PRINCIPAL COMPONENT REGRESSION
WITHOUT PRINCIPAL COMPONENT ANALYSIS

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

¹Stanford, ²MIT, ³Microsoft Research
Paper, slides, and template code available at chrismusco.com
Simple, robust algorithms for principal component regression.
Principal Component Regression (PCR) =

Principal Component Analysis (unsupervised) +

Linear Regression (supervised)
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

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: $A$, $b$
Solve: $x^* = \arg\min_x \|Ax - b\|^2$
Standard Regression:

Given: $A$, $b$
Solve: $x^* = \arg\min_x \|Ax - b\|^2$

Principal Component Regression:

Given: $A$, $b$, $\lambda$
Solve: $x^* = \arg\min_x \|A\lambda x - b\|^2$
**Standard Regression:**

Given: \( A, b \)

Solve: \( x^* = \arg \min_x \|Ax - b\|^2 \)

**Principal Component Regression:**

Given: \( A, b, \lambda \)

Solve: \( x^* = \arg \min_x \|A\lambda x - b\|^2 \)
Standard Regression:

Given: $A, b$
Solve: $x^* = \arg \min_x \|Ax - b\|^2$

Principal Component Regression:

Given: $A, b, \lambda$
Solve: $x^* = \arg \min_x \|A_\lambda x - b\|^2$
Singular values of $A$
FORMAL SETUP

Singular values of $A$

$\sigma_i$

“Signal”

“Noise”

$\lambda$

i

$0 \ 100 \ 200 \ 300 \ 400 \ 500 \ 600 \ 700 \ 800 \ 900 \ 1000$

$0 \ 5 \ 10 \ 15 \ 20$

$\sigma_i$

$\lambda$

$\sigma_i$

“Signal”

“Noise”
Singular values of $A_\lambda$
Principal Component Regression (PCR):

Goal: \( x^* = \arg \min_x \|A_\lambda x - b\|^2 \)

Solution: \( x = (A_\lambda^T A_\lambda)^{-1} A_\lambda^T b \)

What’s the computational cost?
Cost of computing $A_\lambda$ (PCA):

$$\approx O(\text{nnz}(A)k + dk^2).$$
Cost of computing $A_\lambda$ (PCA):

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

$k$ is the number of principal components with value $> \lambda$. 
Cost of computing $A_\lambda$ (PCA):

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

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

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

\[ \approx O(\text{nnz}(A) \cdot \sqrt{\kappa}). \]
Cost of computing $A_\lambda$ (PCA):

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

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

Cost of evaluating $x = (A_\lambda^TA_\lambda)^{-1}A_\lambda^Tb$ (regression):

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

For PCR, $k$ is large, $\kappa$ is small ($A_\lambda$ is well conditioned).
Goal: Remove bottleneck dependence on $k$
Optimistic observation: PCA computes too much information.
Optimistic observation: PCA computes too much information.

\[ x^* = (A^T_\lambda A_\lambda)^{-1} A^T_\lambda b = \]
Optimistic observation: PCA computes too much information.

\[ x^* = (A^T \Lambda A) \Lambda^{-1} A^T b = (A^T A)^{-1} A^T \Lambda b \]
Optimistic observation: PCA computes too much information.

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

Don’t need to compute \( A_\lambda \) (which incurs a \( k \) dependence) as long as we can apply it to a single vector efficiently.
It’s very often more efficient to apply a matrix function once than compute the matrix function explicitly.
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 \((A^TA)^3x\)
It’s very often more efficient to apply a matrix function once than compute the matrix function explicitly.

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

Why not $A_\lambda$?
Theorem (Main Result)

There’s an algorithm that approximately applies $A^n$ to any well conditioned linear system solutions.
Theorem (Main Result)

There’s an algorithm that approximately applies $A^T_\lambda$ to any vector $b$ using $\approx \log(1/\epsilon)$ well conditioned linear system solutions.

PCR in $\approx O(\text{nnz}(A) \cdot \sqrt{\kappa})$ time.
Goal: Apply $A^T_\lambda$ quickly to any vector.
**Goal:** Apply $A^T_\lambda$ quickly to any vector.
DECOMPOSING MATRIX

\[ \mathbf{A}_\lambda^T = \mathbf{S} \mathbf{A}^T \]

Spectrum of \( \mathbf{A} \)

Spectrum of \( \mathbf{S} \)

(large \( \sigma_i \))

(small \( \sigma_i \))
DECOMPOSING MATRIX

\[ A^T \mathbf{b} = S A^T \mathbf{b} \]

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

This is actually a common task.
How do we apply $S$ to $A^Tb$?

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 $S$ to $A^Tb$?

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:

\[
\begin{align*}
    \text{Goal:} & \quad x = \arg \min_{x} \|Ax - b\|_2 + \|x\|_2 \\
    \text{Solution:} & \quad x = \left( \begin{bmatrix} A^T & \mathbf{I} \end{bmatrix} \begin{bmatrix} A & \mathbf{I} \end{bmatrix} \right)^{-1} A^T b \\
    \text{Claim:} & \quad R = \left( \begin{bmatrix} A^T & \mathbf{I} \end{bmatrix} \begin{bmatrix} A & \mathbf{I} \end{bmatrix} \right)^{-1} A^T \text{coarsely approximates } S.
\end{align*}
\]
We turn to **Ridge Regression**, a popular alternative to PCR:

Goal: \( x^* = \arg\min_x \|Ax - b\|^2 + \lambda \|x\|^2 \).

Solution: \( x^* = (A^TA + \lambda I)^{-1}A^Tb \).
We turn to Ridge Regression, a popular alternative to PCR:

Goal: \( \mathbf{x}^* = \arg \min_{\mathbf{x}} \| \mathbf{A}\mathbf{x} - \mathbf{b} \|_2^2 + \lambda \| \mathbf{x} \|_2^2. \)

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 $S$: 

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

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

$$
\sigma_i(\mathbf{R}) = \frac{\sigma_i^2(\mathbf{A})}{\sigma_i^2(\mathbf{A}) + \lambda}
$$
Singular values of $S$:

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

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

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

Spectrum of $S$

Spectrum of $R$

(large $\sigma_i$) $i$ (small $\sigma_i$)
Easy to sharpen this approximation.
Easy to sharpen this approximation.

\[ \sigma_i(R) = \begin{cases} 
  \geq 1/2 & \text{if } \sigma_i^2(A) \geq \lambda, \\
  < 1/2 & \text{if } \sigma_i^2(A) < \lambda.
\end{cases} \]
Easy to sharpen this approximation.

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

Compose \( R \) with approximate symmetric step function:

\[ S \approx poly(R) = c_1 R + c_2 R^2 + \]
Easy to sharpen this approximation.

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

Compose \( R \) with approximate symmetric step function:

\[ S \approx poly(R) = c_1R + c_2R^2 + c_3R^3 + \]
Easy to sharpen this approximation.

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

Compose \( R \) with approximate \textit{symmetric} step function:

\[ S \approx poly(R) = c_1 R + c_2 R^2 + c_3 R^3 + c_4 R^4 + \ldots \]
Easy to sharpen this approximation.

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

Compose $R$ with approximate symmetric step function:

$$S \approx \text{poly}(R) = c_1R + c_2R^2 + c_3R^3 + c_4R^4 + c_5R^5 + \ldots$$
1. Compute \( RA^T b, \ R^2 A^T b, \ldots \ R^{O(\log 1/\epsilon)} A^T b \).
1. Compute $RA^Tb$, $R^2A^Tb$, $\ldots$, $R^{O(\log 1/\epsilon)}A^Tb$.

2. Approximate $SA^Tb \approx (c_1R + c_2R^2 + \ldots c_{O(\log 1/\epsilon)}R^{O(\log 1/\epsilon)})A^Tb$. 
1. Compute $R^{A^T}b$, $R^{2A^T}b$, \ldots $R^{O(\log \frac{1}{\epsilon})A^T}b$. 

2. Approximate $SA^Tb \approx (c_1R + c_2R^2 + \ldots c_{O(\log \frac{1}{\epsilon})}R^{O(\log \frac{1}{\epsilon})})A^Tb$. 

3. Apply $(A^TA)^{-1}$ to $SA^Tb$.
1. Compute $\mathbf{R}^T \mathbf{b}$, $\mathbf{R}^2 \mathbf{A}^T \mathbf{b}$, \ldots $\mathbf{R}^{O(\log 1/\epsilon)} \mathbf{A}^T \mathbf{b}$.

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

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

3. Apply $(A^T A)^{-1}$ to $SA^T b$ 

1. $O(\log 1/\epsilon)$ calls to a regression algorithm. 
2. Low cost linear combination of vectors.
1. Compute $R A^T b$, $R^2 A^T b$, \ldots $R^{O(\log 1/\epsilon)} A^T b$.
2. Approximate $S A^T b \approx (c_1 R + c_2 R^2 + \ldots c_{O(\log 1/\epsilon)} R^{O(\log 1/\epsilon)}) A^T b$.
3. Apply $(A^T A)^{-1}$ to $S A^T b$

1. $O(\log 1/\epsilon)$ calls to a regression algorithm.
2. Low cost linear combination of vectors.
3. One call to a regression algorithm
In prior work, $S$ is approximated *directly* using a matrix polynomial. Why not here?
In prior work, $S$ 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.
I’m leaving out a lot of details...
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.
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!
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)

The graph shows the relative error over iterations for two algorithms:

- **Standard Algorithm**
- **Krylov Accelerated Algorithm**

The y-axis represents the relative error, ranging from 0 to 0.25, and the x-axis represents the number of iterations, ranging from 0 to 200.
Synthetic data (with small spectral gap)

![Graph showing experimental results for Standard Algorithm and Krylov Accelerated Algorithm.]
Synthetic data (with small spectral gap)
**Take away:** PCR can be computed *without explicit PCA*.

Ridge regression + matrix polynomial = efficient operator access to A’s top principal components.
**Take away:** PCR can be computed *without explicit PCA.*

Ridge regression + matrix polynomial = efficient operator access to $A$’s top principal components.

Questions? Joint work with:

Roy Frostig  
Cameron Musco  
Aaron Sidford