon_3 \leq \epsilon i$.

Compute monomials:

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

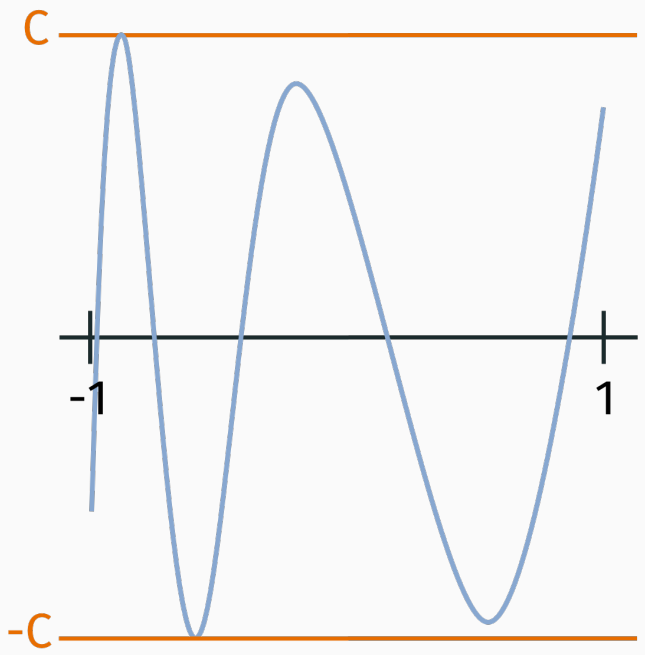
Since $|x| \leq 1$, error on x^i bounded by $\epsilon_1 + \epsilon_2 + \ldots + \epsilon_3 \leq \epsilon i$.

We can then compute $p(x) = c_0 + c_1 x + \ldots c_k x^k$ up to error:

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

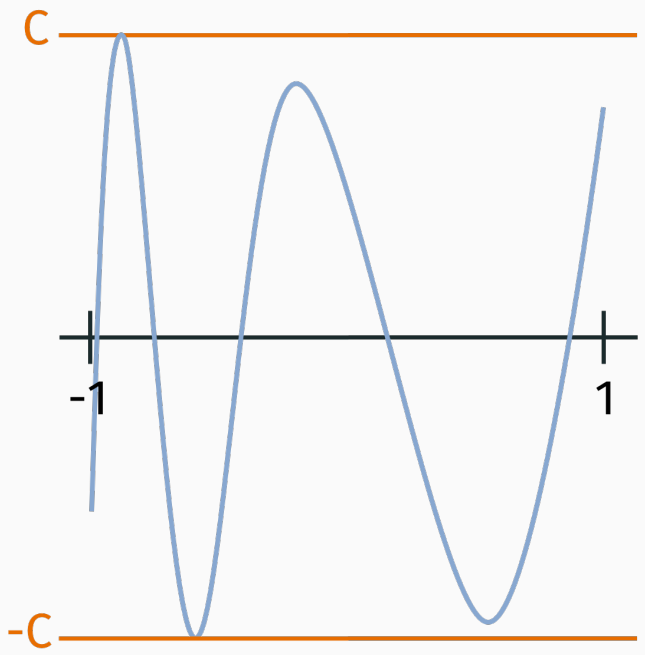
FIRST ATTEMPT

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



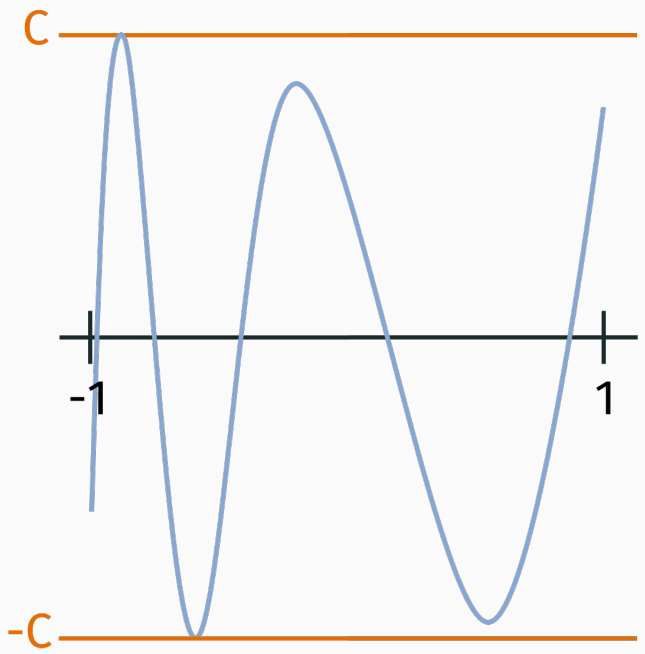
FIRST ATTEMPT

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



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

$\sum_{i=1}^{k} |c_k|$ 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.

$$\log\left(\frac{1}{\epsilon/2^k}\right) = k + \log(1/\epsilon)$$

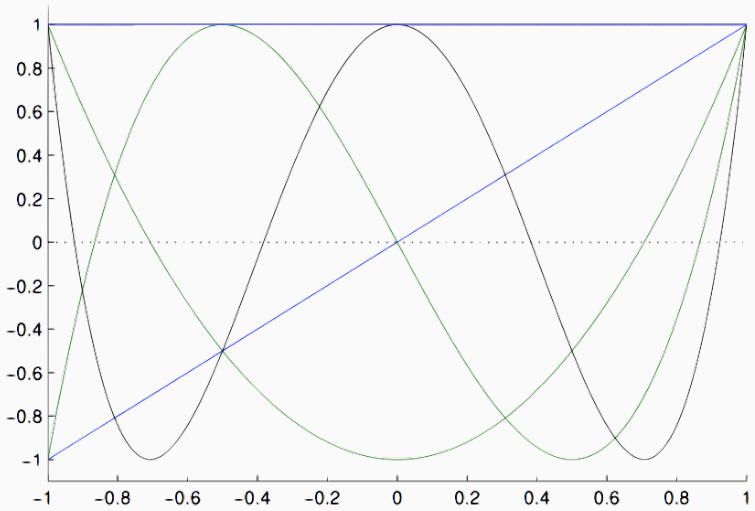
“BAD” POLYNOMIALS

What are those 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.

$$
\begin{aligned}
\mathrm{T}_0(x) &= 1 \\
\mathrm{T}_1(x) &= x \\
\mathrm{T}_2(x) &= 2x^2 - 1 \\
\mathrm{T}_3(x) &= 4x^3 - 3x \\
\mathrm{T}_4(x) &= 8x^4 - 8x^2 + 1 \\
\mathrm{T}_5(x) &= 16x^5 - 20x^3 + 5x \\
\mathrm{T}_6(x) &= 32x^6 - 48x^4 + 18x^2 - 1 \\
\mathrm{T}_7(x) &= 64x^7 - 112x^5 + 56x^3 - 7x \\
\mathrm{T}_8(x) &= 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1 \\
\mathrm{T}_9(x) &= 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x \\
\mathrm{T}_{10}(x) &= 512x^{10} - 1280x^8 + 1120x^6 - 400x^4 + 50x^2 - 1 \\
\mathrm{T}_{11}(x) &= 1024x^{11} - 2816x^9 + 2816x^7 - 1232x^5 + 220x^3 - 11x
\end{aligned}
$$

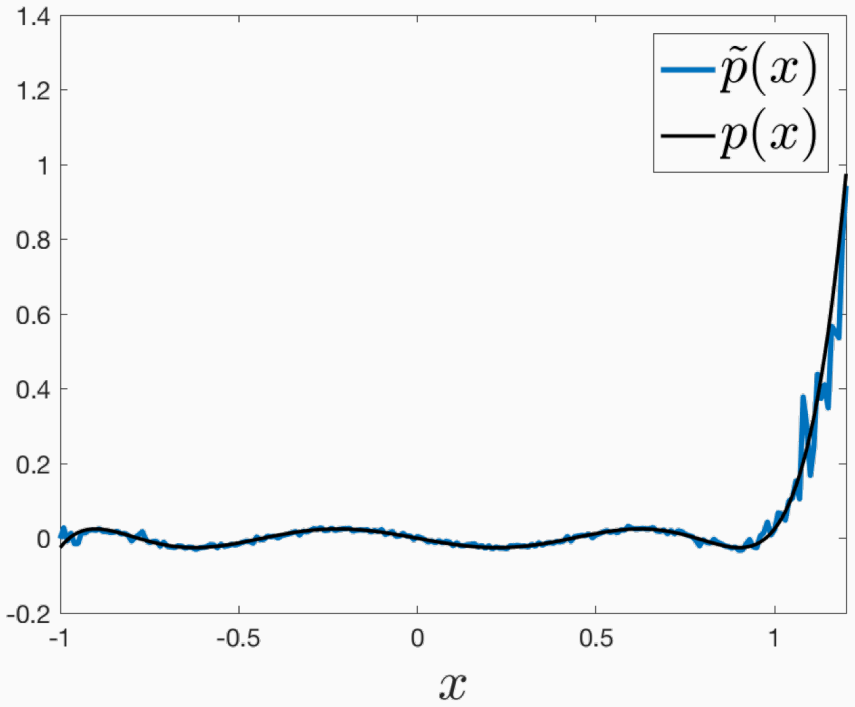
What are those polynomials?

Chebyshev polynomials of the first kind.

$$
\begin{aligned}
\mathrm{T}_0(x) &= 1 \\
\mathrm{T}_1(x) &= x \\
\mathrm{T}_2(x) &= 2x^2 - 1 \\
\mathrm{T}_3(x) &= 4x^3 - 3x \\
\mathrm{T}_4(x) &= 8x^4 - 8x^2 + 1 \\
\mathrm{T}_5(x) &= 16x^5 - 20x^3 + 5x \\
\mathrm{T}_6(x) &= 32x^6 - 48x^4 + 18x^2 - 1 \\
\mathrm{T}_7(x) &= 64x^7 - 112x^5 + 56x^3 - 7x \\
\mathrm{T}_8(x) &= 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1 \\
\mathrm{T}_9(x) &= 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x \\
\mathrm{T}_{10}(x) &= 512x^{10} - 1280x^8 + 1120x^6 - 400x^4 + 50x^2 - 1 \\
\mathrm{T}_{11}(x) &= 1024x^{11} - 2816x^9 + 2816x^7 - 1232x^5 + 220x^3 - 11x
\end{aligned}
$$

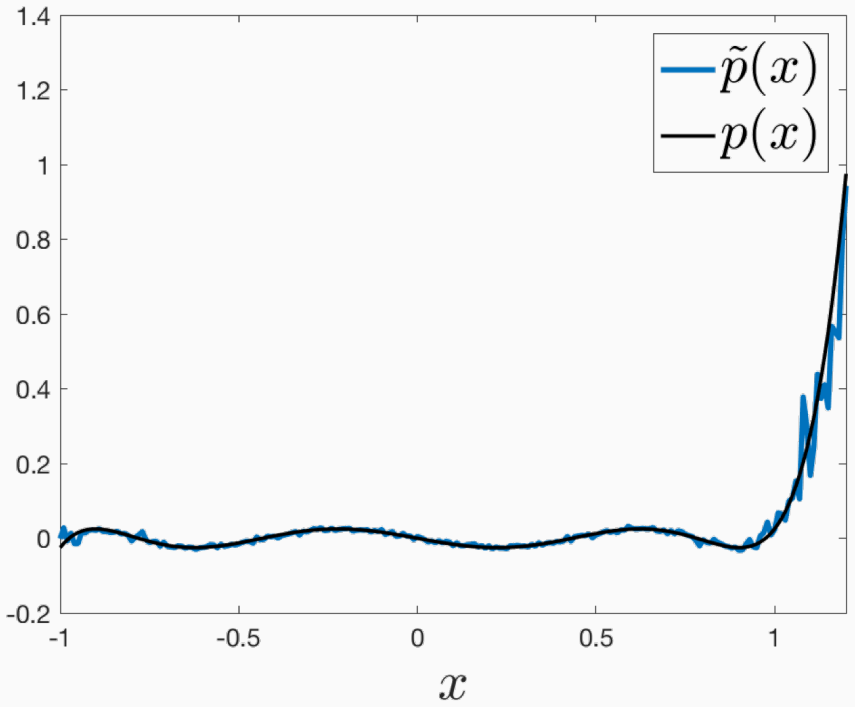
We can apply these in a stable way, using their recurrence!

“GOOD” POLYNOMIALS?



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

"GOOD" POLYNOMIALS?



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

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

KEY OBSERVATION

Chebyshev polynomails are the only hard case.

KEY OBSERVATION

Chebyshev polynomails 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

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

where every $|c_i| \leq 2C$.

Chebyshev polynomails 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

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

where every $|c_i| \leq 2C$.

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

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

Same arguments extends from scalar polynomials to matrix polynomials.

STABILITY OF LANCZOS

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.

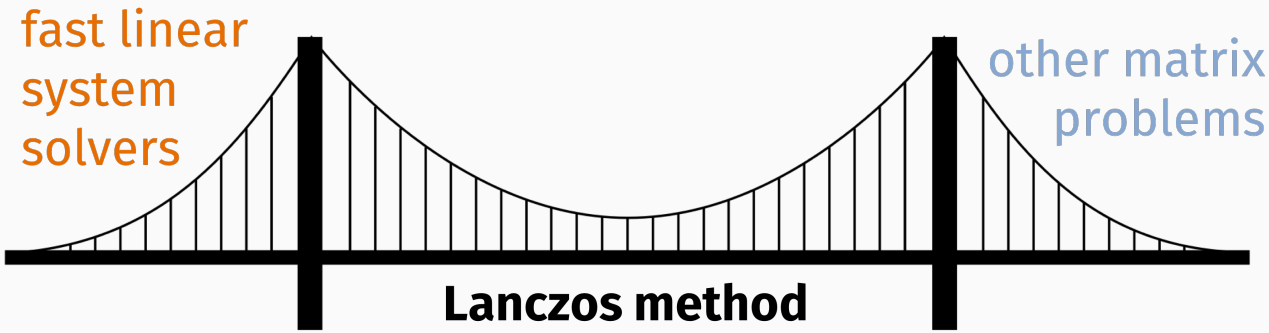
Step 1: Lanczos stably applies Chebyshev polynomials (building on results of Paige [‘71, ‘76, ‘80]).

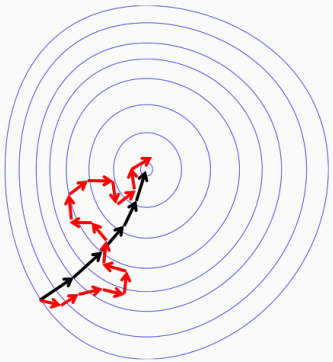
Step 2: By linearity, Lanczos stably applies polynomials bounded by C.

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

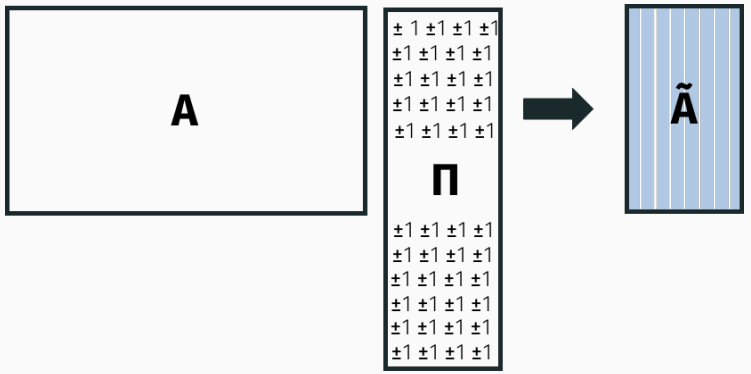
Use Lanczos without fear on bounded functions!

STABILITY OF LANCZOS





Stochastic Iterative Methods



Randomized Sketching

Paper gives 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 $x \in [\lambda_{\min}(\mathsf{A}), \lambda_{\max}(\mathsf{A})]$, 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

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

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 $x \in [\lambda_{\min}(\mathsf{A}), \lambda_{\max}(\mathsf{A})]$, 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

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

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

- Compare to $\|f(\mathsf{A})\mathbf{x} - \mathbf{y}\| \leq 2 \cdot \delta_k \cdot \|\mathbf{x}\|$ 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

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

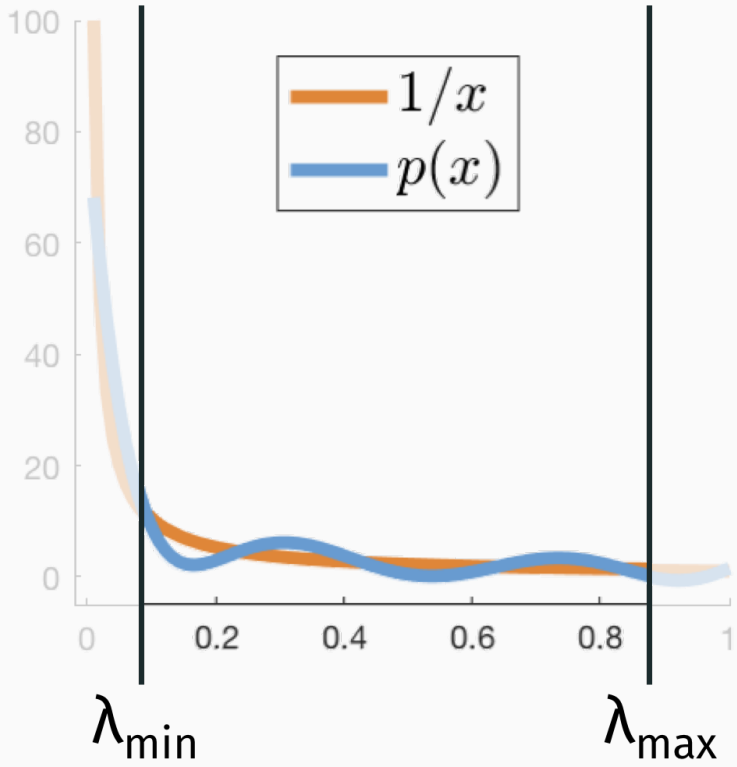
where δ_k is the error of the best degree k uniform approximation to f.

- 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).

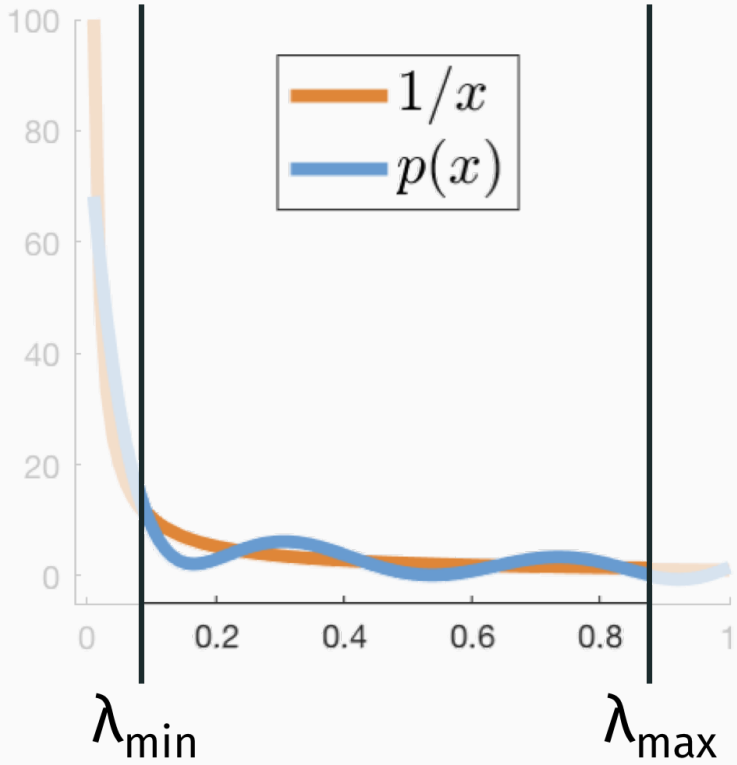
NEGATIVE RESULT FOR

LINEAR SYSTEMS

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

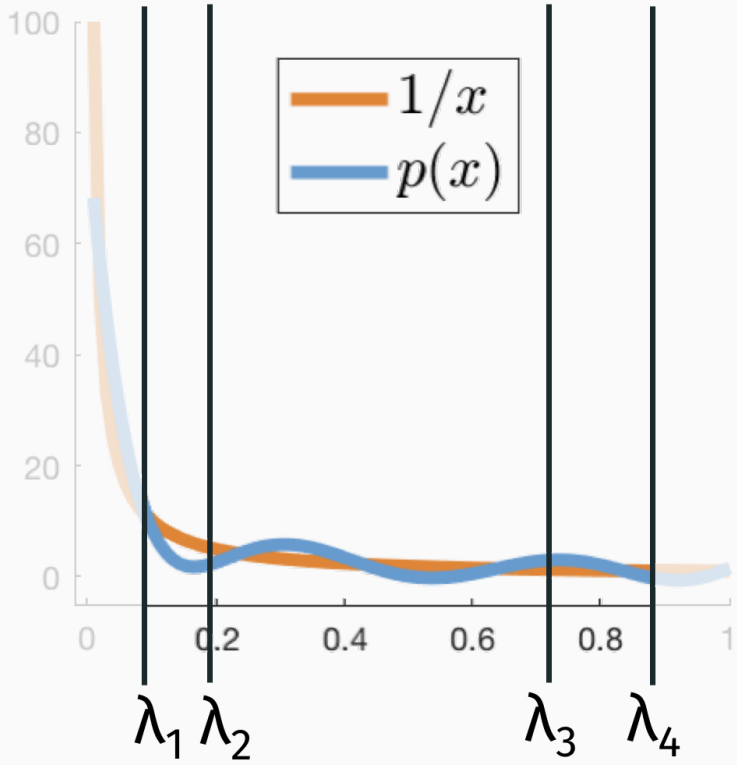


We proved earlier that Lanczos always matches the best uniform approximating polynomial for $f(x)$:



For linear systems it actually does better than that.

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

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

LANCZOS FOR LINEAR SYSTEMS

- 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 $\mathbf{A}$ 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 $\mathbf{A}$ only has n eigenvalues.

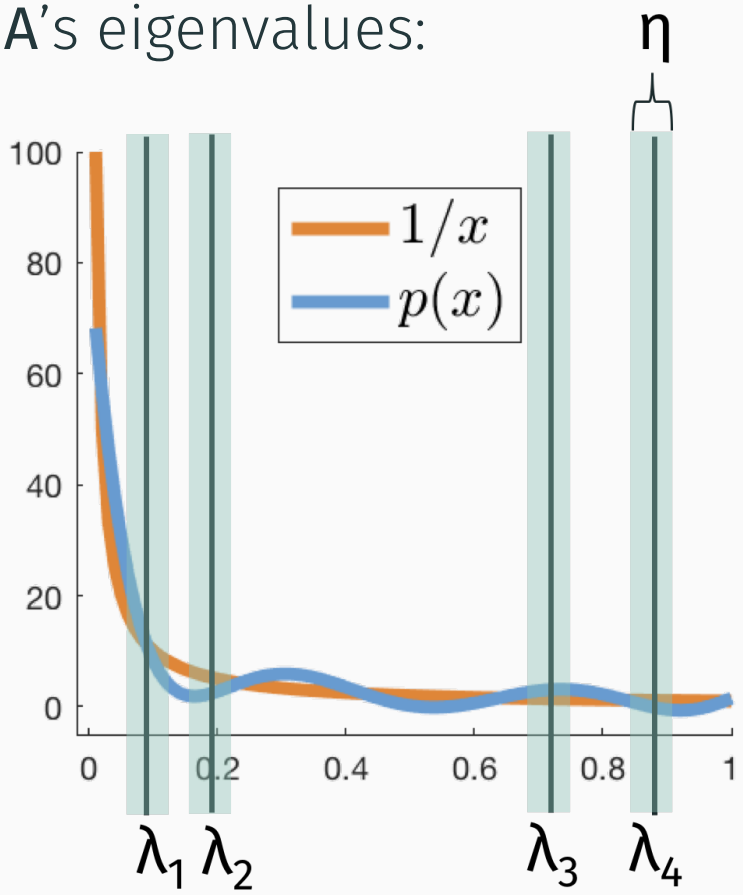
Claim: On exact arithmetic computers, linear systems can be solved in $O(\text{nnz}(\mathbf{A}) \cdot n)$ time (i.e. n iterations of Lanczos)

- 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 $\mathbf{A}$ 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?

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.)

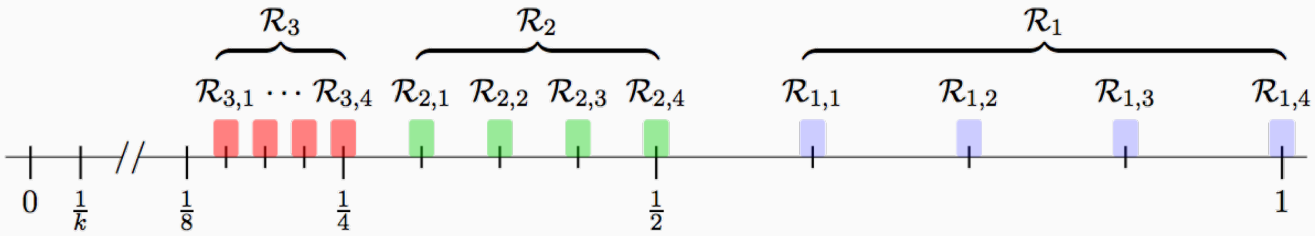
For any n, there is a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ with condition number $\lambda_{\max} / \lambda_{\min}$ such that no k degree polynomial satisfies Greenbaum's condition with error $\leq 1/3$ for all

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

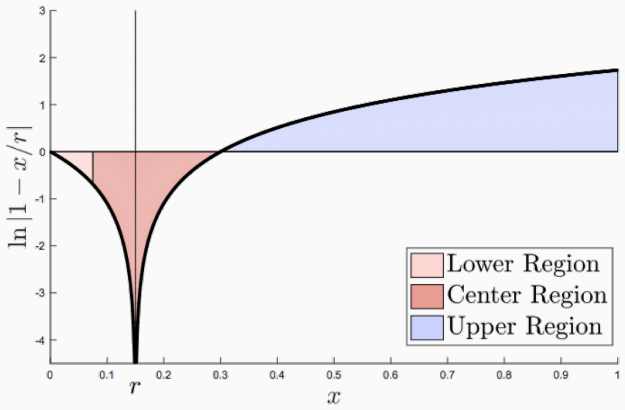
even when $\eta \leq \frac{1}{2^{n/\log \kappa}}$.

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

Construction: Eigenvalues roughly uniform on geometric scale.



Proof: Discretization of new potential function proof for Markov brother's inequality.



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 extend to an actual runtime lower bound?
- How about for a more general class of algorithms? Any method accessing $\mathbf{A}$ only through noisy matrix-vector products?

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 extend to an actual runtime lower bound?
- How about for a more general class of algorithms? Any method accessing $\mathbf{A}$ only through noisy matrix-vector products?

Are $O(\text{nnz}(\mathbf{A}) \cdot n)$ time algorithms possible for linear systems?