PRINCIPAL COMPONENT REGRESSION WITHOUT PRINCIPAL COMPONENT ANALYSIS

Roy Frostig [1], Cameron Musco [2], Christopher Musco [2], Aaron Sidford [3]

[1]Stanford, [2]MIT, [3]Microsoft Reseach

RESOURCES

Paper, slides, and template code available at

chrismusco.com

OUR WORK

Simple, robust algorithms for principal component regression.

PRINCIPAL COMPONENT REGRESSION

Principal Component Regression (PCR) =

Principal Component Analysis (unsupervised)

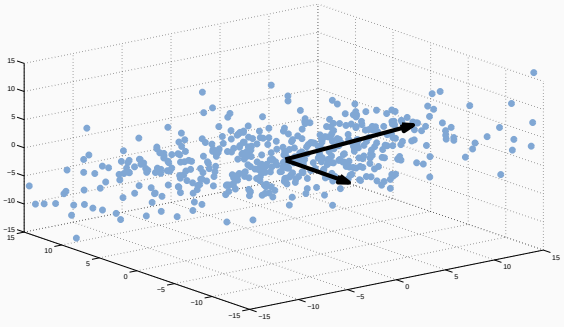
+

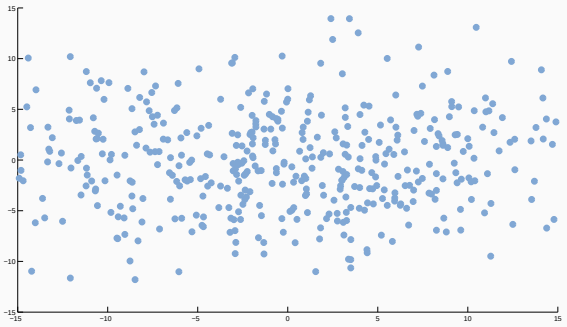
Linear Regression (supervised)

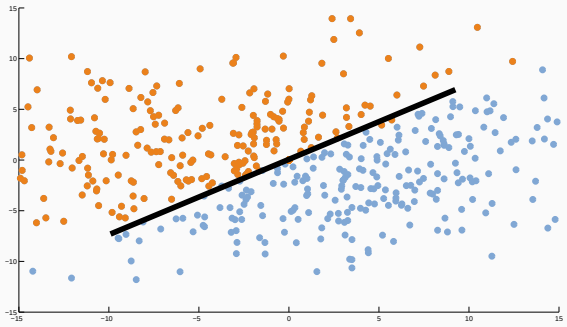
OUR APPROACH: SKIP THE DIMENSIONALITY REDUCTION

PCA

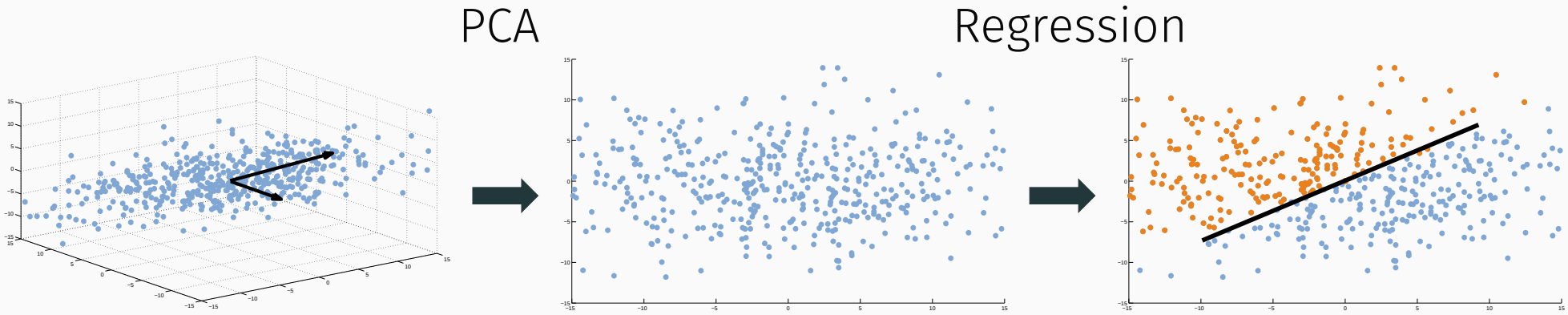
Regression







OUR APPROACH: SKIP THE DIMENSIONALITY REDUCTION

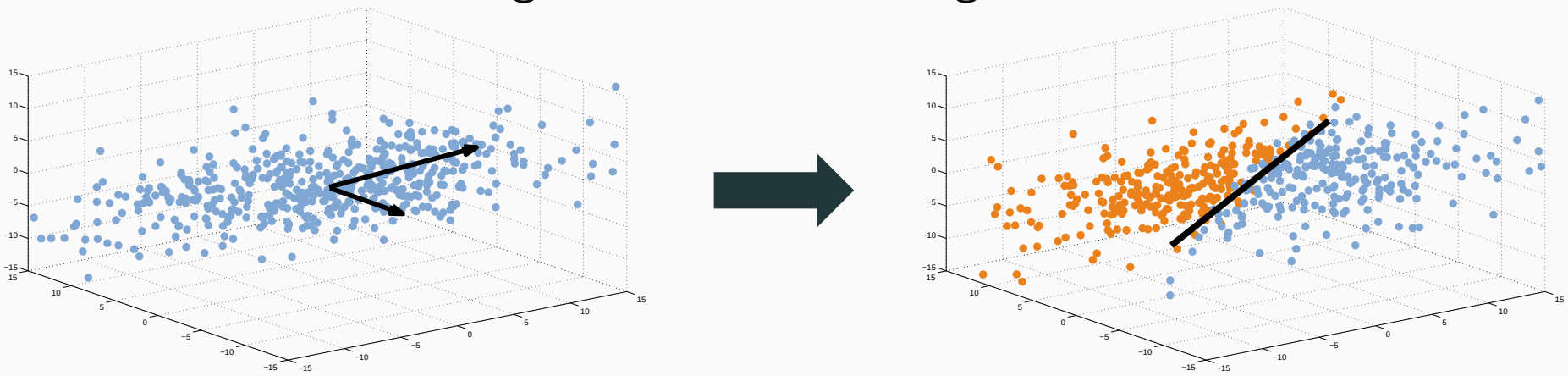


Regression is cheap (fast iterative or stochastic methods).

PCA is a major computational bottleneck.

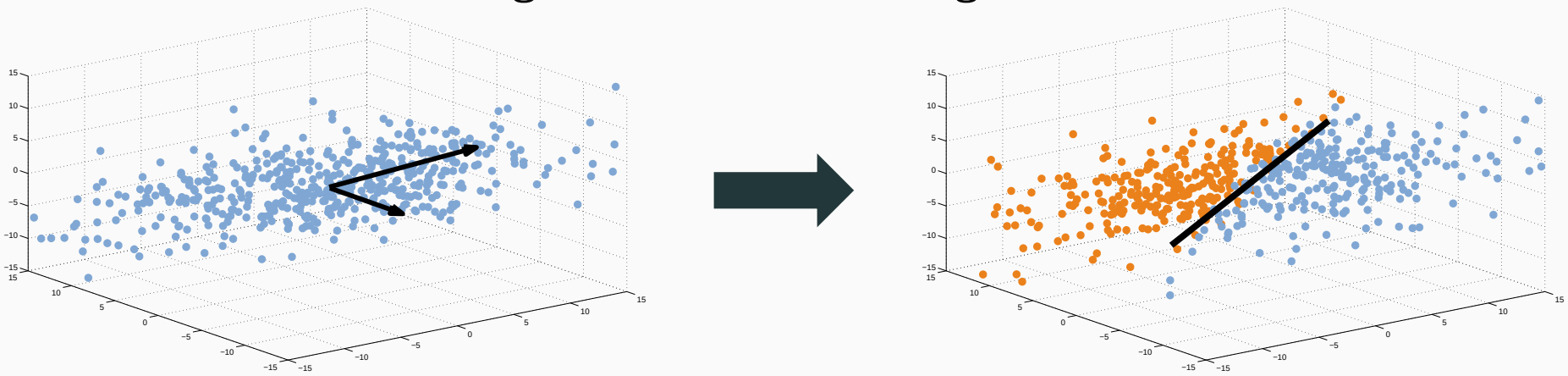
OUR APPROACH: SKIP THE DIMENSIONALITY REDUCTION

Single-shot iterative algorithm



OUR APPROACH: SKIP THE DIMENSIONALITY REDUCTION

Single-shot iterative algorithm



Final algorithm just uses a few applications of any fast, black-box regression routine.

Standard Regression:

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

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

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

Standard Regression:

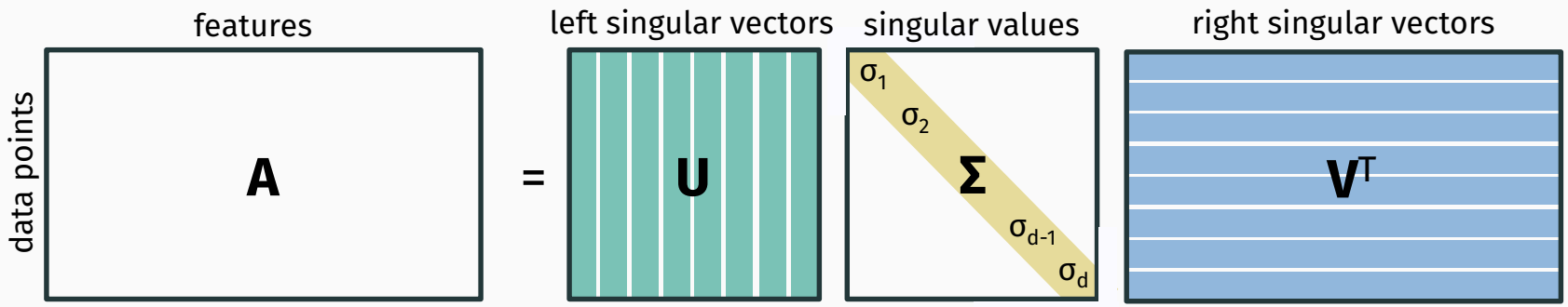
Given: $\mathbf{A}$, $\mathbf{b}$

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

Principal Component Regression:

Given: $\mathbf{A}$, $\mathbf{b}$, λ

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



Standard Regression:

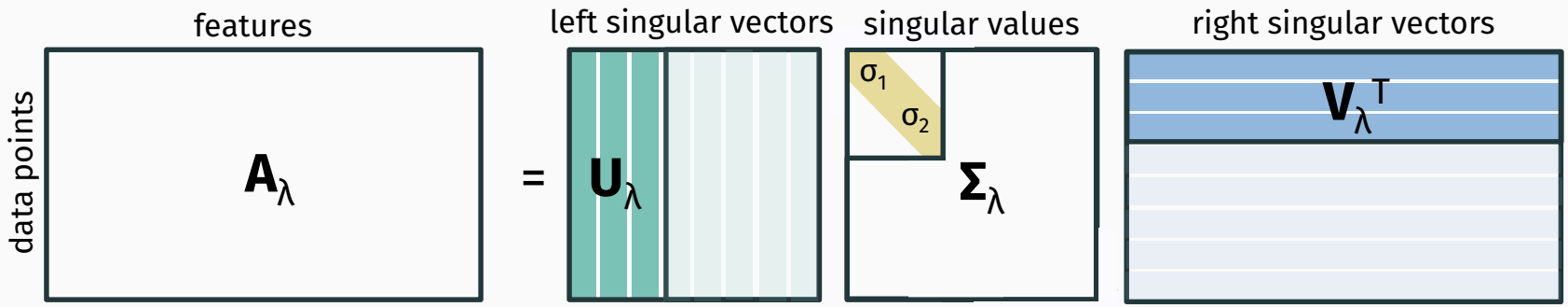
Given: $\mathbf{A}$, $\mathbf{b}$

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

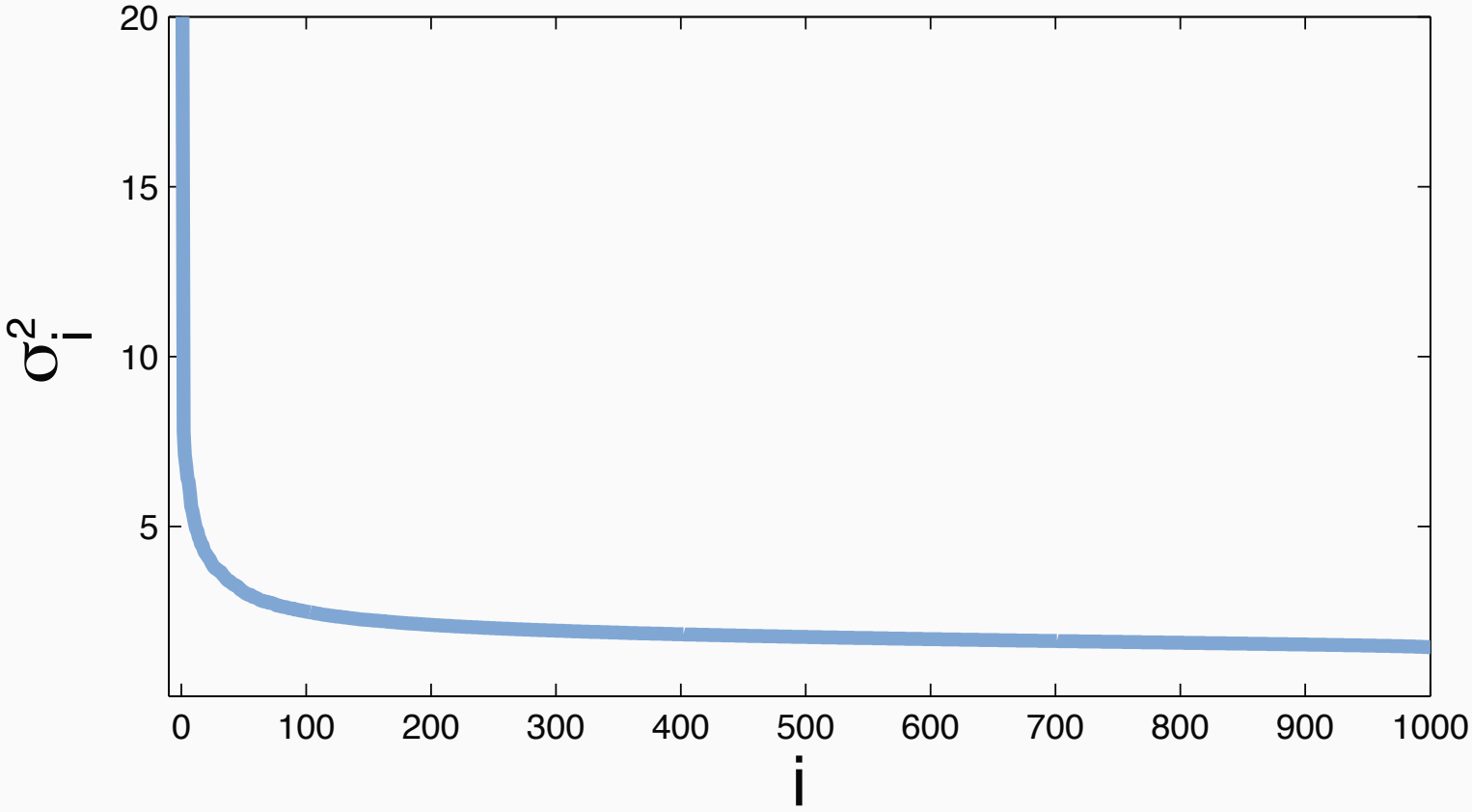
Principal Component Regression:

Given: $\mathbf{A}$, $\mathbf{b}$, λ

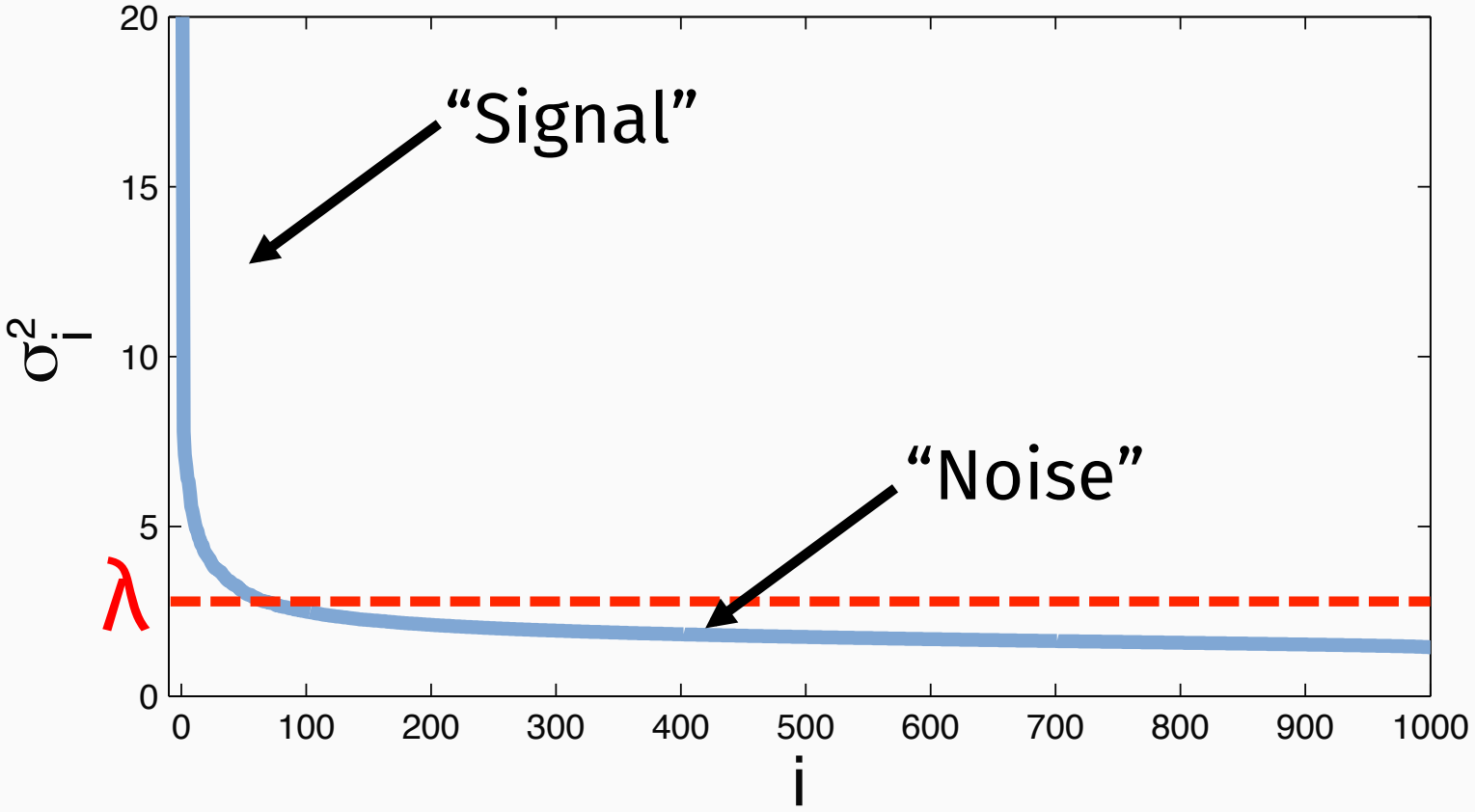
Solve: $\mathbf{x}^* = \arg\min_{\mathbf{x}} \|\mathbf{A}_\lambda\mathbf{x} - \mathbf{b}\|^2$



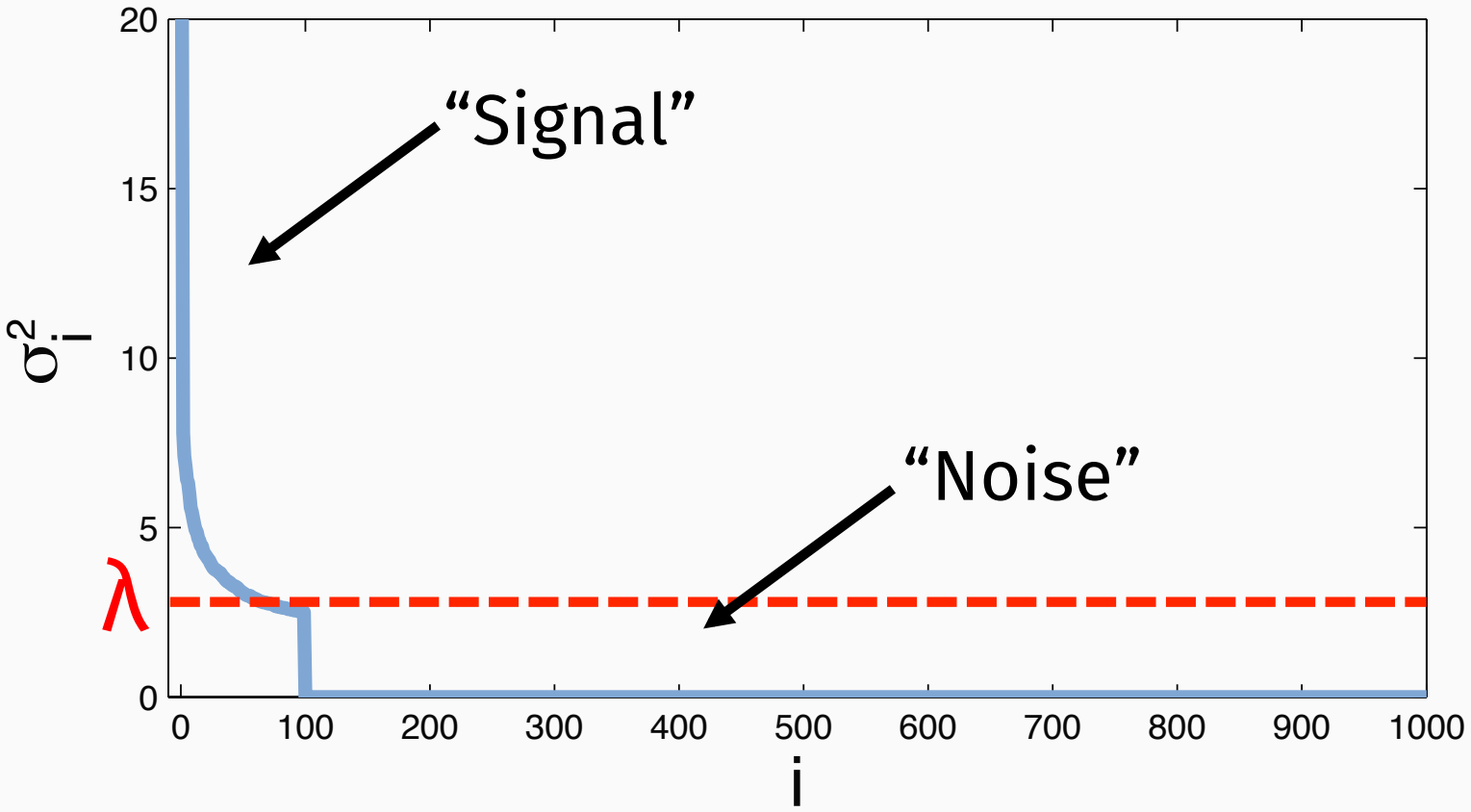
Singular values of **A**



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

What's the computational cost?

Cost of computing A_λ (PCA):

$$\approx O(\mathrm{nnz}(\mathbf{A})k + dk^2).$$

Cost of computing A_λ (PCA):

$$\approx O(\mathrm{nnz}(\mathbf{A})k + dk^2).$$

k is the number of principal components with value $> \lambda$.

NAIVE PCR RUNTIME

Cost of computing $\mathbf{A}_\lambda$ (PCA):

$$\approx O(\text{nnz}(\mathbf{A})k + dk^2).$$

k is the number of principal components with value $> \lambda$.

Cost of evaluating $\mathbf{x} = (\mathbf{A}_\lambda^T \mathbf{A}_\lambda)^{-1} \mathbf{A}_\lambda^T \mathbf{b}$ (regression):

$$\approx O(\text{nnz}(\mathbf{A}) \cdot \sqrt{\kappa}).$$

Cost of computing $\mathbf{A}_\lambda$ (PCA):

$$\approx O(\text{nnz}(\mathbf{A})k + dk^2).$$

k is the number of principal components with value $> \lambda$.

Cost of evaluating $\mathbf{x} = (\mathbf{A}_\lambda^T \mathbf{A}_\lambda)^{-1} \mathbf{A}_\lambda^T \mathbf{b}$ (regression):

$$\approx O(\text{nnz}(\mathbf{A}) \cdot \sqrt{\kappa}).$$

For PCR, k is large, κ is small ($\mathbf{A}_\lambda$ is well conditioned).

Goal: Remove bottleneck dependence on k

Reason for hope?

Optimistic observation: PCA computes too much information.

REASON FOR HOPE?

Optimistic observation: PCA computes too much information.

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

REASON FOR HOPE?

Optimistic observation: PCA computes too much information.

$$x^* = (A_\lambda^T A_\lambda)^{-1} A_\lambda^T b = (A^T A)^{-1} \, A_\lambda^T b$$

REASON FOR HOPE?

Optimistic observation: PCA computes too much information.

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

Don't need to compute $\mathbf{A}_\lambda$ (which incurs a k dependence) as long as we can apply it to a *single vector* efficiently.

REASON FOR HOPE?

It's very often more efficient to *apply* a matrix function once than compute the matrix function explicitly.

REASON FOR HOPE?

It's very often more efficient to *apply* a matrix function once than compute the matrix function explicitly.

- $(\mathbf{A}^T\mathbf{A})\mathbf{x}$, $(\mathbf{A}^T\mathbf{A})^2\mathbf{x}$, or $(\mathbf{A}^T\mathbf{A})^3\mathbf{x}$

REASON FOR HOPE?

It's very often more efficient to *apply* a matrix function once than compute the matrix function explicitly.

- $(A^TA)x$, $(A^TA)^2x$, or $(A^TA)^3x$
- $A^{-1}x$

REASON FOR HOPE?

It's very often more efficient to *apply* a matrix function once than compute the matrix function explicitly.

- $(\mathbf{A}^T\mathbf{A})\mathbf{x}$, $(\mathbf{A}^T\mathbf{A})^2\mathbf{x}$, or $(\mathbf{A}^T\mathbf{A})^3\mathbf{x}$
- $\mathbf{A}^{-1}\mathbf{x}$
- $\exp(\mathbf{A})$ . . . many more

REASON FOR HOPE?

It's very often more efficient to *apply* a matrix function once than compute the matrix function explicitly.

- $(\mathbf{A}^T\mathbf{A})\mathbf{x}$, $(\mathbf{A}^T\mathbf{A})^2\mathbf{x}$, or $(\mathbf{A}^T\mathbf{A})^3\mathbf{x}$
- $\mathbf{A}^{-1}\mathbf{x}$
- $\exp(\mathbf{A})$ . . . many more

Why not $\mathbf{A}_\lambda$?

Theorem (Main Result)

There's an algorithm that approximately applies $\mathbf{A}_\lambda^T$ *to any vector* $\mathbf{b}$ *using* $\approx \log(1/\epsilon)$ *well conditioned linear system solutions.*

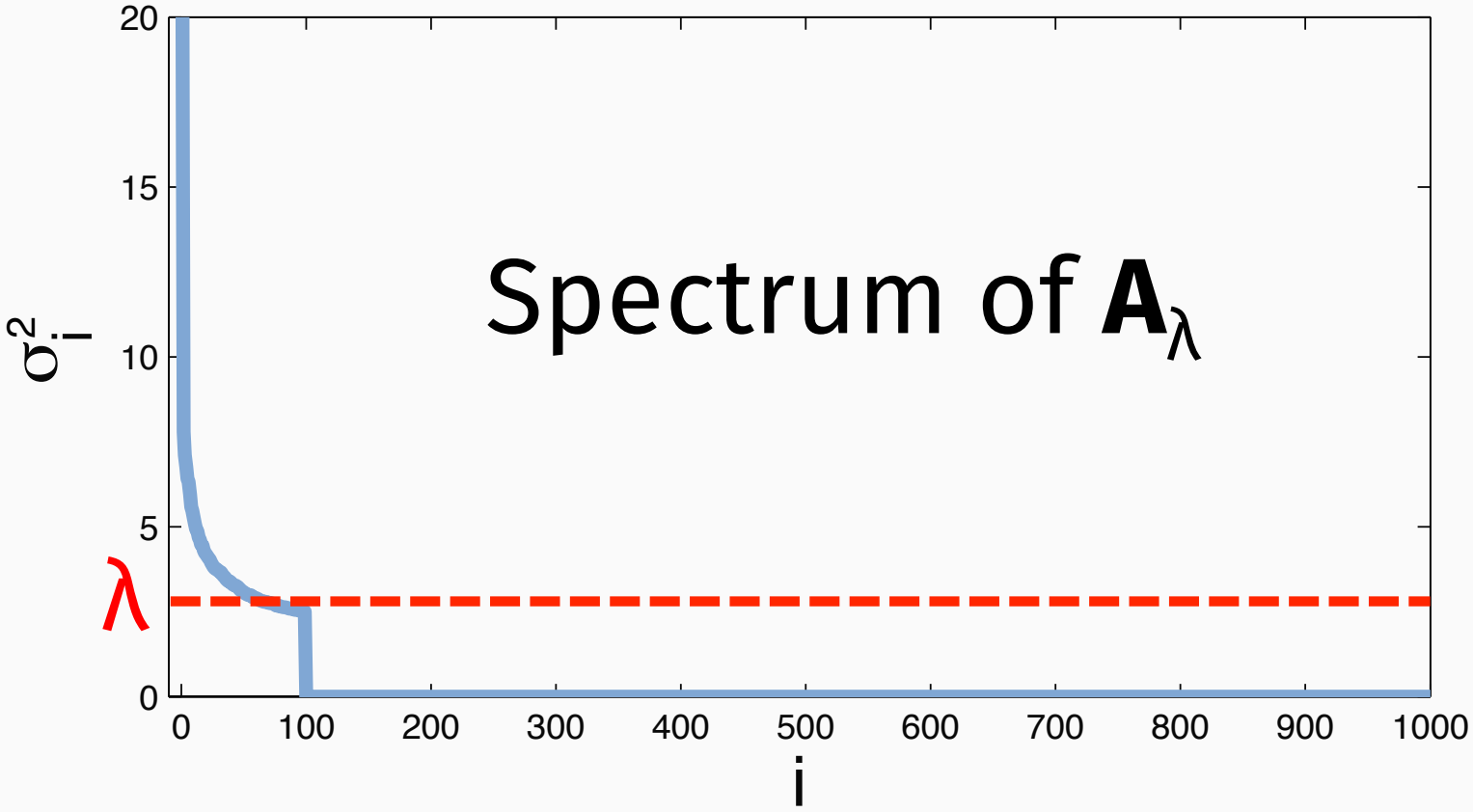
Theorem (Main Result)

There's an algorithm that approximately applies $\mathbf{A}_\lambda^\dagger$ *to any vector* $\mathbf{b}$ *using* $\approx \log(1/\epsilon)$ *well conditioned linear system solutions.*

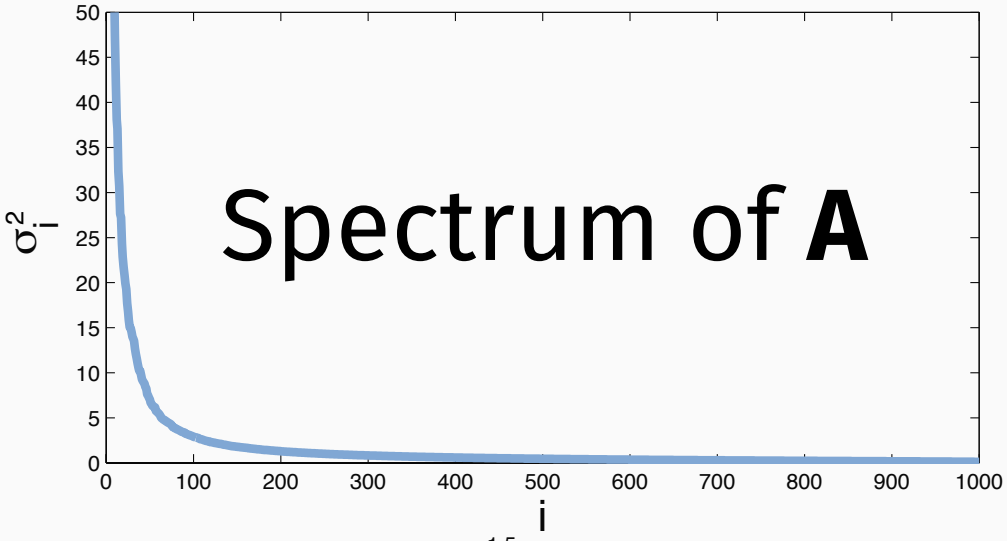
$$\text{PCR in} \approx O(\text{nnz}(\mathbf{A}) \cdot \sqrt{\kappa}) \text{ time.}$$

Goal: Apply $\mathbf{A}_\lambda^T$ quickly to any vector.

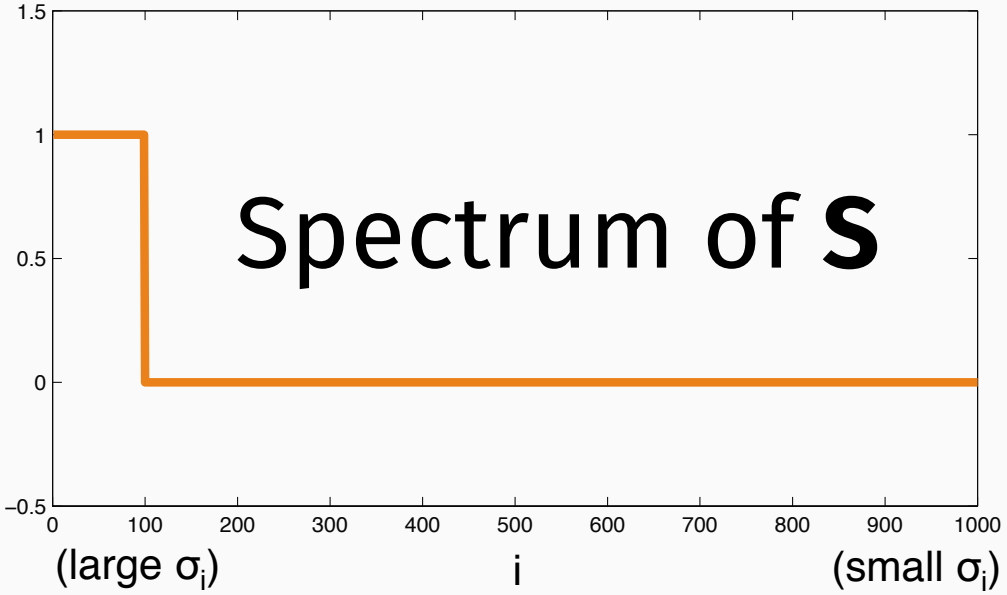
Goal: Apply $\mathbf{A}_\lambda^T$ quickly to any vector.



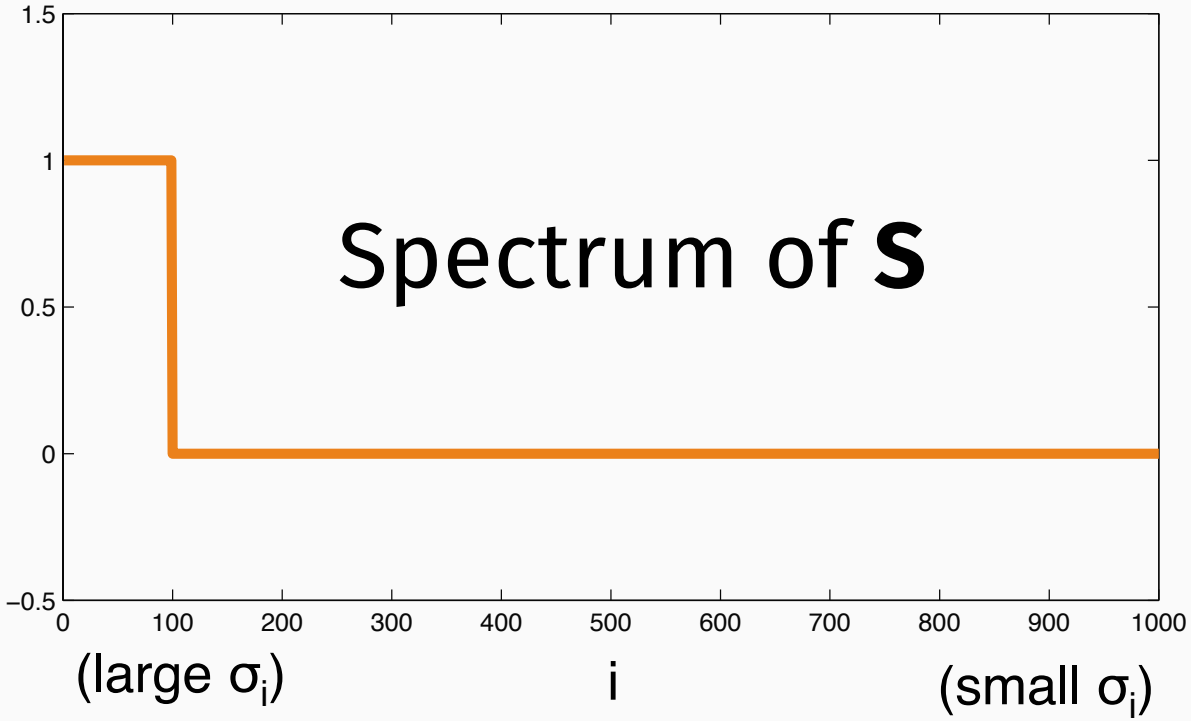
DECOMPOSING MATRIX



$$\mathbf{A}_\lambda^T = \mathbf{S}\mathbf{A}^T$$



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



How do we apply **S** to $\mathbf{A^T b}$?

How do we apply $\mathbf{S}$ to $\mathbf{A}^T\mathbf{b}$?

This is actually a common task.

How do we apply $\mathbf{S}$ to $\mathbf{A}^T\mathbf{b}$?

This is actually a common task.

- **Saad, Bekas, Kokiopoulou, Erhel, Guyomarc'h, Napoli, Polizzi, others**: Applications in eigenvalue counting, computational materials science, learning problems like eigenfaces and LSI, etc.

How do we apply $\mathbf{S}$ to $\mathbf{A}^T\mathbf{b}$?

This is actually a common task.

- **Saad, Bekas, Kokiopoulou, Erhel, Guyomarc'h, Napoli, Polizzi, others**: Applications in eigenvalue counting, computational materials science, learning problems like eigenfaces and LSI, etc.
- **Tremblay, Puy, Gribonval, Vandergheynst**: "Compressive Spectral Clustering" ICML 2016.

We turn to **Ridge Regression**, a popular alternative to PCR:

We turn to **Ridge Regression**, a popular alternative to PCR:

Goal: $\mathbf{x}^* = \arg\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2 + \lambda\|\mathbf{x}\|^2$.
Solution: $\mathbf{x}^* = (\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{A}^T\mathbf{b}$.

We turn to **Ridge Regression**, a popular alternative to PCR:

Goal: $\mathbf{x}^* = \arg\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2 + \lambda\|\mathbf{x}\|^2$.
Solution: $\mathbf{x}^* = (\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{A}^T\mathbf{b}$.

Claim:

$$\mathbf{R} = (\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{A}^T\mathbf{A}$$
coarsely approximates $\mathbf{S}$.

Singular values of S:

$$\sigma_i(\mathsf{S}) = \begin{cases} 1 & \text{if } \sigma_i^2(\mathsf{A}) \geq \lambda, \\ 0 & \text{if } \sigma_i^2(\mathsf{A}) < \lambda. \end{cases}$$

Singular values of S:

$$\sigma_i(\mathsf{S}) = \begin{cases} 1 & \text{if } \sigma_i^2(\mathsf{A}) \geq \lambda, \\ 0 & \text{if } \sigma_i^2(\mathsf{A}) < \lambda. \end{cases}$$

Singular values of $\mathsf{R} = (\mathsf{A}^T\mathsf{A} + \lambda\mathsf{I})^{-1}\mathsf{A}^T\mathsf{A}$:

$$\sigma_i(\mathsf{R}) = \frac{\sigma_i^2(\mathsf{A})}{\sigma_i^2(\mathsf{A}) + \lambda}$$

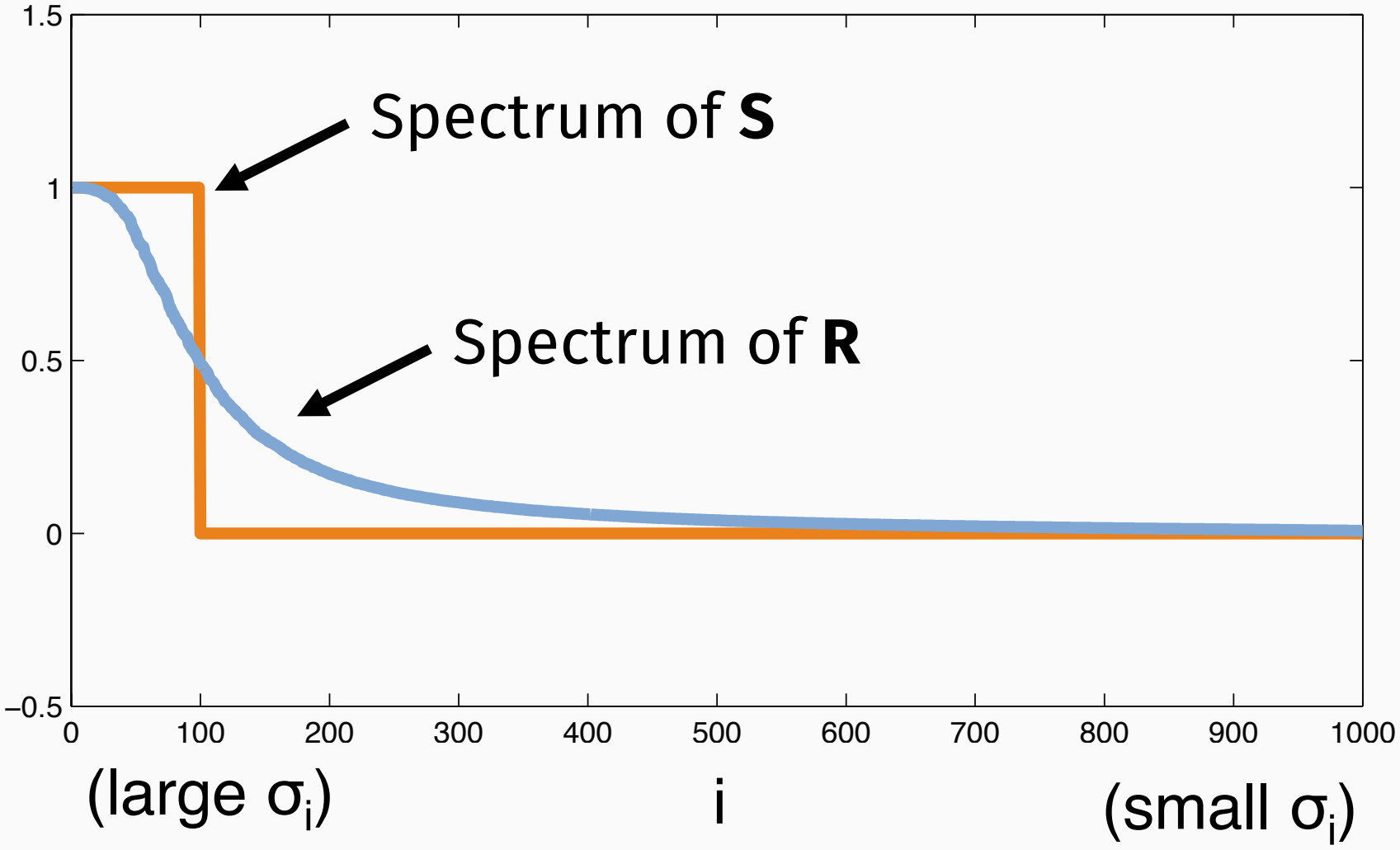
Singular values of $\mathbf{S}$:

$$\sigma_i(\mathbf{S}) = \begin{cases} 1 & \text{if } \sigma_i^2(\mathbf{A}) \geq \lambda, \\ 0 & \text{if } \sigma_i^2(\mathbf{A}) < \lambda. \end{cases}$$

Singular values of $\mathbf{R} = (\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{A}^T\mathbf{A}$:

$$\sigma_i(\mathbf{R}) = \frac{\sigma_i^2(\mathbf{A})}{\sigma_i^2(\mathbf{A}) + \lambda} \approx \begin{cases} 1 & \text{if } \sigma_i^2(\mathbf{A}) >> \lambda, \\ 0 & \text{if } \sigma_i^2(\mathbf{A}) << \lambda. \end{cases}$$

A FIRST APPROXIMATION



FINAL STEP

Easy to sharpen this approximation.

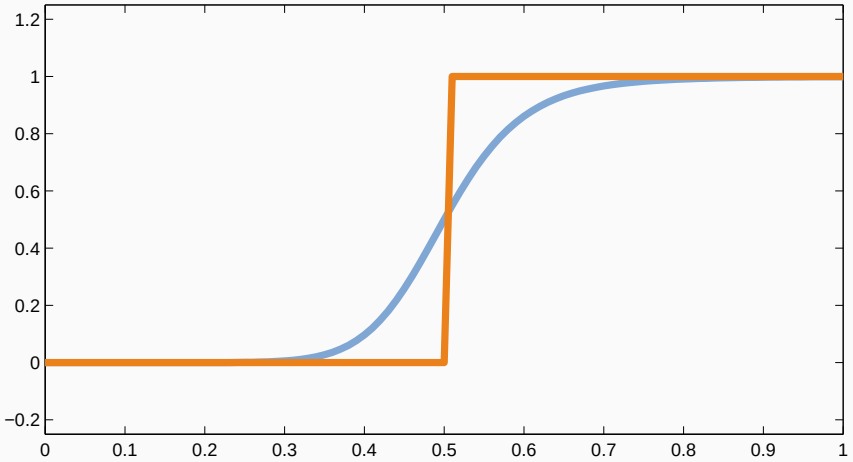
Easy to sharpen this approximation.

$$\sigma_i(\mathsf{R}) = \begin{cases} \geq 1/2 & \text{if } \sigma_i^2(\mathsf{A}) \geq \lambda, \\ < 1/2 & \text{if } \sigma_i^2(\mathsf{A}) < \lambda. \end{cases}$$

Easy to sharpen this approximation.

$$\sigma_i(\mathbf{R}) = \begin{cases} \geq 1/2 & \text{if } \sigma_i^2(\mathbf{A}) \geq \lambda, \\ < 1/2 & \text{if } \sigma_i^2(\mathbf{A}) < \lambda. \end{cases}$$

Compose $\mathbf{R}$ with approximate *symmetric* step function:

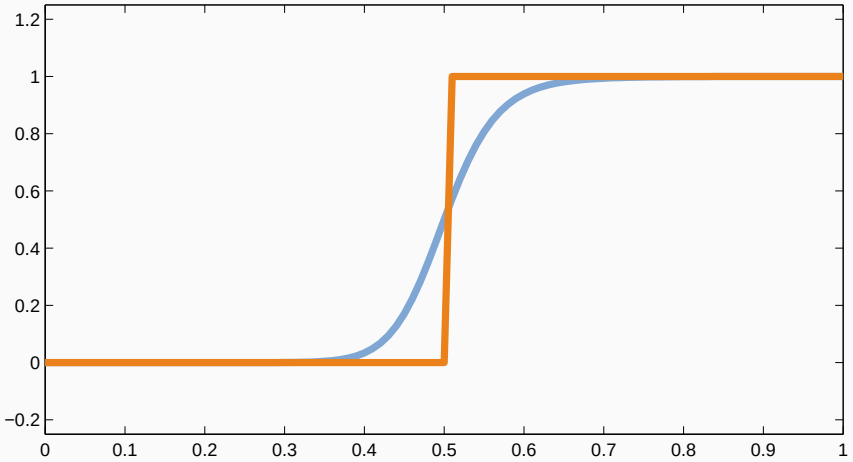


$\mathbf{S} \approx poly(\mathbf{R}) = c_1\mathbf{R} + c_2\mathbf{R}^2 +$

Easy to sharpen this approximation.

$$\sigma_i(\mathbf{R}) = \begin{cases} \geq 1/2 & \text{if } \sigma_i^2(\mathbf{A}) \geq \lambda, \\ < 1/2 & \text{if } \sigma_i^2(\mathbf{A}) < \lambda. \end{cases}$$

Compose $\mathbf{R}$ with approximate *symmetric* step function:

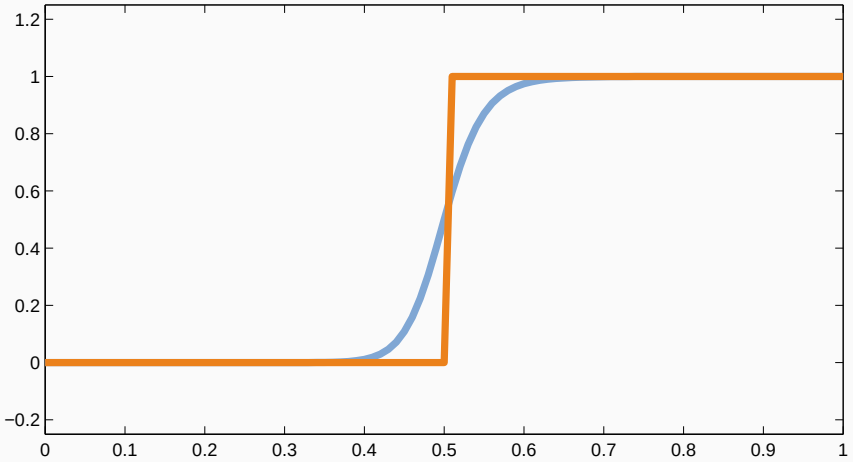


$$\mathbf{S} \approx poly(\mathbf{R}) = c_1\mathbf{R} + c_2\mathbf{R}^2 + c_3\mathbf{R}^3 +$$

Easy to sharpen this approximation.

$$\sigma_i(\mathbf{R}) = \begin{cases} \geq 1/2 & \text{if } \sigma_i^2(\mathbf{A}) \geq \lambda, \\ < 1/2 & \text{if } \sigma_i^2(\mathbf{A}) < \lambda. \end{cases}$$

Compose $\mathbf{R}$ with approximate *symmetric* step function:

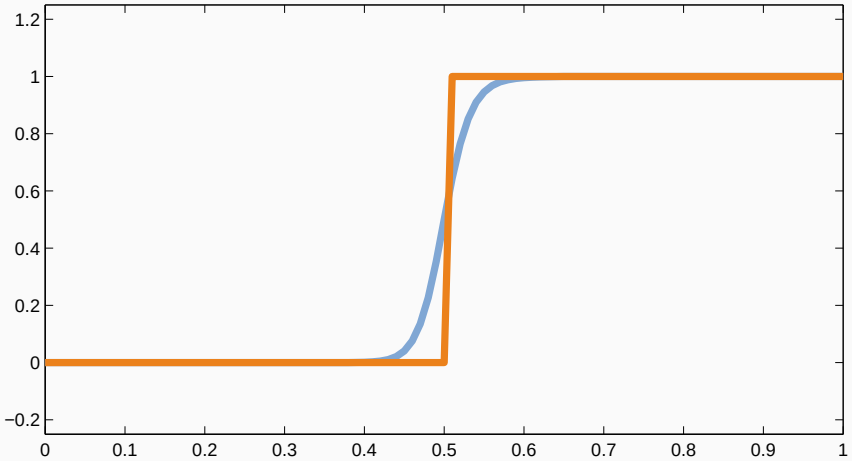


$$\mathbf{S} \approx poly(\mathbf{R}) = c_1\mathbf{R} + c_2\mathbf{R}^2 + c_3\mathbf{R}^3 + c_4\mathbf{R}^4 + \cdots$$

Easy to sharpen this approximation.

$$\sigma_i(\mathbf{R}) = \begin{cases} \geq 1/2 & \text{if } \sigma_i^2(\mathbf{A}) \geq \lambda, \\ < 1/2 & \text{if } \sigma_i^2(\mathbf{A}) < \lambda. \end{cases}$$

Compose $\mathbf{R}$ with approximate *symmetric* step function:



$$\mathbf{S} \approx \mathit{poly}(\mathbf{R}) = c_1\mathbf{R} + c_2\mathbf{R}^2 + c_3\mathbf{R}^3 + c_4\mathbf{R}^4 + c_5\mathbf{R}^5 + \dots$$

FINAL ALGORITHM

1. Compute $\mathbf{R}\mathbf{A}^T\mathbf{b}$, $\mathbf{R}^2\mathbf{A}^T\mathbf{b}$, $\ldots$ $\mathbf{R}^{O(\log 1/\epsilon)}\mathbf{A}^T\mathbf{b}$.

FINAL ALGORITHM

1. Compute $\mathbf{R}\mathbf{A}^T\mathbf{b}, \quad \mathbf{R}^2\mathbf{A}^T\mathbf{b}, \quad \ldots \quad \mathbf{R}^{O(\log 1/\epsilon)}\mathbf{A}^T\mathbf{b}$.
2. Approximate $\mathbf{S}\mathbf{A}^T\mathbf{b} \approx (c_1\mathbf{R} + c_2\mathbf{R}^2 + \ldots c_{O(\log 1/\epsilon)}\mathbf{R}^{O(\log 1/\epsilon)})\mathbf{A}^T\mathbf{b}$.

FINAL ALGORITHM

1. Compute $\mathbf{R}\mathbf{A}^T\mathbf{b}, \quad \mathbf{R}^2\mathbf{A}^T\mathbf{b}, \quad \ldots \quad \mathbf{R}^{O(\log 1/\epsilon)}\mathbf{A}^T\mathbf{b}$.
2. Approximate $\mathbf{S}\mathbf{A}^T\mathbf{b} \approx (c_1\mathbf{R} + c_2\mathbf{R}^2 + \ldots c_{O(\log 1/\epsilon)}\mathbf{R}^{O(\log 1/\epsilon)})\mathbf{A}^T\mathbf{b}$.
3. Apply $(\mathbf{A}^T\mathbf{A})^{-1}$ to $\mathbf{S}\mathbf{A}^T\mathbf{b}$

FINAL ALGORITHM

1. Compute $\mathbf{R}\mathbf{A}^T\mathbf{b}, \quad \mathbf{R}^2\mathbf{A}^T\mathbf{b}, \quad \ldots \quad \mathbf{R}^{O(\log 1/\epsilon)}\mathbf{A}^T\mathbf{b}$.
2. Approximate $\mathbf{S}\mathbf{A}^T\mathbf{b} \approx (c_1\mathbf{R} + c_2\mathbf{R}^2 + \ldots c_{O(\log 1/\epsilon)}\mathbf{R}^{O(\log 1/\epsilon)})\mathbf{A}^T\mathbf{b}$.
3. Apply $(\mathbf{A}^T\mathbf{A})^{-1}$ to $\mathbf{S}\mathbf{A}^T\mathbf{b}$

1. $O(\log 1/\epsilon)$ calls to a regression algorithm.

FINAL ALGORITHM

1. Compute $\mathbf{R}\mathbf{A}^T\mathbf{b}, \quad \mathbf{R}^2\mathbf{A}^T\mathbf{b}, \quad \ldots \quad \mathbf{R}^{O(\log 1/\epsilon)}\mathbf{A}^T\mathbf{b}$.
2. Approximate $\mathbf{S}\mathbf{A}^T\mathbf{b} \approx (c_1\mathbf{R} + c_2\mathbf{R}^2 + \ldots c_{O(\log 1/\epsilon)}\mathbf{R}^{O(\log 1/\epsilon)})\mathbf{A}^T\mathbf{b}$.
3. Apply $(\mathbf{A}^T\mathbf{A})^{-1}$ to $\mathbf{S}\mathbf{A}^T\mathbf{b}$

1. $O(\log 1/\epsilon)$ calls to a regression algorithm.
2. Low cost linear combination of vectors.

FINAL ALGORITHM

1. Compute $\mathbf{R}\mathbf{A}^T\mathbf{b}, \quad \mathbf{R}^2\mathbf{A}^T\mathbf{b}, \quad \ldots \quad \mathbf{R}^{O(\log 1/\epsilon)}\mathbf{A}^T\mathbf{b}$.
2. Approximate $\mathbf{S}\mathbf{A}^T\mathbf{b} \approx (c_1\mathbf{R} + c_2\mathbf{R}^2 + \ldots c_{O(\log 1/\epsilon)}\mathbf{R}^{O(\log 1/\epsilon)})\mathbf{A}^T\mathbf{b}$.
3. Apply $(\mathbf{A}^T\mathbf{A})^{-1}$ to $\mathbf{S}\mathbf{A}^T\mathbf{b}$

1. $O(\log 1/\epsilon)$ calls to a regression algorithm.
2. Low cost linear combination of vectors.
3. One call to a regression algorithm

NOTE ON PRIOR WORK

In prior work, $\mathbf{S}$ is approximated *directly* using a matrix polynomial. Why not here?

NOTE ON PRIOR WORK

In prior work, $\mathbf{S}$ is approximated *directly* using a matrix polynomial. Why not here?

- We match polynomial approximation, but can be faster when non-standard regression algorithms are used.
- We give a full end-to-end runtime and stability analysis.

FINAL ALGORITHM

I’m leaving out a lot of details...

FINAL ALGORITHM

I'm leaving out a lot of details...

- Analysis of error propagation through approximate operations.
- Recurrence for stable application of symmetric step polynomial.
- More work to make last regression step stable and fast.

FINAL ALGORITHM

I'm leaving out a lot of details...

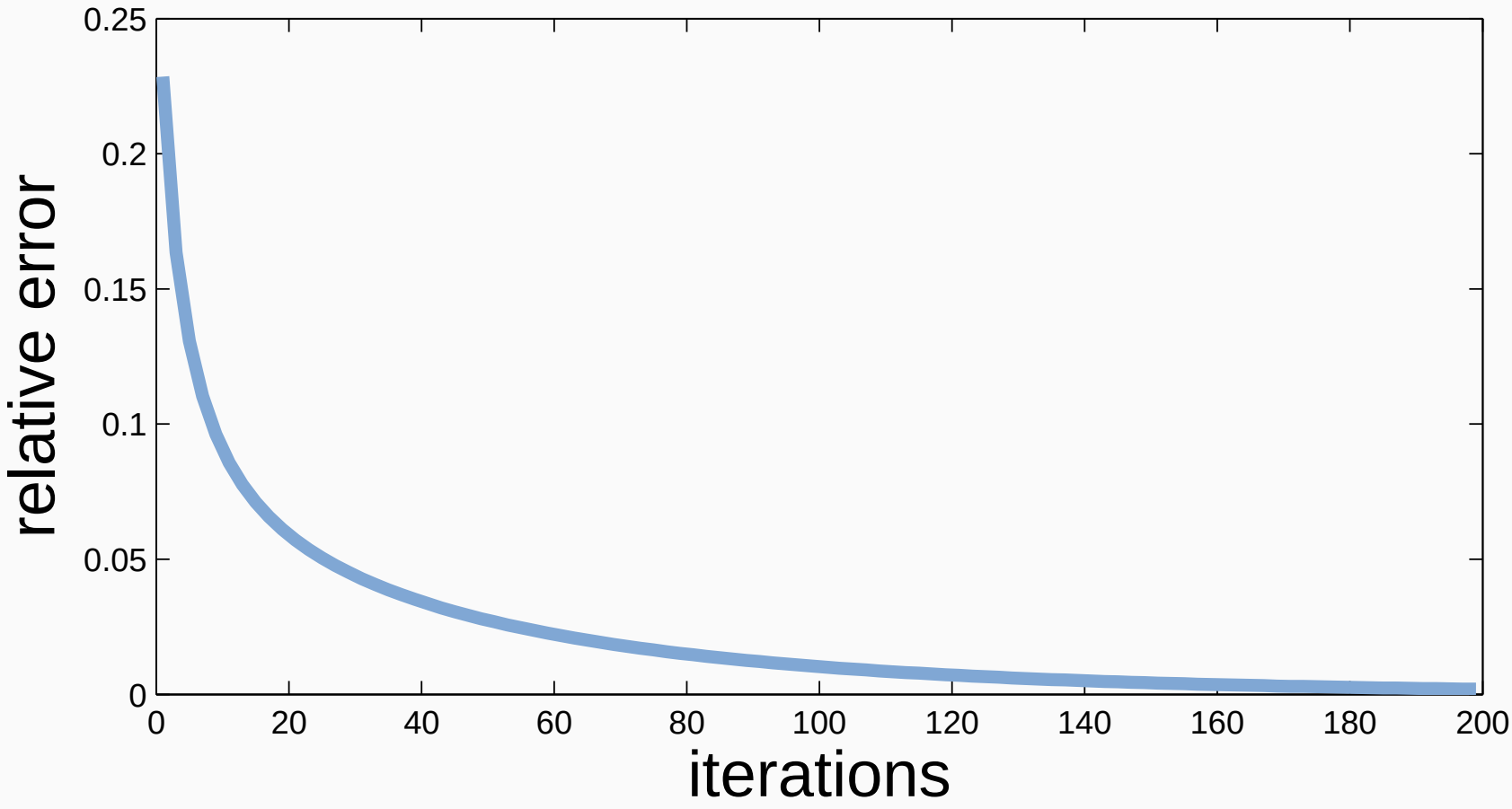
- Analysis of error propagation through approximate operations.
- Recurrence for stable application of symmetric step polynomial.
- More work to make last regression step stable and fast.

But the algorithm itself remains simple!

SAMPLE CODE

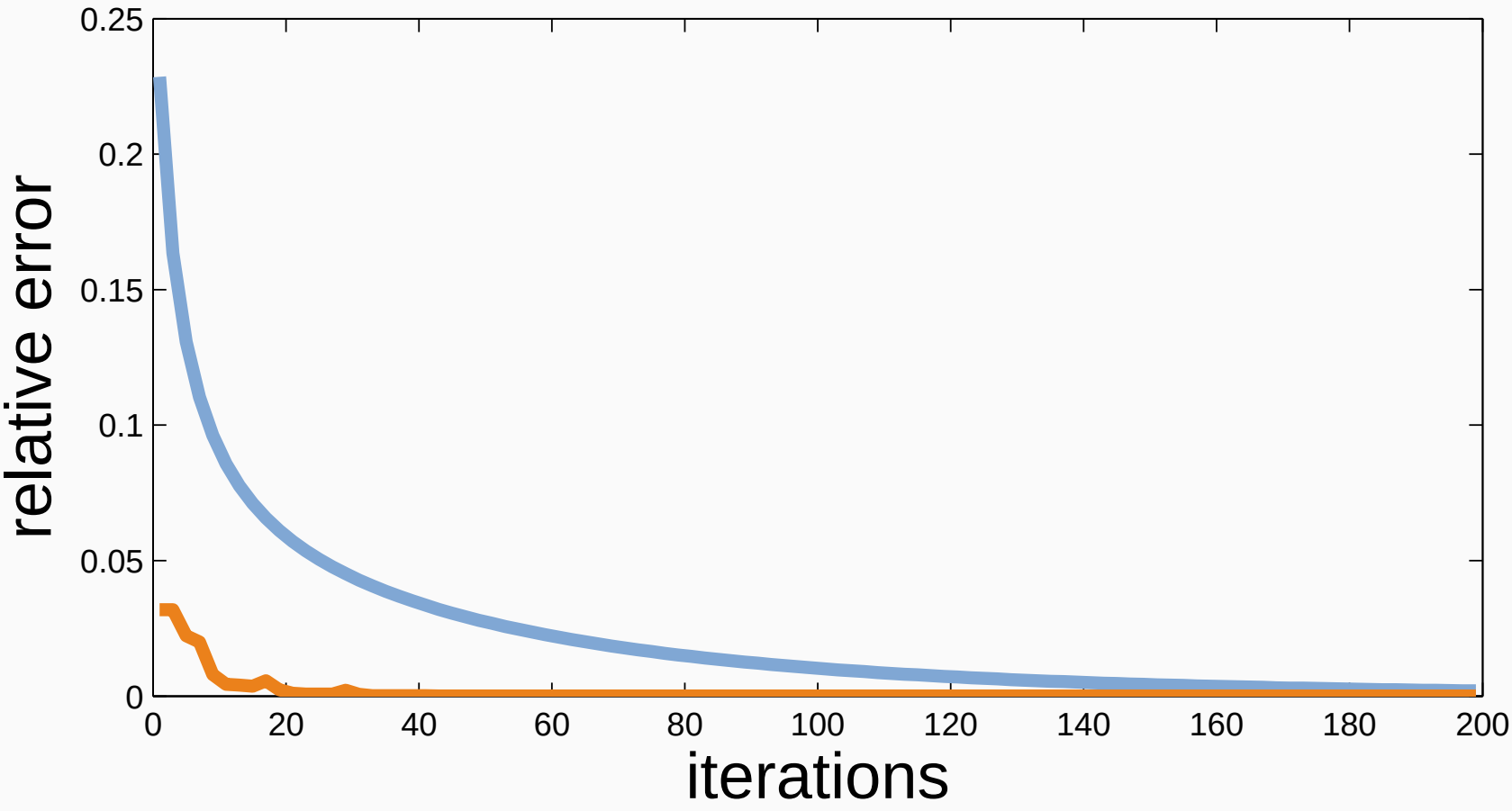
```
function [x,patb] = fpcr(A, b, lambda, iter)
z = A'*b;
pz = ridgeReg(A,A*z,lambda);

w = pz - z/2;
for i = 1:iter
    w = 4*(2*i+1)/(2*i)*ridgeReg(A, ...
        A*(w - ridgeReg(A,A*w,lambda)), lambda);
    pz = pz + 1/(2*i+1)*w;
end
patb = pz;

x = robustReg(A,pz,lambda);
end

function x = robustReg(A,pz,lambda)
    tol = 1e-5; %default
    function y = afun(z,~)
        y = A'*(A*z) + tol*lambda*z;
    end
    [x,~] = pcg(@afun,pz);
end
```

Synthetic data (with small spectral gap)



Standard Algorithm

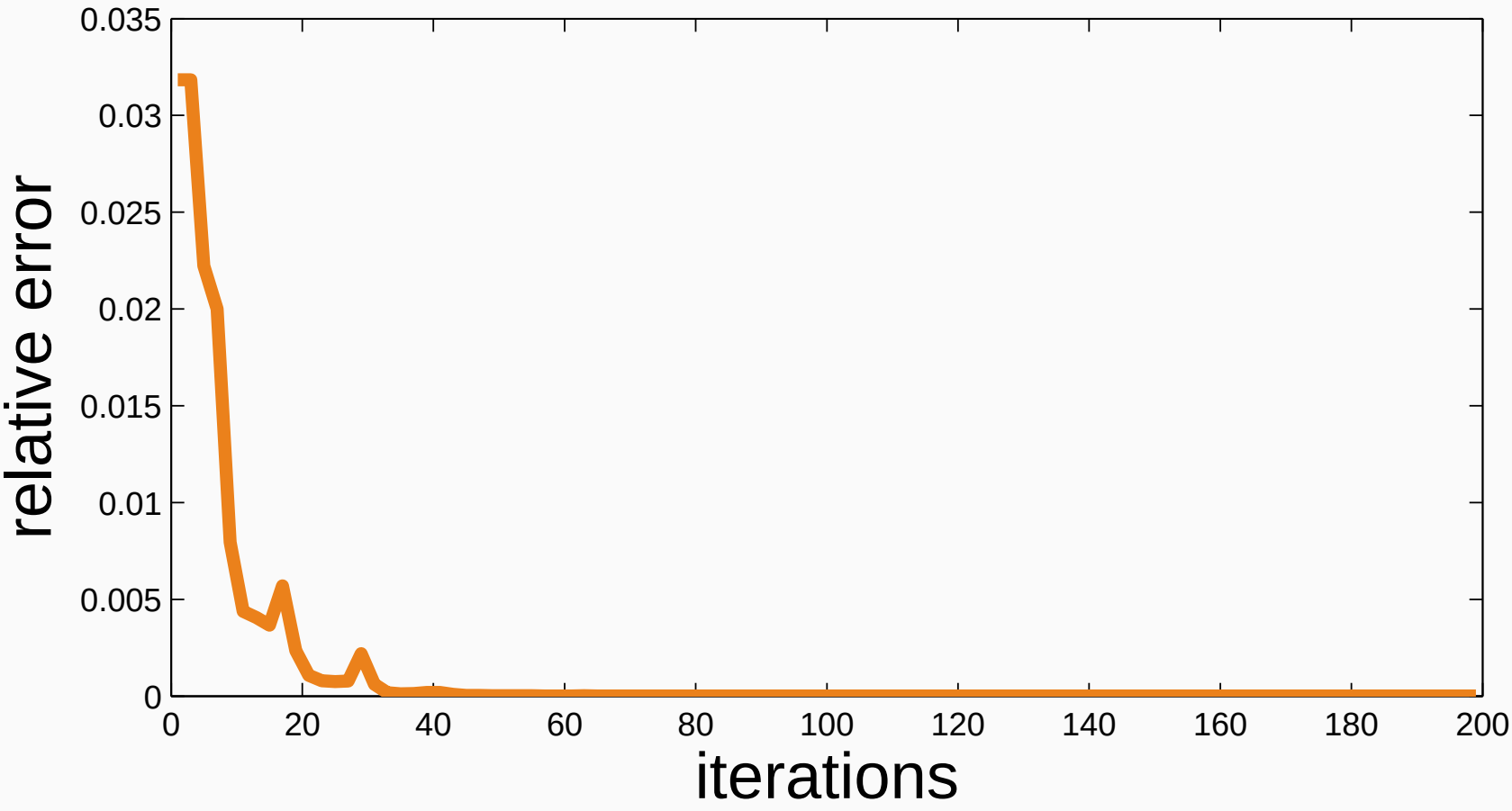
Synthetic data (with small spectral gap)



Standard Algorithm

Krylov Accelerated Algorithm

Synthetic data (with small spectral gap)



Standard Algorithm

Krylov Accelerated Algorithm

Take away: PCR can be computed *without explicit PCA*.

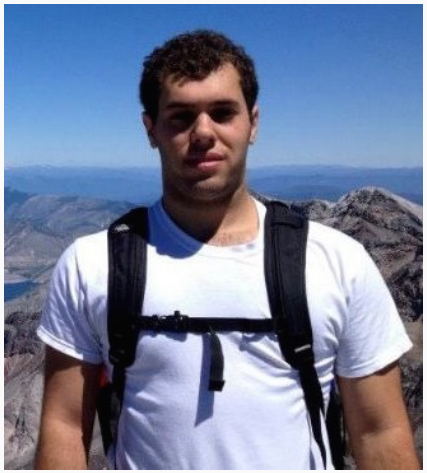
Ridge regression + matrix polynomial = efficient operator access to $\mathbf{A}$'s top principal components.

Take away: PCR can be computed *without explicit PCA*.

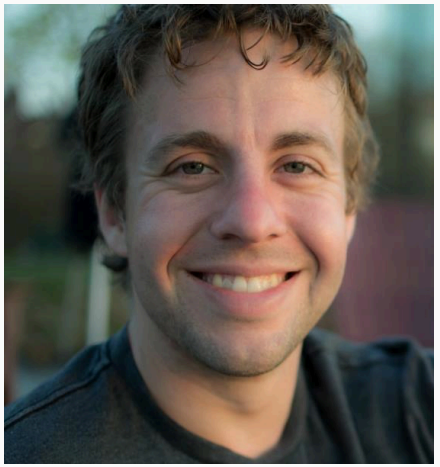
Ridge regression + matrix polynomial = efficient operator access to $\mathbf{A}$'s top principal components.

Questions? Joint work with:

Roy Frostig



Cameron Musco



Aaron Sidford