Randomized matrix decompositions for faster scientific computing

Robert J. Webber[1]

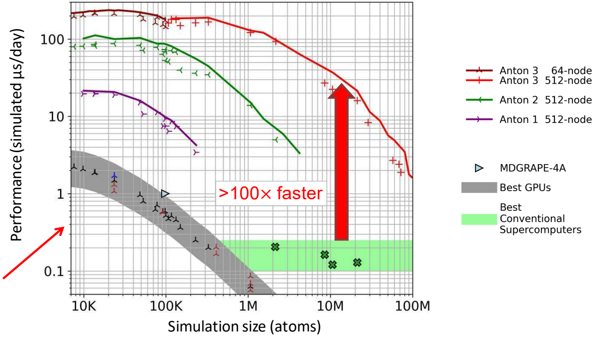
[1]Computing + Mathematical Sciences, California Institute of Technology

January 12, 2024

Processing data is slow.

Computers are producing massive amounts of data.

Figure: Anton 3 supercomputer



log scale!

Figure: Each day, Anton 3 simulates 10^6 atoms for 10^{11} time steps.

Processing data is slow.

Computers are producing massive amounts of data.

Figure: Anton 3 supercomputer

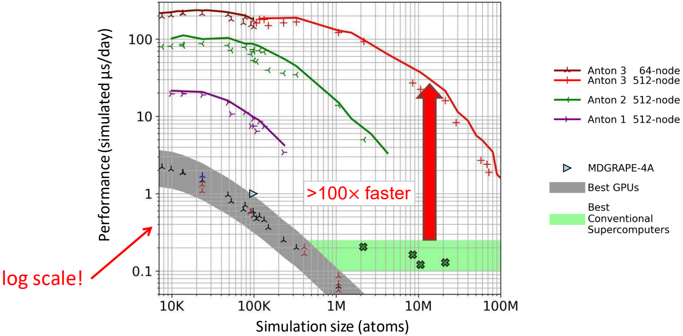


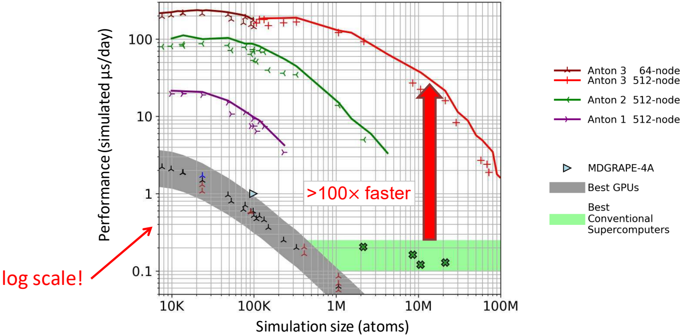
Figure: Each day, Anton 3 simulates 10^6 atoms for 10^{11} time steps.

Question. Let's say we want to analyze 10^4 molecules, how large is a $10^4 \times 3$ million array?

Processing data is slow.

Computers are producing massive amounts of data.

Figure: Anton 3 supercomputer



>100× faster

log scale!

Figure: Each day, Anton 3 simulates 10^6 atoms for 10^{11} time steps.

Question. Let's say we want to analyze 10^4 molecules, how large is a $10^4 \times 3$ million array?

Answer. Storage $\approx 8 \times r$ rows $\times$ c columns$/10^9 = 240$GB.

Processing data is slow.

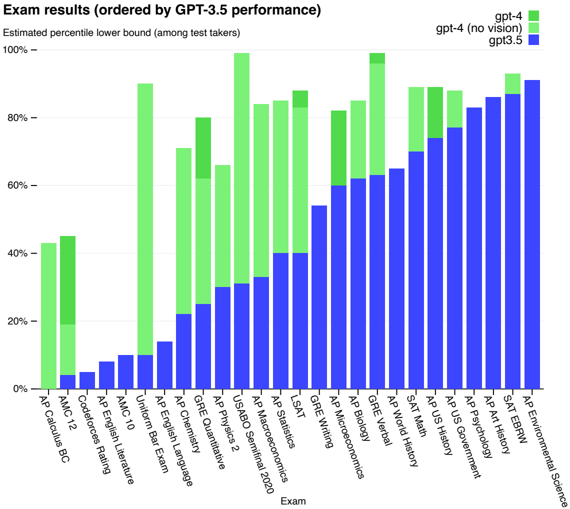
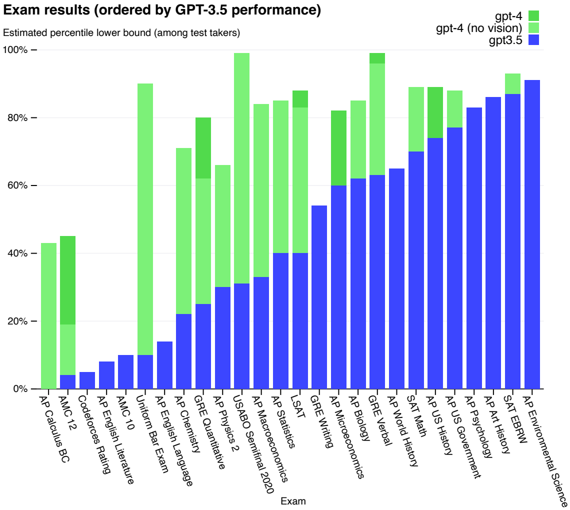


Figure: GPT-4 is a large language model with ≈ 1 trillion parameters

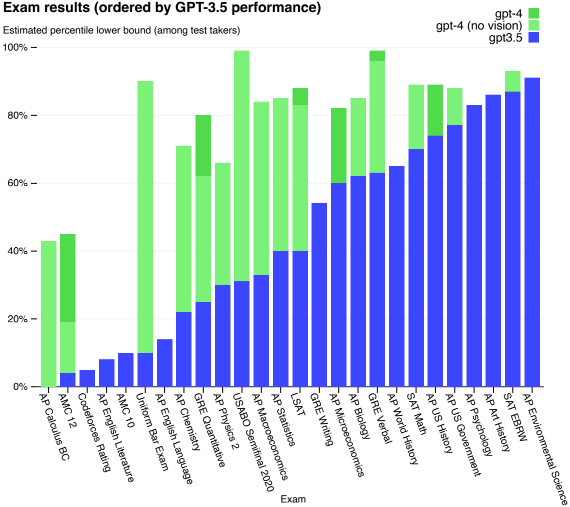
Processing data is slow.



Question. How long does it take to train GPT-4?

Figure: GPT-4 is a large language model with ≈ 1 trillion parameters

Processing data is slow.



Question. How long does it take to train GPT-4?

Answer. Training requires 25,000 GPUs working constantly over 100 days.

Figure: GPT-4 is a large language model with ≈ 1 trillion parameters

Processing data is slow.

Computers are producing massive amounts of data.

Processing data is slow.

Computers are producing massive amounts of data.

But processing data is difficult.

Processing data is slow.

Computers are producing massive amounts of data.

But processing data is difficult.

Computers are often solving linear systems and eigenvalue problems —
if we can accelerate these primitives, we can accelerate computation overall.

Processing data is slow.

Computers are producing massive amounts of data.

But processing data is difficult.

Computers are often solving linear systems and eigenvalue problems —
if we can accelerate these primitives, we can accelerate computation overall.

Randomized iterative methods lead to speed-ups for linear systems and eigenvalue problems.

Processing data is slow.

Computers are producing massive amounts of data.

But processing data is difficult.

Computers are often solving linear systems and eigenvalue problems —
if we can accelerate these primitives, we can accelerate computation overall.

Randomized iterative methods lead to speed-ups for linear systems and eigenvalue problems.

Randomized iterative methods identify low-rank structure in large matrices.

- Find structure by repeatedly randomly searching.
- Low-rank structure leads to computational speed-ups.

How do we find dominant eigenvectors fast?

Consider three approaches to calculating the dominant eigenvectors of a matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, which is positive semidefinite (psd), i.e., symmetric with nonnegative eigenvalues.

How do we find dominant eigenvectors fast?

Consider three approaches to calculating the dominant eigenvectors of a matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, which is positive semidefinite (psd), i.e., symmetric with nonnegative eigenvalues.

Classical QR iteration. Gives a full eigendecomposition $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$ in $\mathcal{O}(N^3)$ operations.

How do we find dominant eigenvectors fast?

Consider three approaches to calculating the dominant eigenvectors of a matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, which is positive semidefinite (psd), i.e., symmetric with nonnegative eigenvalues.

Classical QR iteration. Gives a full eigendecomposition $\boldsymbol{A} = \boldsymbol{Q\Lambda Q}^T$ in $\mathcal{O}(N^3)$ operations.

Randomized block Krylov iteration. Gives an approximation of the top r eigenvectors and eigenvalues in $\mathcal{O}(N^2 r)$ operations.

How do we find dominant eigenvectors fast?

Consider three approaches to calculating the dominant eigenvectors of a matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, which is positive semidefinite (psd), i.e., symmetric with nonnegative eigenvalues.

Classical QR iteration. Gives a full eigendecomposition $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$ in $\mathcal{O}(N^3)$ operations.

Randomized block Krylov iteration. Gives an approximation of the top r eigenvectors and eigenvalues in $\mathcal{O}(N^2 r)$ operations.

- *RBKI is accurate even without eigenvalue decay (see Tropp & Webber, 2023).*

How do we find dominant eigenvectors fast?

Consider three approaches to calculating the dominant eigenvectors of a matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, which is positive semidefinite (psd), i.e., symmetric with nonnegative eigenvalues.

Classical QR iteration. Gives a full eigendecomposition $\boldsymbol{A} = \boldsymbol{Q \Lambda Q}^T$ in $\mathcal{O}(N^3)$ operations.

Randomized block Krylov iteration. Gives an approximation of the top r eigenvectors and eigenvalues in $\mathcal{O}(N^2 r)$ operations.

- *RBKI is accurate even without eigenvalue decay (see Tropp & Webber, 2023)*.

Randomly pivoted Cholesky. Gives an approximation of the top r eigenvectors and eigenvalues in $\mathcal{O}(Nr^2)$ operations, using $\mathcal{O}(r)$ adaptively sampled columns.

How do we find dominant eigenvectors fast?

Consider three approaches to calculating the dominant eigenvectors of a matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, which is positive semidefinite (psd), i.e., symmetric with nonnegative eigenvalues.

Classical QR iteration. Gives a full eigendecomposition $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$ in $\mathcal{O}(N^3)$ operations.

Randomized block Krylov iteration. Gives an approximation of the top r eigenvectors and eigenvalues in $\mathcal{O}(N^2 r)$ operations.

- *RBKI is accurate even without eigenvalue decay (see Tropp & Webber, 2023).*

Randomly pivoted Cholesky. Gives an approximation of the top r eigenvectors and eigenvalues in $\mathcal{O}(Nr^2)$ operations, using $\mathcal{O}(r)$ adaptively sampled columns.

- *RPC is accurate given eigenvalue decay (Chen, Epperly, Tropp, & Webber, 2023).*

Low-rank approximation

The modern approach for calculating dominant eigenvectors of a positive semidefinite (psd) matrix is based on **low-rank approximation**:

Low-rank approximation

The modern approach for calculating dominant eigenvectors of a positive semidefinite (psd) matrix is based on **low-rank approximation**:

- Form approximation $\hat{\boldsymbol{A}} \approx \boldsymbol{A}$ where $\operatorname{rank}(\hat{\boldsymbol{A}}) \ll \operatorname{rank}(\boldsymbol{A})$.

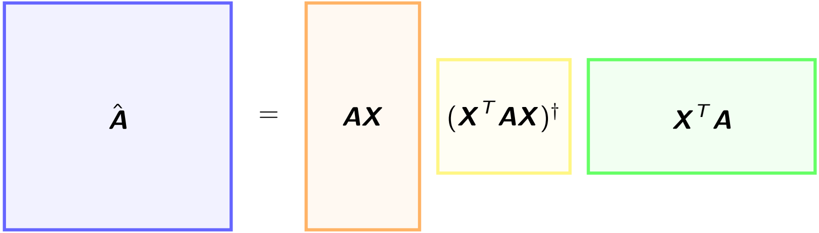
Low-rank approximation

The modern approach for calculating dominant eigenvectors of a positive semidefinite (psd) matrix is based on **low-rank approximation**:

- Form approximation $\hat{\boldsymbol{A}} \approx \boldsymbol{A}$ where $\operatorname{rank}(\hat{\boldsymbol{A}}) \ll \operatorname{rank}(\boldsymbol{A})$.
- Dominant eigenvectors and eigenvalues of $\hat{\boldsymbol{A}}$ approximate dominant eigenvectors and eigenvalues of $\boldsymbol{A}$.

Low-rank approximation

If $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ is psd, we use the Nyström approximation:

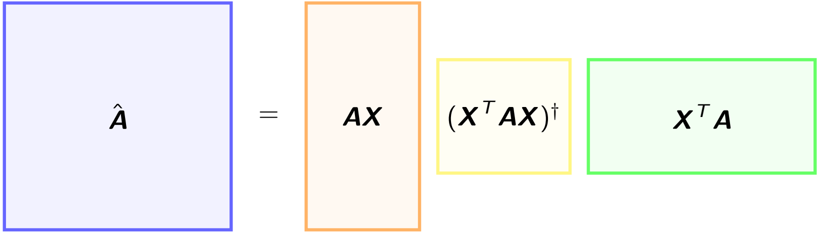


$$\hat{\boldsymbol{A}} = \boldsymbol{AX} \; (\boldsymbol{X}^T\boldsymbol{AX})^{\dagger} \; \boldsymbol{X}^T\boldsymbol{A}$$

$\boldsymbol{X} \in \mathbb{R}^{N \times k}$ is the test matrix, $\boldsymbol{AX} \in \mathbb{R}^{N \times k}$ is the output matrix, T is the transpose, $\dagger$ is the pseudoinverse.

Low-rank approximation

If $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ is psd, we use the Nyström approximation:



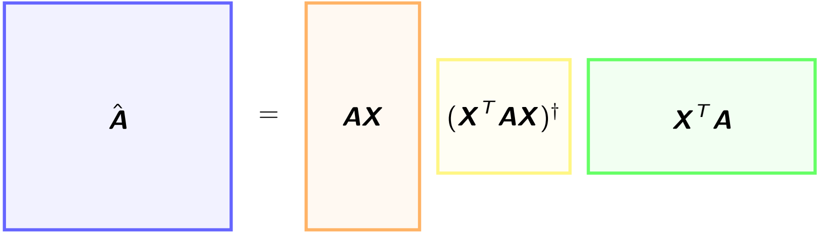
$$\hat{\boldsymbol{A}} = \boldsymbol{AX} \, (\boldsymbol{X}^T\boldsymbol{AX})^\dagger \, \boldsymbol{X}^T\boldsymbol{A}$$

$\boldsymbol{X} \in \mathbb{R}^{N \times k}$ is the test matrix, $\boldsymbol{AX} \in \mathbb{R}^{N \times k}$ is the output matrix, T is the transpose, $\dagger$ is the pseudoinverse.

1. All the columns in $\hat{\boldsymbol{A}}$ are linear combinations of the columns $\boldsymbol{AX}$.

Low-rank approximation

If $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ is psd, we use the Nyström approximation:



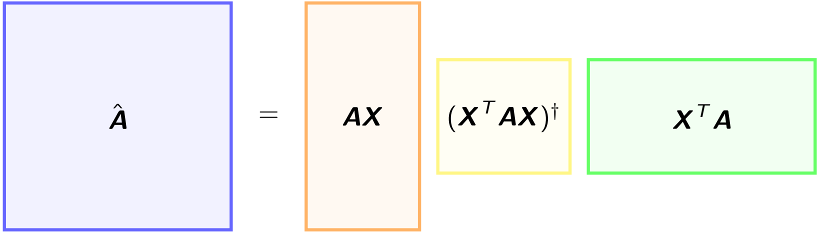
$$\hat{\boldsymbol{A}} = \boldsymbol{AX} \, (\boldsymbol{X}^T\boldsymbol{AX})^\dagger \, \boldsymbol{X}^T\boldsymbol{A}$$

$\boldsymbol{X} \in \mathbb{R}^{N \times k}$ is the test matrix, $\boldsymbol{AX} \in \mathbb{R}^{N \times k}$ is the output matrix, T is the transpose, $\dagger$ is the pseudoinverse.

1. All the columns in $\hat{\boldsymbol{A}}$ are linear combinations of the columns $\boldsymbol{AX}$.
2. $\hat{\boldsymbol{A}}$ is an approximation from below in the sense of psd ordering, i.e., $\mathbf{0} \preceq \hat{\boldsymbol{A}} \preceq \boldsymbol{A}$.

Low-rank approximation

If $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ is psd, we use the Nyström approximation:



$$\hat{\boldsymbol{A}} = \boldsymbol{AX} \quad (\boldsymbol{X}^T \boldsymbol{AX})^{\dagger} \quad \boldsymbol{X}^T \boldsymbol{A}$$

$\boldsymbol{X} \in \mathbb{R}^{N \times k}$ is the test matrix, $\boldsymbol{AX} \in \mathbb{R}^{N \times k}$ is the output matrix, T is the transpose, $\dagger$ is the pseudoinverse.

1. All the columns in $\hat{\boldsymbol{A}}$ are linear combinations of the columns $\boldsymbol{AX}$.
2. $\hat{\boldsymbol{A}}$ is an approximation from below in the sense of psd ordering, i.e., $\mathbf{0} \preceq \hat{\boldsymbol{A}} \preceq \boldsymbol{A}$.
3. $\hat{\boldsymbol{A}}$ gives the minimum residual of any approximation satisfying 1–2.

Low-rank approximation

The optimal test matrix $\boldsymbol{X}$ for the Nyström approximation

$$\hat{\boldsymbol{A}} = \boldsymbol{A}\boldsymbol{X}\left(\boldsymbol{X}^{T}\boldsymbol{A}\boldsymbol{X}\right)^{\dagger}\boldsymbol{X}^{T}\boldsymbol{A},$$

is any matrix whose columns span the dominant eigenvectors.

Low-rank approximation

The optimal test matrix $\boldsymbol{X}$ for the Nyström approximation

$$\hat{\boldsymbol{A}} = \boldsymbol{AX}(\boldsymbol{X}^T\boldsymbol{AX})^\dagger\boldsymbol{X}^T\boldsymbol{A},$$

is any matrix whose columns span the dominant eigenvectors.

We don't know the dominant eigenvectors, so high accuracy is obtained with the randomized block Krylov matrix

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{\Omega} & \boldsymbol{A\Omega} & \cdots & \boldsymbol{A}^{m-1}\boldsymbol{\Omega} \end{bmatrix},$$

where $\boldsymbol{\Omega} \in \mathbb{R}^{N\times k}$ has independent Gaussian entries $\omega_{ij} \sim \mathcal{N}(0,1)$.

Low-rank approximation

The optimal test matrix $\boldsymbol{X}$ for the Nyström approximation

$$\hat{\boldsymbol{A}} = \boldsymbol{A}\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{A}\boldsymbol{X})^{\dagger}\boldsymbol{X}^T\boldsymbol{A},$$

is any matrix whose columns span the dominant eigenvectors.

We don't know the dominant eigenvectors, so high accuracy is obtained with the randomized block Krylov matrix

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{\Omega} & \boldsymbol{A}\boldsymbol{\Omega} & \cdots & \boldsymbol{A}^{m-1}\boldsymbol{\Omega} \end{bmatrix},$$

where $\boldsymbol{\Omega} \in \mathbb{R}^{N \times k}$ has independent Gaussian entries $\omega_{ij} \sim \mathcal{N}(0, 1)$.

Question. Why do we take random $\boldsymbol{\Omega}$?

Low-rank approximation

The optimal test matrix $\boldsymbol{X}$ for the Nyström approximation

$$\hat{\boldsymbol{A}} = \boldsymbol{A}\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{A}\boldsymbol{X})^{\dagger}\boldsymbol{X}^T\boldsymbol{A},$$

is any matrix whose columns span the dominant eigenvectors.

We don't know the dominant eigenvectors, so high accuracy is obtained with the randomized block Krylov matrix

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{\Omega} & \boldsymbol{A}\boldsymbol{\Omega} & \cdots & \boldsymbol{A}^{m-1}\boldsymbol{\Omega} \end{bmatrix},$$

where $\boldsymbol{\Omega} \in \mathbb{R}^{N \times k}$ has independent Gaussian entries $\omega_{ij} \sim \mathcal{N}(0,1)$.

Question. Why do we take random $\boldsymbol{\Omega}$?

Answer. Gaussian vectors automatically find low-rank structure when it exists.

Low-rank approximation

The optimal test matrix $\boldsymbol{X}$ for the Nyström approximation

$$\hat{\boldsymbol{A}} = \boldsymbol{A}\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{A}\boldsymbol{X})^{\dagger}\boldsymbol{X}^T\boldsymbol{A},$$

is any matrix whose columns span the dominant eigenvectors.

We don't know the dominant eigenvectors, so high accuracy is obtained with the randomized block Krylov matrix

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{\Omega} & \boldsymbol{A}\boldsymbol{\Omega} & \cdots & \boldsymbol{A}^{m-1}\boldsymbol{\Omega} \end{bmatrix},$$

where $\boldsymbol{\Omega} \in \mathbb{R}^{N \times k}$ has independent Gaussian entries $\omega_{ij} \sim \mathcal{N}(0,1)$.

Question. Why do we take random $\boldsymbol{\Omega}$?

Answer. Gaussian vectors automatically find low-rank structure when it exists.

* One Gaussian vector might miss the top eigenvectors, but many Gaussian vectors multiplied several times by $\boldsymbol{A}$ are exponentially unlikely to miss the top eigenvectors.

Low-rank approximation

The optimal test matrix $\boldsymbol{X}$ for the Nyström approximation

$$\hat{\boldsymbol{A}} = \boldsymbol{A}\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{A}\boldsymbol{X})^{\dagger}\boldsymbol{X}^T\boldsymbol{A},$$

is any matrix whose columns span the dominant eigenvectors.

We don't know the dominant eigenvectors, so high accuracy is obtained with the randomized block Krylov matrix

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{\Omega} & \boldsymbol{A}\boldsymbol{\Omega} & \cdots & \boldsymbol{A}^{m-1}\boldsymbol{\Omega} \end{bmatrix},$$

where $\boldsymbol{\Omega} \in \mathbb{R}^{N\times k}$ has independent Gaussian entries $\omega_{ij} \sim \mathcal{N}(0,1)$.

Question. Why do we take random $\boldsymbol{\Omega}$?

Answer. Gaussian vectors automatically find low-rank structure when it exists.

* One Gaussian vector might miss the top eigenvectors, but many Gaussian vectors multiplied several times by $\boldsymbol{A}$ are exponentially unlikely to miss the top eigenvectors.

* Single-vector Krylov with $\boldsymbol{\Omega} = \begin{bmatrix}\boldsymbol{\omega}_1\end{bmatrix}$ is old, but using many random vectors $\boldsymbol{\Omega} = \begin{bmatrix}\boldsymbol{\omega}_1 & \cdots & \boldsymbol{\omega}_k\end{bmatrix}$ is modern.

Low-rank approximation

The optimal test matrix $\boldsymbol{X}$ for the Nyström approximation

$$\hat{\boldsymbol{A}} = \boldsymbol{A}\boldsymbol{X}\left(\boldsymbol{X}^T\boldsymbol{A}\boldsymbol{X}\right)^{\dagger}\boldsymbol{X}^T\boldsymbol{A},$$

is any matrix whose columns span the dominant eigenvectors.

We don't know the dominant eigenvectors, so high accuracy is obtained with the randomized block Krylov matrix

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{\Omega} & \boldsymbol{A}\boldsymbol{\Omega} & \cdots & \boldsymbol{A}^{m-1}\boldsymbol{\Omega} \end{bmatrix},$$

where $\boldsymbol{\Omega} \in \mathbb{R}^{N \times k}$ has independent Gaussian entries $\omega_{ij} \sim \mathcal{N}(0,1)$.

Question. Why do we take random $\boldsymbol{\Omega}$?

Answer. Gaussian vectors automatically find low-rank structure when it exists.

- One Gaussian vector might miss the top eigenvectors, but many Gaussian vectors multiplied several times by $\boldsymbol{A}$ are exponentially unlikely to miss the top eigenvectors.
- Single-vector Krylov with $\boldsymbol{\Omega} = \begin{bmatrix} \boldsymbol{\omega}_1 \end{bmatrix}$ is old, but using many random vectors $\boldsymbol{\Omega} = \begin{bmatrix} \boldsymbol{\omega}_1 & \cdots & \boldsymbol{\omega}_k \end{bmatrix}$ is modern.
- We offer an efficient implementation and theoretical guarantees in Tropp & Webber, 2023.

Description of RBKI

Randomized block Krylov iteration (Tropp & Webber, 2023)

1. Generate random Gaussian $\boldsymbol{\Omega} \in \mathbb{R}^{N \times k}$ and set $\boldsymbol{Y}_0 = \boldsymbol{\Omega}$.
2. For $i = 0, \ldots, m-1$:
   a) Compute an orthonormal matrix $\boldsymbol{Q}_i$ with
   $$\operatorname{range}\begin{bmatrix} \boldsymbol{Q}_0 & \cdots & \boldsymbol{Q}_i \end{bmatrix} = \operatorname{range}\begin{bmatrix} \boldsymbol{Y}_0 & \cdots & \boldsymbol{Y}_i \end{bmatrix}.$$
   b) Set $\boldsymbol{Y}_{i+1} = \boldsymbol{A}\boldsymbol{Q}_i$.
3. Compute the Nyström approximation $\hat{\boldsymbol{A}} = \boldsymbol{F}\boldsymbol{F}^T$ using $\begin{bmatrix} \boldsymbol{Q}_0 & \cdots & \boldsymbol{Q}_{m-1} \end{bmatrix}$ and $\begin{bmatrix} \boldsymbol{Y}_1 & \cdots & \boldsymbol{Y}_m \end{bmatrix}$.

Numerically stable code

Generates the Nyström approximation for $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{\Omega} & \boldsymbol{A}\boldsymbol{\Omega} & \cdots & \boldsymbol{A}^{m-1}\boldsymbol{\Omega} \end{bmatrix}$.

2× faster than original RBKI (Rokhlin, Szlam, & Tygert, 2010).

When $\boldsymbol{A}$ is large and dense, the bottleneck is m multiplications with $\boldsymbol{A}$, requiring $\mathcal{O}(N^2 km)$ operations.

Demonstration of RBKI

Example

Consider the following matrices

$$\boldsymbol{A} = \operatorname{diag}(1, e^{-0.1}, \dots, e^{-9999.9})$$
$$\boldsymbol{B} = \boldsymbol{A} + \operatorname{diag}(0.1, 0.099999, \dots, 0.000001).$$

The top eigenvalues are similar, but $\boldsymbol{B}$ has slower eigenvalue decay than $\boldsymbol{A}$.

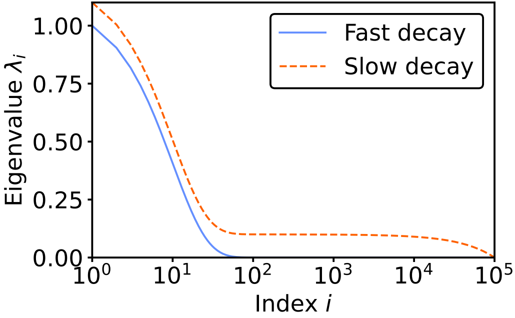


Figure: Eigenvalues decay fast or slow.

Demonstration of RBKI

$$A = \begin{bmatrix} 1.000 & 0.000 & 0.000 & 0.000 & \\ 0.000 & 0.905 & 0.000 & 0.000 & \\ 0.000 & 0.000 & 0.819 & 0.000 & \\ 0.000 & 0.000 & 0.000 & 0.741 & \\ & & & & \ddots \end{bmatrix} \qquad B = \begin{bmatrix} 1.100 & 0.000 & 0.000 & 0.000 & \\ 0.000 & 1.005 & 0.000 & 0.000 & \\ 0.000 & 0.000 & 0.919 & 0.000 & \\ 0.000 & 0.000 & 0.000 & 0.841 & \\ & & & & \ddots \end{bmatrix}$$

Demonstration of RBKI

$$\boldsymbol{A} = \begin{bmatrix} 1.000 & 0.000 & 0.000 & 0.000 & \\ 0.000 & 0.905 & 0.000 & 0.000 & \\ 0.000 & 0.000 & 0.819 & 0.000 & \\ 0.000 & 0.000 & 0.000 & 0.741 & \\ & & & & \ddots \end{bmatrix} \qquad \boldsymbol{B} = \begin{bmatrix} 1.100 & 0.000 & 0.000 & 0.000 & \\ 0.000 & 1.005 & 0.000 & 0.000 & \\ 0.000 & 0.000 & 0.919 & 0.000 & \\ 0.000 & 0.000 & 0.000 & 0.841 & \\ & & & & \ddots \end{bmatrix}$$

RBKI with block size of $k = 100$ Gaussian vectors, depth of $m = 1$ multiplication $\Longrightarrow$

$$\hat{\boldsymbol{A}} = \begin{bmatrix} 1.000 & 0.000 & 0.000 & 0.000 & \\ 0.000 & 0.905 & 0.000 & 0.000 & \\ 0.000 & 0.000 & 0.819 & 0.000 & \\ 0.000 & 0.000 & 0.000 & 0.741 & \\ & & & & \ddots \end{bmatrix} \qquad \hat{\boldsymbol{B}} = \begin{bmatrix} 0.024 & -0.001 & -0.001 & -0.004 & \\ -0.001 & 0.016 & -0.003 & 0.001 & \\ -0.001 & -0.003 & 0.017 & -0.001 & \\ -0.004 & 0.001 & -0.001 & 0.015 & \\ & & & & \ddots \end{bmatrix}$$

Demonstration of RBKI

$$\boldsymbol{A} = \begin{bmatrix} 1.000 & 0.000 & 0.000 & 0.000 & \\ 0.000 & 0.905 & 0.000 & 0.000 & \\ 0.000 & 0.000 & 0.819 & 0.000 & \\ 0.000 & 0.000 & 0.000 & 0.741 & \\ & & & & \ddots \end{bmatrix} \qquad \boldsymbol{B} = \begin{bmatrix} 1.100 & 0.000 & 0.000 & 0.000 & \\ 0.000 & 1.005 & 0.000 & 0.000 & \\ 0.000 & 0.000 & 0.919 & 0.000 & \\ 0.000 & 0.000 & 0.000 & 0.841 & \\ & & & & \ddots \end{bmatrix}$$

RBKI with block size of $k = 100$ Gaussian vectors, depth of $m = 2$ multiplications $\Longrightarrow$

$$\hat{\boldsymbol{A}} = \begin{bmatrix} 1.000 & 0.000 & 0.000 & 0.000 & \\ 0.000 & 0.905 & 0.000 & 0.000 & \\ 0.000 & 0.000 & 0.819 & 0.000 & \\ 0.000 & 0.000 & 0.000 & 0.741 & \\ & & & & \ddots \end{bmatrix} \qquad \hat{\boldsymbol{B}} = \begin{bmatrix} 1.072 & -0.002 & -0.002 & -0.007 & \\ -0.002 & 0.960 & -0.005 & 0.001 & \\ -0.002 & -0.005 & 0.880 & -0.001 & \\ -0.007 & 0.001 & -0.002 & 0.796 & \\ & & & & \ddots \end{bmatrix}$$

Demonstration of RBKI

$$\boldsymbol{A} = \begin{bmatrix} 1.000 & 0.000 & 0.000 & 0.000 & \\ 0.000 & 0.905 & 0.000 & 0.000 & \\ 0.000 & 0.000 & 0.819 & 0.000 & \\ 0.000 & 0.000 & 0.000 & 0.741 & \\ & & & & \ddots \end{bmatrix} \qquad \boldsymbol{B} = \begin{bmatrix} 1.100 & 0.000 & 0.000 & 0.000 & \\ 0.000 & 1.005 & 0.000 & 0.000 & \\ 0.000 & 0.000 & 0.919 & 0.000 & \\ 0.000 & 0.000 & 0.000 & 0.841 & \\ & & & & \ddots \end{bmatrix}$$

RBKI with block size of $k = 100$ Gaussian vectors, depth of $m = 3$ multiplications $\Longrightarrow$

$$\hat{\boldsymbol{A}} = \begin{bmatrix} 1.000 & 0.000 & 0.000 & 0.000 & \\ 0.000 & 0.905 & 0.000 & 0.000 & \\ 0.000 & 0.000 & 0.819 & 0.000 & \\ 0.000 & 0.000 & 0.000 & 0.741 & \\ & & & & \ddots \end{bmatrix} \qquad \hat{\boldsymbol{B}} = \begin{bmatrix} 1.100 & 0.000 & 0.000 & 0.000 & \\ 0.000 & 1.005 & 0.000 & 0.000 & \\ 0.000 & 0.000 & 0.919 & 0.000 & \\ 0.000 & 0.000 & 0.000 & 0.841 & \\ & & & & \ddots \end{bmatrix}$$

Demonstration of RBKI

$$A = \begin{bmatrix} 1.000 & 0.000 & 0.000 & 0.000 & \\ 0.000 & 0.905 & 0.000 & 0.000 & \\ 0.000 & 0.000 & 0.819 & 0.000 & \\ 0.000 & 0.000 & 0.000 & 0.741 & \\ & & & & \ddots \end{bmatrix} \qquad B = \begin{bmatrix} 1.100 & 0.000 & 0.000 & 0.000 & \\ 0.000 & 1.005 & 0.000 & 0.000 & \\ 0.000 & 0.000 & 0.919 & 0.000 & \\ 0.000 & 0.000 & 0.000 & 0.841 & \\ & & & & \ddots \end{bmatrix}$$

RBKI with block size of $k = 100$ Gaussian vectors, depth of $m = 3$ multiplications $\Longrightarrow$

$$\hat{A} = \begin{bmatrix} 1.000 & 0.000 & 0.000 & 0.000 & \\ 0.000 & 0.905 & 0.000 & 0.000 & \\ 0.000 & 0.000 & 0.819 & 0.000 & \\ 0.000 & 0.000 & 0.000 & 0.741 & \\ & & & & \ddots \end{bmatrix} \qquad \hat{B} = \begin{bmatrix} 1.100 & 0.000 & 0.000 & 0.000 & \\ 0.000 & 1.005 & 0.000 & 0.000 & \\ 0.000 & 0.000 & 0.919 & 0.000 & \\ 0.000 & 0.000 & 0.000 & 0.841 & \\ & & & & \ddots \end{bmatrix}$$

- With fast eigenvalue decay, just $m = 1$ matrix multiplication guarantees high accuracy.
- With slow eigenvalue decay, $m = 3$ matrix multiplications is enough.

Practical runtime of RBKI

Question. Why is the small number of multiplications m important?

Practical runtime of RBKI

Question. Why is the small number of multiplications *m* important?

Answer. RBKI runs fast with small *m*, even when the block size *k* is large.

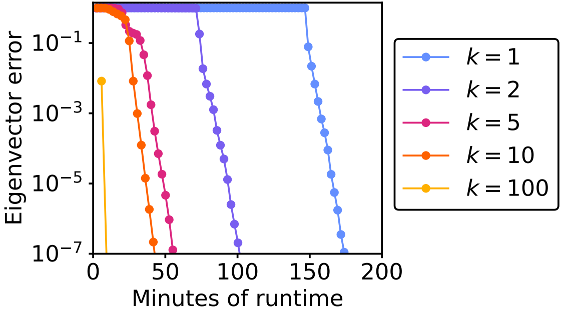
Matrix multiplications with large *k* are optimized with multithreading, caching, and parallelization.

Practical runtime of RBKI

Question. Why is the small number of multiplications m important?

Answer. RBKI runs fast with small m, even when the block size k is large.

Matrix multiplications with large k are optimized with multithreading, caching, and parallelization.

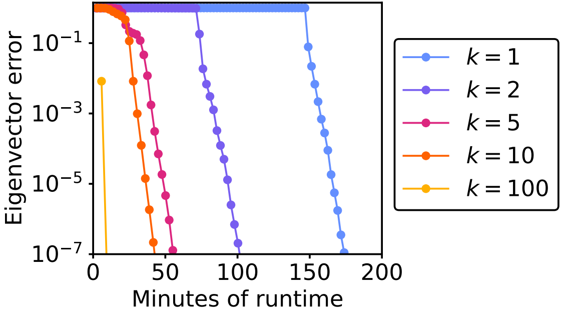


Practical runtime of RBKI

Question. Why is the small number of multiplications m important?

Answer. RBKI runs fast with small m, even when the block size k is large.

Matrix multiplications with large k are optimized with multithreading, caching, and parallelization.



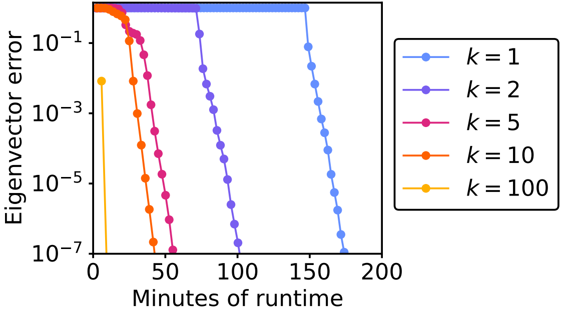
∗ Traditional algorithms use $k = 1$ vector and many matrix multiplications.

Practical runtime of RBKI

Question. Why is the small number of multiplications m important?

Answer. RBKI runs fast with small m, even when the block size k is large.

Matrix multiplications with large k are optimized with multithreading, caching, and parallelization.



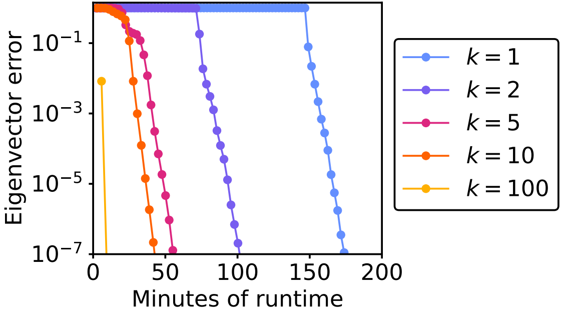
- ∗ Traditional algorithms use $k = 1$ vector and many matrix multiplications.
- ∗ RBKI uses fewer multiplications, leading to 10× speed-ups over `np.linalg.sparse.eigsh` in Python and `eigs` in Matlab.

Practical runtime of RBKI

Question. Why is the small number of multiplications *m* important?

Answer. RBKI runs fast with small *m*, even when the block size *k* is large.

Matrix multiplications with large *k* are optimized with multithreading, caching, and parallelization.



- ∗ Traditional algorithms use $k = 1$ vector and many matrix multiplications.
- ∗ RBKI uses fewer multiplications, leading to 10× speed-ups over `np.linalg.sparse.eigsh` in Python and `eigs` in Matlab.
- ∗ I'm working with M Melnichenko and R Murray to implement RBKI in RandLAPACK, a randomized software package faster than the current standard LAPACK.

Theoretical properties of RBKI

For any psd matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, let $\lfloor \boldsymbol{A} \rfloor_r$ be the best rank-r approximation, which comes from an r-truncated eigendecomposition of $\boldsymbol{A}$.

Theoretical properties of RBKI

For any psd matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, let $\lfloor \boldsymbol{A} \rfloor_r$ be the best rank-r approximation, which comes from an r-truncated eigendecomposition of $\boldsymbol{A}$.

RBKI guarantees for psd matrices (Tropp & Webber, 2023)

For each block size $k \geq 2r + 1$ and number of multiplications $m \geq 2$,

$$\mathbb{E}\|\boldsymbol{A} - \hat{\boldsymbol{A}}\| \leq \exp\left(\left[\frac{\log(4N+4)}{4m-6}\right]^2\right)\|\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r\|.$$

Proof uses "almost convexity" of Chebyshev polynomials and properties of Gaussian matrices (5 bonus slides).

Theoretical properties of RBKI

For any psd matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, let $\lfloor \boldsymbol{A} \rfloor_r$ be the best rank-r approximation, which comes from an r-truncated eigendecomposition of $\boldsymbol{A}$.

RBKI guarantees for psd matrices (Tropp & Webber, 2023)

For each block size $k \geq 2r+1$ and number of multiplications $m \geq 2$,

$$\mathbb{E}\|\boldsymbol{A} - \hat{\boldsymbol{A}}\| \leq \exp\left(\left[\frac{\log(4N+4)}{4m-6}\right]^2\right)\|\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r\|.$$

Proof uses "almost convexity" of Chebyshev polynomials and properties of Gaussian matrices (5 bonus slides).

$\ast$ After just $m = \frac{1}{4}\log(N+1)+2$ multiplications, RBKI guarantees $\mathbb{E}\|\boldsymbol{A} - \hat{\boldsymbol{A}}\| \leq 2\|\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r\|$.

Theoretical properties of RBKI

For any psd matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, let $\lfloor \boldsymbol{A} \rfloor_r$ be the best rank-r approximation, which comes from an r-truncated eigendecomposition of $\boldsymbol{A}$.

RBKI guarantees for psd matrices (Tropp & Webber, 2023)

For each block size $k \geq 2r + 1$ and number of multiplications $m \geq 2$,

$$\mathbb{E} \| \boldsymbol{A} - \hat{\boldsymbol{A}} \| \leq \exp\left(\left[\frac{\log(4N + 4)}{4m - 6} \right]^2 \right) \| \boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r \|.$$

Proof uses "almost convexity" of Chebyshev polynomials and properties of Gaussian matrices (5 bonus slides).

- ∗ After just $m = \frac{1}{4} \log(N + 1) + 2$ multiplications, RBKI guarantees $\mathbb{E} \| \boldsymbol{A} - \hat{\boldsymbol{A}} \| \leq 2 \| \boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r \|$.
- ∗ These are the first bounds that explicitly quantify how small we can take m.

Theoretical properties of RBKI

For any psd matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, let $\lfloor \boldsymbol{A} \rfloor_r$ be the best rank-r approximation, which comes from an r-truncated eigendecomposition of $\boldsymbol{A}$.

RBKI guarantees for psd matrices (Tropp & Webber, 2023)

For each block size $k \geq 2r + 1$ and number of multiplications $m \geq 2$,

$$\mathbb{E}\|\boldsymbol{A} - \hat{\boldsymbol{A}}\| \leq \exp\left(\left[\frac{\log(4N+4)}{4m-6}\right]^2\right)\|\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r\|.$$

Proof uses “almost convexity” of Chebyshev polynomials and properties of Gaussian matrices (5 bonus slides).

- $*$ After just $m = \frac{1}{4}\log(N+1) + 2$ multiplications, RBKI guarantees $\mathbb{E}\|\boldsymbol{A} - \hat{\boldsymbol{A}}\| \leq 2\|\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r\|$.
- $*$ These are the first bounds that explicitly quantify how small we can take m.
- $*$ They are universal bounds that hold for any psd matrix $\boldsymbol{A}$.

Theoretical properties of RBKI

For any psd matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, let $\lfloor \boldsymbol{A} \rfloor_r$ be the best rank-r approximation, which comes from an r-truncated eigendecomposition of $\boldsymbol{A}$.

RBKI guarantees for psd matrices (Tropp & Webber, 2023)

For each block size $k \geq 2r + 1$ and number of multiplications $m \geq 2$,

$$\mathbb{E}\|\boldsymbol{A} - \hat{\boldsymbol{A}}\| \leq \exp\left(\left[\frac{\log(4N + 4)}{4m - 6}\right]^2\right)\|\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r\|.$$

Proof uses "almost convexity" of Chebyshev polynomials and properties of Gaussian matrices (5 bonus slides).

- $*$ After just $m = \frac{1}{4}\log(N + 1) + 2$ multiplications, RBKI guarantees $\mathbb{E}\|\boldsymbol{A} - \hat{\boldsymbol{A}}\| \leq 2\|\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r\|$.
- $*$ These are the first bounds that explicitly quantify how small we can take m.
- $*$ They are universal bounds that hold for any psd matrix $\boldsymbol{A}$.
- $*$ We have also extended RBKI to compute a low–rank approximation for any *rectangular matrix* $\boldsymbol{A} \in \mathbb{R}^{M \times N}$.

Application in genetics

Genetics data sets are large ($\geq 10^4$ single-nucleotide polymorphisms) and noisy:

> *"In large datasets, eigenvalues may be highly significant (reflecting real population structure in the data) but only slightly larger than background noise eigenvalues."*

Application in genetics

Genetics data sets are large ($\geq 10^4$ single-nucleotide polymorphisms) and noisy:

> *"In large datasets, eigenvalues may be highly significant (reflecting real population structure in the data) but only slightly larger than background noise eigenvalues."*

RBKI filters out the noise components, while accurately approximating the signal components.

Application in genetics

Genetics data sets are large ($\geq 10^4$ single-nucleotide polymorphisms) and noisy:

> *"In large datasets, eigenvalues may be highly significant (reflecting real population structure in the data) but only slightly larger than background noise eigenvalues."*

RBKI filters out the noise components, while accurately approximating the signal components.

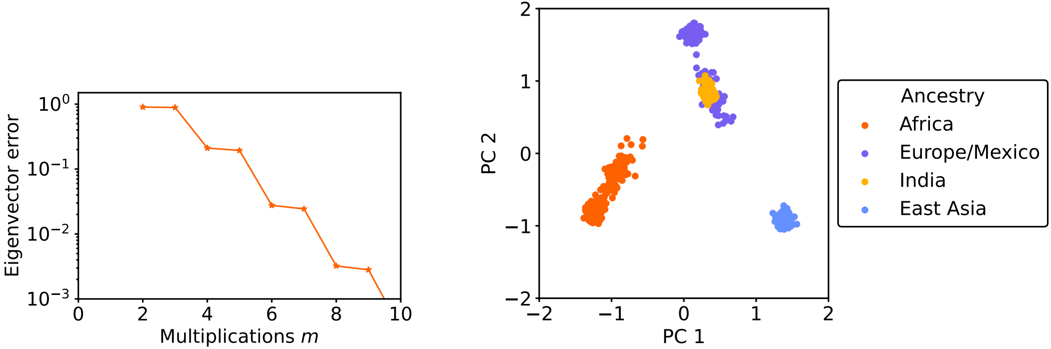


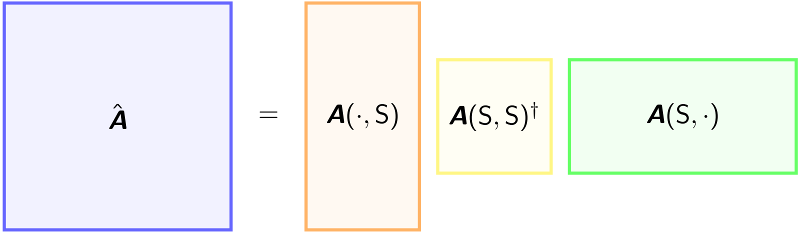
Figure: After $m = 4$ multiplications (left), RBKI reproduces the ideal clustering (right).

Column Nyström approximation

Often, the most efficient approximation of psd $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ is the *column* Nyström approximation:

Column Nyström approximation

Often, the most efficient approximation of psd $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ is the *column* Nyström approximation:

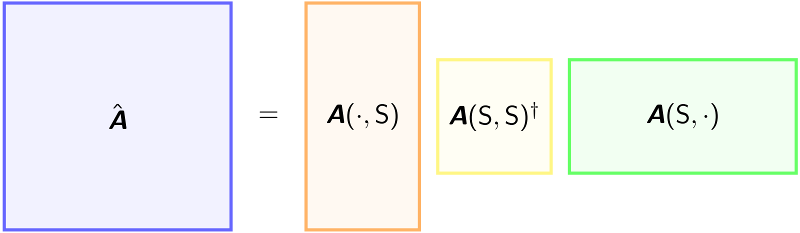


$$\hat{\boldsymbol{A}} = \boldsymbol{A}(\cdot, \mathsf{S}) \, \boldsymbol{A}(\mathsf{S}, \mathsf{S})^{\dagger} \, \boldsymbol{A}(\mathsf{S}, \cdot)$$

$\mathsf{S} = \{s_1, \ldots, s_k\}$ is a set of k column indices, $\dagger$ is the pseudoinverse.

Column Nyström approximation

Often, the most efficient approximation of psd $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ is the *column* Nyström approximation:



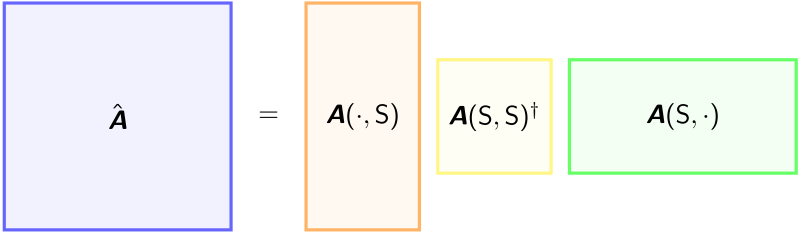
$$\hat{\boldsymbol{A}} = \boldsymbol{A}(\cdot, \mathsf{S}) \; \boldsymbol{A}(\mathsf{S}, \mathsf{S})^{\dagger} \; \boldsymbol{A}(\mathsf{S}, \cdot)$$

$\mathsf{S} = \{s_1, \ldots, s_k\}$ is a set of k column indices, $\dagger$ is the pseudoinverse.

1. $\hat{\boldsymbol{A}}$ is a specialization of the Nyström decomposition.

Column Nyström approximation

Often, the most efficient approximation of psd $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ is the *column* Nyström approximation:



$$\hat{\boldsymbol{A}} = \boldsymbol{A}(\cdot, \mathsf{S}) \; \boldsymbol{A}(\mathsf{S}, \mathsf{S})^{\dagger} \; \boldsymbol{A}(\mathsf{S}, \cdot)$$

$\mathsf{S} = \{s_1, \dots, s_k\}$ is a set of k column indices, $\dagger$ is the pseudoinverse.

1. $\hat{\boldsymbol{A}}$ is a specialization of the Nyström decomposition.
2. $\hat{\boldsymbol{A}}$ perfectly recovers the selected columns $\boldsymbol{A}(\cdot, \mathsf{S})$.

Partial Cholesky approximation

The partial Cholesky decomposition generates the column Nyström approximation:

Partial Cholesky decomposition

1. Initialize $\boldsymbol{F} = \mathbf{0}_{N \times k}$.
2. For $i = 1, \ldots, k$:
   a) Select a column index $s_i \in \{1, \ldots, N\}$.
   b) Evaluate the s_i column of the target matrix $\boldsymbol{g} = \boldsymbol{A}(\cdot, s_i)$.
   c) Subtract the s_i column of the current Nyström approximation $\boldsymbol{g} = \boldsymbol{g} - (\boldsymbol{F}\boldsymbol{F}^T)(\cdot, s_i)$.
   d) Rescale the column $\boldsymbol{F}(\cdot, i) = \boldsymbol{g} / \sqrt{\boldsymbol{g}(s_i)}$.
3. Return the Nyström approximation $\hat{\boldsymbol{A}} = \boldsymbol{F}\boldsymbol{F}^T$.

Partial Cholesky approximation

The partial Cholesky decomposition generates the column Nyström approximation:

Partial Cholesky decomposition

```
1. Initialize $\boldsymbol{F} = \mathbf{0}_{N \times k}$.
2. For $i = 1, \ldots, k$:
    a) Select a column index $s_i \in \{1, \ldots, N\}$.
    b) Evaluate the $s_i$ column of the target matrix $\boldsymbol{g} = \boldsymbol{A}(\cdot, s_i)$.
    c) Subtract the $s_i$ column of the current Nyström approximation $\boldsymbol{g} = \boldsymbol{g} - (\boldsymbol{F}\boldsymbol{F}^T)(\cdot, s_i)$.
    d) Rescale the column $\boldsymbol{F}(\cdot, i) = \boldsymbol{g} / \sqrt{\boldsymbol{g}(s_i)}$.
3. Return the Nyström approximation $\hat{\boldsymbol{A}} = \boldsymbol{F}\boldsymbol{F}^T$.
```

This is like the standard Cholesky decomposition, but stop after k steps.

Partial Cholesky approximation

The partial Cholesky decomposition generates the column Nyström approximation:

Partial Cholesky decomposition

1. Initialize $\boldsymbol{F} = \mathbf{0}_{N \times k}$.
2. For $i = 1, \ldots, k$:
   a) Select a column index $s_i \in \{1, \ldots, N\}$.
   b) Evaluate the s_i column of the target matrix $\boldsymbol{g} = \boldsymbol{A}(\cdot, s_i)$.
   c) Subtract the s_i column of the current Nyström approximation $\boldsymbol{g} = \boldsymbol{g} - (\boldsymbol{F}\boldsymbol{F}^T)(\cdot, s_i)$.
   d) Rescale the column $\boldsymbol{F}(\cdot, i) = \boldsymbol{g}/\sqrt{\boldsymbol{g}(s_i)}$.
3. Return the Nyström approximation $\hat{\boldsymbol{A}} = \boldsymbol{F}\boldsymbol{F}^T$.

This is like the standard Cholesky decomposition, but stop after k steps.

Only requires evaluating the k selected columns and taking linear combinations, hence $\mathcal{O}(Nk)$ memory and $\mathcal{O}(Nk^2)$ operations.

Partial Cholesky approximation

The partial Cholesky decomposition generates the column Nyström approximation:

Partial Cholesky decomposition

```
1. Initialize $\boldsymbol{F} = \mathbf{0}_{N \times k}$.
2. For $i = 1, \dots, k$:
    a) Select a column index $s_i \in \{1, \dots, N\}$.
    b) Evaluate the $s_i$ column of the target matrix $\boldsymbol{g} = \boldsymbol{A}(\cdot, s_i)$.
    c) Subtract the $s_i$ column of the current Nyström approximation $\boldsymbol{g} = \boldsymbol{g} - (\boldsymbol{F}\boldsymbol{F}^T)(\cdot, s_i)$.
    d) Rescale the column $\boldsymbol{F}(\cdot, i) = \boldsymbol{g} / \sqrt{\boldsymbol{g}(s_i)}$.
3. Return the Nyström approximation $\hat{\boldsymbol{A}} = \boldsymbol{F}\boldsymbol{F}^T$.
```

This is like the standard Cholesky decomposition, but stop after k steps.

Only requires evaluating the k selected columns and taking linear combinations, hence $\mathcal{O}(Nk)$ memory and $\mathcal{O}(Nk^2)$ operations.

Let's apply partial Cholesky to an example and consider the best column indices $s_1, \dots, s_k$.

Example of partial Cholesky decomposition

Example. For data points $\boldsymbol{x}_1, \dots, \boldsymbol{x}_6$, define $\boldsymbol{A} \in \mathbb{R}^{6\times 6}$ with entries $a_{ij} = \exp\left(-\frac{1}{2}\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2\right)$.

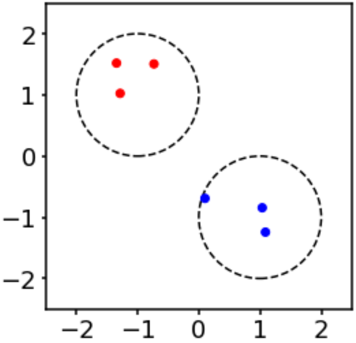


Figure: Two clusters of data points

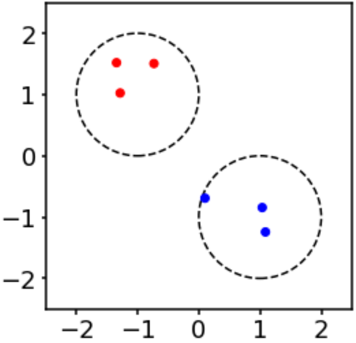
$\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$\boldsymbol{x}_3$	$\boldsymbol{x}_4$	$\boldsymbol{x}_5$	$\boldsymbol{x}_6$
−1.34	−1.28	−0.73	0.10	1.04	1.09
1.52	1.02	1.51	−0.69	−0.84	−1.24

$$\boldsymbol{A} = \left[\begin{array}{ccc|ccc} 1.00 & 0.88 & 0.83 & 0.03 & 0.00 & 0.00 \\ 0.88 & 1.00 & 0.76 & 0.09 & 0.01 & 0.00 \\ 0.83 & 0.76 & 1.00 & 0.06 & 0.01 & 0.00 \\ \hline 0.03 & 0.09 & 0.06 & 1.00 & 0.64 & 0.53 \\ 0.00 & 0.01 & 0.01 & 0.64 & 1.00 & 0.92 \\ 0.00 & 0.00 & 0.00 & 0.53 & 0.92 & 1.00 \end{array}\right]$$

Column $\boldsymbol{A}(\cdot, i)$ measures the similarity of $\boldsymbol{x}_i$ to the other data points.

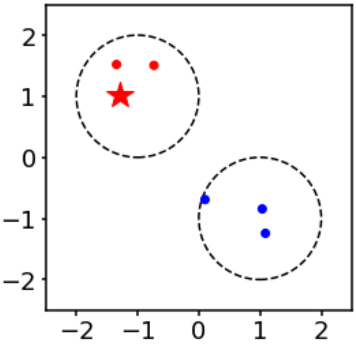
Example of partial Cholesky decomposition

Example. Let's apply the partial Cholesky decomposition and track the diagonal of the residual.



$$\boldsymbol{A}^{(0)} = \left[\begin{array}{ccc|ccc} 1.00 & 0.88 & 0.83 & 0.03 & 0.00 & 0.00 \\ 0.88 & 1.00 & 0.76 & 0.09 & 0.01 & 0.00 \\ 0.83 & 0.76 & 1.00 & 0.06 & 0.01 & 0.00 \\ \hline 0.03 & 0.09 & 0.06 & 1.00 & 0.64 & 0.53 \\ 0.00 & 0.01 & 0.01 & 0.64 & 1.00 & 0.92 \\ 0.00 & 0.00 & 0.00 & 0.53 & 0.92 & 1.00 \end{array}\right]$$

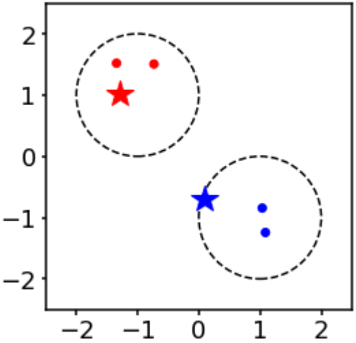
Example of partial Cholesky decomposition



$$\hat{\boldsymbol{A}}^{(1)} = \begin{bmatrix}0.88\\1.00\\0.76\\\hline 0.09\\0.01\\0.00\end{bmatrix}\begin{bmatrix}1.00\end{bmatrix}^{-1}\begin{bmatrix}0.88\\1.00\\0.76\\\hline 0.09\\0.01\\0.00\end{bmatrix}^{T}.$$

$$\boldsymbol{A}^{(1)} = \left[\begin{array}{cc|ccc} 0.22 & 0.15 & -0.05 & -0.01 & 0.00\\ 0.15 & 0.41 & 0.00 & 0.00 & 0.00\\ \hline -0.05 & 0.00 & 0.99 & 0.64 & 0.53\\ -0.01 & 0.00 & 0.64 & 1.00 & 0.92\\ 0.00 & 0.00 & 0.53 & 0.92 & 1.00\end{array}\right].$$

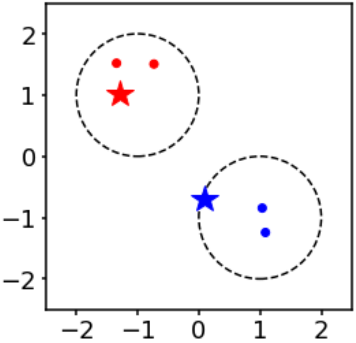
Example of partial Cholesky decomposition



$$\hat{\boldsymbol{A}}^{(2)} = \left[\begin{array}{c|c} 0.88 & -0.05 \\ 1.00 & 0.00 \\ 0.76 & 0.00 \\ \hline 0.09 & 0.99 \\ 0.01 & 0.64 \\ 0.00 & 0.53 \end{array}\right] \left[\begin{array}{c|c} 1.00 & \\ \hline & 0.99 \end{array}\right]^{-1} \left[\begin{array}{c|c} 0.88 & -0.05 \\ 1.00 & 0.00 \\ 0.76 & 0.00 \\ \hline 0.09 & 0.99 \\ 0.01 & 0.64 \\ 0.00 & 0.53 \end{array}\right]^{T}$$

$$\boldsymbol{A}^{(2)} = \left[\begin{array}{cc|cc} 0.22 & 0.15 & 0.02 & 0.02 \\ 0.15 & 0.41 & 0.01 & 0.00 \\ \hline 0.02 & 0.01 & 0.59 & 0.58 \\ 0.02 & 0.00 & 0.58 & 0.72 \end{array}\right].$$

Example of partial Cholesky decomposition



$$\hat{\boldsymbol{A}}^{(2)} = \left[\begin{array}{c|c} 0.88 & -0.05 \\ 1.00 & 0.00 \\ 0.76 & 0.00 \\ \hline 0.09 & 0.99 \\ 0.01 & 0.64 \\ 0.00 & 0.53 \end{array}\right] \left[\begin{array}{c|c} 1.00 & \\ \hline & 0.99 \end{array}\right]^{-1} \left[\begin{array}{c|c} 0.88 & -0.05 \\ 1.00 & 0.00 \\ 0.76 & 0.00 \\ \hline 0.09 & 0.99 \\ 0.01 & 0.64 \\ 0.00 & 0.53 \end{array}\right]^{T}$$

$$\boldsymbol{A}^{(2)} = \left[\begin{array}{cc|cc} 0.22 & 0.15 & 0.02 & 0.02 \\ 0.15 & 0.41 & 0.01 & 0.00 \\ \hline 0.02 & 0.01 & 0.59 & 0.58 \\ 0.02 & 0.00 & 0.58 & 0.72 \end{array}\right].$$

Just two columns typically give an approximation with a small residual.

Choosing the column indices

Question. How can we choose the best column indices?

Choosing the column indices

Question. How can we choose the best column indices?

Answer. For any psd matrix, the diagonal entries are important.

Choosing the column indices

Question. How can we choose the best column indices?

Answer. For any psd matrix, the diagonal entries are important.

– The diagonal entries are nonnegative, and they bound the off-diagonal entries according to

$$|a_{ij}| \le \sqrt{a_{ii}a_{jj}}.$$

Choosing the column indices

Question. How can we choose the best column indices?

Answer. For any psd matrix, the diagonal entries are important.

- The diagonal entries are nonnegative, and they bound the off-diagonal entries according to

$$|a_{ij}| \leq \sqrt{a_{ii} a_{jj}}.$$

- Each large diagonal entry a_{ii} shows that column i might contain large-magnitude elements — we want to include large-magnitude elements in our approximation.

Choosing the column indices

Question. How can we choose the best column indices?

Answer. For any psd matrix, the diagonal entries are important.

- The diagonal entries are nonnegative, and they bound the off-diagonal entries according to

$$|a_{ij}| \leq \sqrt{a_{ii} a_{jj}}.$$

- Each large diagonal entry a_{ii} shows that column i might contain large-magnitude elements — we want to include large-magnitude elements in our approximation.

Follow-up question. How do we use the large diagonal elements?

Choosing the column indices

Question. How can we choose the best column indices?

Answer. For any psd matrix, the diagonal entries are important.

- The diagonal entries are nonnegative, and they bound the off-diagonal entries according to

$$|a_{ij}| \leq \sqrt{a_{ii}a_{jj}}.$$

- Each large diagonal entry a_{ii} shows that column i might contain large-magnitude elements — we want to include large-magnitude elements in our approximation.

Follow-up question. How do we use the large diagonal elements?

* In the greedy method, we select each column according to the largest diagonal element in the residual:

$$s_i \in \operatorname{argmax}_j \left(\boldsymbol{A}^{(i-1)}\right)_{jj}.$$

Choosing the column indices

Question. How can we choose the best column indices?

Answer. For any psd matrix, the diagonal entries are important.

- The diagonal entries are nonnegative, and they bound the off-diagonal entries according to

$$|a_{ij}| \leq \sqrt{a_{ii} a_{jj}}.$$

- Each large diagonal entry a_{ii} shows that column i might contain large-magnitude elements — we want to include large-magnitude elements in our approximation.

Follow-up question. How do we use the large diagonal elements?

* In the greedy method, we select each column according to the largest diagonal element in the residual:

$$s_i \in \operatorname{argmax}_j \left(\boldsymbol{A}^{(i-1)}\right)_{jj}.$$

* This leads to bad performance — the greedy method is easily fooled by *outlier* columns.

Choosing the column indices

Question. How can we choose the best column indices?

Answer. For any psd matrix, the diagonal entries are important.

– The diagonal entries are nonnegative, and they bound the off-diagonal entries according to

$$|a_{ij}| \leq \sqrt{a_{ii}a_{jj}}.$$

– Each large diagonal entry a_{ii} shows that column i might contain large-magnitude elements — we want to include large-magnitude elements in our approximation.

Follow-up question. How do we use the large diagonal elements?

* In the greedy method, we select each column according to the largest diagonal element in the residual:

$$s_i \in \operatorname{argmax}_j \left(\boldsymbol{A}^{(i-1)}\right)_{jj}.$$

* This leads to bad performance — the greedy method is easily fooled by *outlier* columns.

* We could ignore the diagonal and sample uniformly at random

$$s_i \sim \text{Unif}\{1, \ldots, N\}.$$

Choosing the column indices

Question. How can we choose the best column indices?

Answer. For any psd matrix, the diagonal entries are important.

– The diagonal entries are nonnegative, and they bound the off-diagonal entries according to

$$|a_{ij}| \leq \sqrt{a_{ii}a_{jj}}.$$

– Each large diagonal entry a_{ii} shows that column i might contain large-magnitude elements — we want to include large-magnitude elements in our approximation.

Follow-up question. How do we use the large diagonal elements?

* In the greedy method, we select each column according to the largest diagonal element in the residual:

$$s_i \in \operatorname{argmax}_j \left(\boldsymbol{A}^{(i-1)}\right)_{jj}.$$

* This leads to bad performance — the greedy method is easily fooled by *outlier* columns.

* We could ignore the diagonal and sample uniformly at random

$$s_i \sim \mathrm{Unif}\{1, \ldots, N\}.$$

* Uniform sampling leads to bad performance — the method focuses on the "typical" columns but ignores atypical clusters.

Description of Randomly pivoted Cholesky

New method. `Randomly pivoted Cholesky` randomly chooses a column index according to the diagonal elements of the residual:

$$\mathbb{P}\{s_i = j\} = \frac{\left(\boldsymbol{A}^{(i-1)}\right)_{jj}}{\operatorname{tr} \boldsymbol{A}^{(i-1)}}, \quad j = 1, 2, \ldots, N.$$

Description of Randomly pivoted Cholesky

New method. Randomly pivoted Cholesky randomly chooses a column index according to the diagonal elements of the residual:

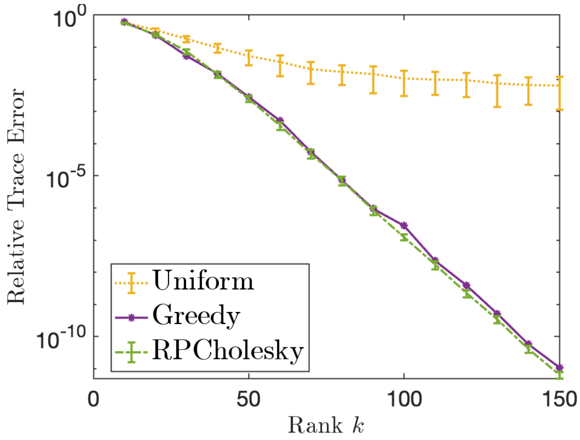
$$\mathbb{P}\{s_i = j\} = \frac{\left(\boldsymbol{A}^{(i-1)}\right)_{jj}}{\operatorname{tr}\boldsymbol{A}^{(i-1)}}, \quad j = 1, 2, \ldots, N.$$

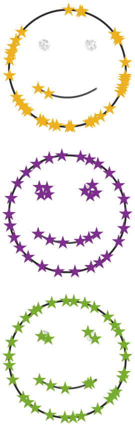
Randomly pivoted Cholesky (Chen, Epperly, Tropp, & Webber, 2022)

```
1. Initialize $\boldsymbol{F} = \boldsymbol{0}_{N\times k}$.
2. Initialize the diagonal of the residual $\boldsymbol{d} = \operatorname{diag}(\boldsymbol{A})$.
3. For $i = 1, \ldots, k$:
    a) Sample a column index $s_i \sim \boldsymbol{d} / \sum_{i=1}^{N} \boldsymbol{d}(i)$.
    b) Evaluate the $s_i$ column of the target matrix $\boldsymbol{g} = \boldsymbol{A}(\cdot, s_i)$.
    c) Subtract the $s_i$ column of the current approximation $\boldsymbol{g} = \boldsymbol{g} - (\boldsymbol{F}\boldsymbol{F}^T)(\cdot, s_i)$.
    d) Rescale the column $\boldsymbol{F}(\cdot, i) = \boldsymbol{g} / \sqrt{\boldsymbol{g}(s_i)}$.
    e) Update the diagonal of the residual matrix $\boldsymbol{d} = \boldsymbol{d} - |\boldsymbol{F}(\cdot, i)|^2$.
4. Return the approximation $\hat{\boldsymbol{A}} = \boldsymbol{F}\boldsymbol{F}^T$.
```

Performance of RPCholesky

In comparisons, RPCholesky outperforms uniform sampling and greedy sampling.





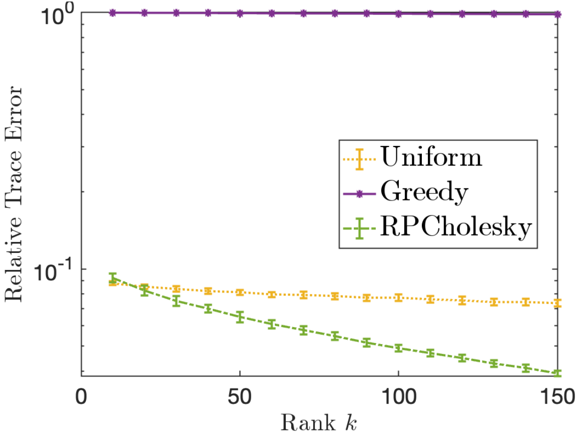
Uniform

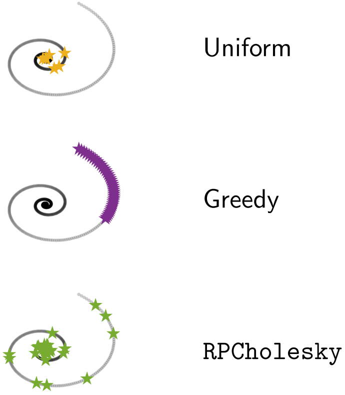
Greedy

RPCholesky

Performance of RPCholesky

In comparisons, RPCholesky outperforms uniform sampling and greedy sampling.





Theoretical properties of RPCholesky

For any psd matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, let $\lfloor \boldsymbol{A} \rfloor_r$ be the best rank-r approximation, which comes from an r-truncated eigendecomposition of $\boldsymbol{A}$.

Theoretical properties of RPCholesky

For any psd matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, let $\lfloor \boldsymbol{A} \rfloor_r$ be the best rank-r approximation, which comes from an r-truncated eigendecomposition of $\boldsymbol{A}$.

Math question. How many columns k are needed to guarantee an (r, ε)-approximation

$$\mathbb{E}\, \mathrm{tr}(\boldsymbol{A} - \hat{\boldsymbol{A}}) \le (1 + \varepsilon)\, \mathrm{tr}(\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r)$$

for every $N \times N$ input matrix?

Theoretical properties of RPCholesky

For any psd matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, let $\lfloor \boldsymbol{A} \rfloor_r$ be the best rank-r approximation, which comes from an r-truncated eigendecomposition of $\boldsymbol{A}$.

Math question. How many columns k are needed to guarantee an (r, ε)-approximation

$$\mathbb{E}\,\mathrm{tr}(\boldsymbol{A} - \hat{\boldsymbol{A}}) \le (1 + \varepsilon)\,\mathrm{tr}(\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r)$$

for every $N \times N$ input matrix?

RPCholesky error bound (Chen, Epperly, Tropp & Webber, 2022)

RPCholesky guarantees an (r, ε)-approximation for

$$k \ge \frac{r}{\varepsilon} + r \log\left(\frac{1}{\varepsilon} \cdot \frac{\mathrm{tr}\,\boldsymbol{A}}{\mathrm{tr}(\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r)}\right).$$

Theoretical properties of RPCholesky

For any psd matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, let $\lfloor \boldsymbol{A} \rfloor_r$ be the best rank-r approximation, which comes from an r-truncated eigendecomposition of $\boldsymbol{A}$.

Math question. How many columns k are needed to guarantee an (r, ε)-approximation

$$\mathbb{E}\,\mathrm{tr}(\boldsymbol{A} - \hat{\boldsymbol{A}}) \le (1 + \varepsilon)\,\mathrm{tr}(\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r)$$

for every $N \times N$ input matrix?

RPCholesky error bound
(Chen, Epperly, Tropp & Webber, 2022)

RPCholesky guarantees an (r, ε)-approximation for

$$k \ge \frac{r}{\varepsilon} + r \log\left(\frac{1}{\varepsilon} \cdot \frac{\mathrm{tr}\,\boldsymbol{A}}{\mathrm{tr}(\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r)}\right).$$

∗ When eigenvalues decay fast, $\mathrm{tr}(\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r)$ is small so the error $\mathbb{E}\,\mathrm{tr}(\boldsymbol{A} - \hat{\boldsymbol{A}})$ must be small.

Theoretical properties of RPCholesky

For any psd matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, let $\lfloor \boldsymbol{A} \rfloor_r$ be the best rank-r approximation, which comes from an r-truncated eigendecomposition of $\boldsymbol{A}$.

Math question. How many columns k are needed to guarantee an (r, ε)-approximation

$$\mathbb{E}\,\mathrm{tr}(\boldsymbol{A} - \hat{\boldsymbol{A}}) \le (1+\varepsilon)\mathrm{tr}(\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r)$$

for every $N \times N$ input matrix?

RPCholesky error bound
(Chen, Epperly, Tropp & Webber, 2022)

RPCholesky guarantees an (r, ε)-approximation for

$$k \ge \frac{r}{\varepsilon} + r \log\left(\frac{1}{\varepsilon} \cdot \frac{\mathrm{tr}\boldsymbol{A}}{\mathrm{tr}(\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r)}\right).$$

- ∗ When eigenvalues decay fast, $\mathrm{tr}(\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r)$ is small so the error $\mathbb{E}\,\mathrm{tr}(\boldsymbol{A} - \hat{\boldsymbol{A}})$ must be small.
- ∗ Since the log factor is small, RPCholesky requires a small multiple (≤ 10) of r/ε columns.

Theoretical properties of RPCholesky

For any psd matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, let $\lfloor \boldsymbol{A} \rfloor_r$ be the best rank-r approximation, which comes from an r-truncated eigendecomposition of $\boldsymbol{A}$.

Math question. How many columns k are needed to guarantee an (r, ε)-approximation

$$\mathbb{E}\,\mathrm{tr}(\boldsymbol{A} - \hat{\boldsymbol{A}}) \le (1 + \varepsilon)\,\mathrm{tr}(\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r)$$

for every $N \times N$ input matrix?

RPCholesky error bound (Chen, Epperly, Tropp & Webber, 2022)

RPCholesky guarantees an (r, ε)-approximation for

$$k \ge \frac{r}{\varepsilon} + r \log\left(\frac{1}{\varepsilon} \cdot \frac{\mathrm{tr}\,\boldsymbol{A}}{\mathrm{tr}(\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r)}\right).$$

- When eigenvalues decay fast, $\mathrm{tr}(\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r)$ is small so the error $\mathbb{E}\,\mathrm{tr}(\boldsymbol{A} - \hat{\boldsymbol{A}})$ must be small.
- Since the log factor is small, RPCholesky requires a small multiple ($\lesssim 10$) of r/ε columns.
- The bounds are near-optimal — there is a psd matrix that cannot be approximated in fewer than r/ε columns.

Theoretical properties of RPCholesky

For any psd matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, let $\lfloor \boldsymbol{A} \rfloor_r$ be the best rank-r approximation, which comes from an r-truncated eigendecomposition of $\boldsymbol{A}$.

Math question. How many columns k are needed to guarantee an (r, ε)-approximation

$$\mathbb{E}\operatorname{tr}(\boldsymbol{A} - \hat{\boldsymbol{A}}) \le (1 + \varepsilon)\operatorname{tr}(\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r)$$

for every $N \times N$ input matrix?

RPCholesky error bound (Chen, Epperly, Tropp & Webber, 2022)

RPCholesky guarantees an (r, ε)-approximation for

$$k \ge \frac{r}{\varepsilon} + r \log\left(\frac{1}{\varepsilon} \cdot \frac{\operatorname{tr}\boldsymbol{A}}{\operatorname{tr}(\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r)}\right).$$

- When eigenvalues decay fast, $\operatorname{tr}(\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r)$ is small so the error $\mathbb{E}\operatorname{tr}(\boldsymbol{A} - \hat{\boldsymbol{A}})$ must be small.
- Since the log factor is small, RPCholesky requires a small multiple (≤ 10) of r/ε columns.
- The bounds are near-optimal — there is a psd matrix that cannot be approximated in fewer than r/ε columns.
- Proof uses matrix monotonicity, matrix concavity, and dynamical systems (5 bonus slides).

Application in molecular dynamics

- In biochemistry, we run molecular dynamics simulations to understand protein behavior and design new drugs.

Application in molecular dynamics

- In biochemistry, we run molecular dynamics simulations to understand protein behavior and design new drugs.

- For example, consider alanine dipeptide, $CH_3{-}CO{-}NH{-}C_\alpha HCH_3{-}CO{-}NH{-}CH_3$. After eliminating hydrogens, the data has $N = 250,000$ configurations of 10 atoms.

Application in molecular dynamics

- $\ast$ In biochemistry, we run molecular dynamics simulations to understand protein behavior and design new drugs.
- $\ast$ For example, consider alanine dipeptide, $CH_3{-}CO{-}NH{-}C_\alpha HCH_3{-}CO{-}NH{-}CH_3$. After eliminating hydrogens, the data has $N = 250,000$ configurations of 10 atoms.
- $\ast$ We want to identify *metastable states* which the miniprotein occupies for a long time with rare transitions.

Application in molecular dynamics

- In biochemistry, we run molecular dynamics simulations to understand protein behavior and design new drugs.
- For example, consider alanine dipeptide, $CH_3{-}CO{-}NH{-}C_\alpha HCH_3{-}CO{-}NH{-}CH_3$. After eliminating hydrogens, the data has $N = 250,000$ configurations of 10 atoms.
- We want to identify *metastable states* which the miniprotein occupies for a long time with rare transitions.

Application in molecular dynamics

- In biochemistry, we run molecular dynamics simulations to understand protein behavior and design new drugs.

- For example, consider alanine dipeptide, $CH_3{-}CO{-}NH{-}C_\alpha HCH_3{-}CO{-}NH{-}CH_3$. After eliminating hydrogens, the data has $N = 250,000$ configurations of 10 atoms.

- We want to identify *metastable states* which the miniprotein occupies for a long time with rare transitions.

- Let's introduce a kernel function $\phi : \mathbb{R}^{30} \times \mathbb{R}^{30} \to \mathbb{R}_+$ that quantifies similarity between configurations:

$$\phi(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\Big(-\frac{1}{2\sigma^2}\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2\Big).$$

and identify clusters with high ϕ values within clusters and low ϕ values across clusters (see bonus slides).

Application in molecular dynamics

- ∗ In biochemistry, we run molecular dynamics simulations to understand protein behavior and design new drugs.
- ∗ For example, consider alanine dipeptide, $\mathrm{CH_3{-}CO{-}NH{-}C_\alpha HCH_3{-}CO{-}NH{-}CH_3}$. After eliminating hydrogens, the data has $N = 250,000$ configurations of 10 atoms.
- ∗ We want to identify *metastable states* which the miniprotein occupies for a long time with rare transitions.

– Let's introduce a kernel function $\phi : \mathbb{R}^{30} \times \mathbb{R}^{30} \to \mathbb{R}_+$ that quantifies similarity between configurations:

$$\phi(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\Bigl(-\frac{1}{2\sigma^2}\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2\Bigr).$$

and identify clusters with high ϕ values within clusters and low ϕ values across clusters (see bonus slides).

– The bottleneck is computing the dominant eigenvectors of an $N \times N$ kernel matrix. Even forming and storing this matrix is expensive.

Application in molecular dynamics

- In biochemistry, we run molecular dynamics simulations to understand protein behavior and design new drugs.
- For example, consider alanine dipeptide, $CH_3{-}CO{-}NH{-}C_\alpha HCH_3{-}CO{-}NH{-}CH_3$. After eliminating hydrogens, the data has $N = 250,000$ configurations of 10 atoms.
- We want to identify *metastable states* which the miniprotein occupies for a long time with rare transitions.

- Let's introduce a kernel function $\phi : \mathbb{R}^{30} \times \mathbb{R}^{30} \to \mathbb{R}_+$ that quantifies similarity between configurations:

$$\phi(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{1}{2\sigma^2}\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2\right).$$

  and identify clusters with high ϕ values within clusters and low ϕ values across clusters (see bonus slides).
- The bottleneck is computing the dominant eigenvectors of an $N \times N$ kernel matrix. Even forming and storing this matrix is expensive.
- Traditionally, scientists subsampled their data and ran clustering codes for > 1 day.

Application in molecular dynamics

RPCholesky clusters $250,000$ points in just 10 seconds on a laptop. Near-perfect clustering is obtained with a rank $k = 150$ approximation.

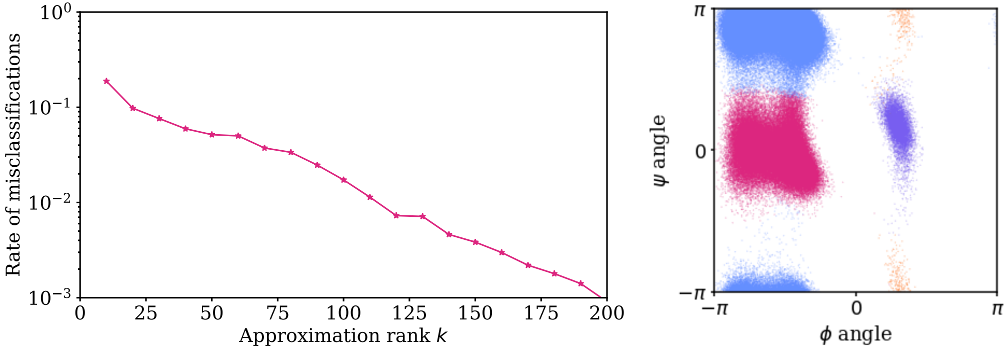


Figure: When we cluster configurations in $\mathbb{R}^{30}$, we find four clusters that are well-aligned with nonlinear functions of the coordinates (ϕ and ψ dihedral angles).

Summary

Question. Why was spectral clustering so slow in the past?

Summary

Question. Why was spectral clustering so slow in the past?

* Many scientists relied on dense eigendecomposition, which costs $\mathcal{O}(N^3)$ operations.

Summary

Question. Why was spectral clustering so slow in the past?

- * Many scientists relied on dense eigendecomposition, which costs $\mathcal{O}(N^3)$ operations.
- * Krylov methods are $10^3\times$ faster (N^2r cost), and the fastest Krylov method is RBKI.

Summary

Question. Why was spectral clustering so slow in the past?

- Many scientists relied on dense eigendecomposition, which costs $\mathcal{O}(N^3)$ operations.
- Krylov methods are $10^3\times$ faster (N^2r cost), and the fastest Krylov method is `RBKI`.
- Column Nyström methods are $10^6\times$ faster than dense eigendecomposition (Nr^2 cost), and the most accurate and robust column Nyström method is `RPCholesky`.

Summary

Question. Why was spectral clustering so slow in the past?

- Many scientists relied on dense eigendecomposition, which costs $\mathcal{O}(N^3)$ operations.
- Krylov methods are $10^3\times$ faster ($N^2 r$ cost), and the fastest Krylov method is `RBKI`.
- Column Nyström methods are $10^6\times$ faster than dense eigendecomposition (Nr^2 cost), and the most accurate and robust column Nyström method is `RPCholesky`.
- As a limitation, `RPCholesky` is only accurate when there is fast eigenvalue decay — it wouldn't work for the noisy genetics data. However, we often expect kernel matrices and neural net matrices to have fast eigenvalue decay.

Summary

Question. Why was spectral clustering so slow in the past?

- Many scientists relied on dense eigendecomposition, which costs $\mathcal{O}(N^3)$ operations.
- Krylov methods are $10^3\times$ faster (N^2r cost), and the fastest Krylov method is RBKI.
- Column Nyström methods are $10^6\times$ faster than dense eigendecomposition (Nr^2 cost), and the most accurate and robust column Nyström method is RPCholesky.
- As a limitation, RPCholesky is only accurate when there is fast eigenvalue decay — it wouldn't work for the noisy genetics data. However, we often expect kernel matrices and neural net matrices to have fast eigenvalue decay.

Main answer. Communication with scientists is essential to make sure they are aware of these advances.

Research vision

Question. How can we deploy randomized methods to speed up scientific computations?

Research vision

Question. How can we deploy randomized methods to speed up scientific computations?

Application area. Kernel methods are machine learning methods that use a positive-definite kernel function $\phi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ to quantify "similarity" between data points, $\phi(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$.

Research vision

Question. How can we deploy randomized methods to speed up scientific computations?

Application area. Kernel methods are machine learning methods that use a positive-definite kernel function $\phi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ to quantify "similarity" between data points, $\phi(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$.

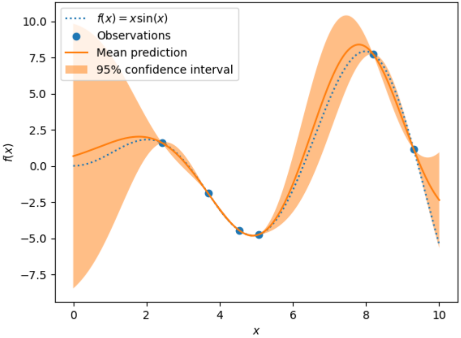


Figure: Used for prediction ("supervised learning")

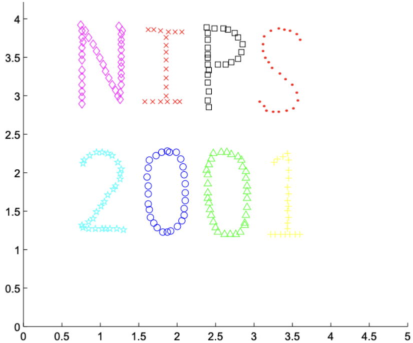


Figure: Used for clustering ("unsupervised learning")

Research vision

Kernel methods perform well for small data sets, after tuning the similarity measure ϕ.

Research vision

Kernel methods perform well for small data sets, after tuning the similarity measure ϕ.

Strong kernel hypothesis. Data-adaptive kernel methods perform as well as neural nets for large data sets ($\geq 10^5$ data points).

Research vision

Kernel methods perform well for small data sets, after tuning the similarity measure ϕ.

Strong kernel hypothesis. Data-adaptive kernel methods perform as well as neural nets for large data sets ($\geq 10^5$ data points).

It is impossible to certify or refute the strong kernel hypothesis, because we lack the computational tools to quickly apply kernel methods to large data sets.

Research vision

Kernel methods perform well for small data sets, after tuning the similarity measure ϕ.

Strong kernel hypothesis. Data-adaptive kernel methods perform as well as neural nets for large data sets ($\geq 10^5$ data points).

It is impossible to certify or refute the strong kernel hypothesis, because we lack the computational tools to quickly apply kernel methods to large data sets.

Research program. Develop randomized algorithms to apply kernel methods at scale and resolve this strong kernel hypothesis.

Research vision

Kernel methods perform well for small data sets, after tuning the similarity measure ϕ.

Strong kernel hypothesis. Data-adaptive kernel methods perform as well as neural nets for large data sets ($\geq 10^5$ data points).

It is impossible to certify or refute the strong kernel hypothesis, because we lack the computational tools to quickly apply kernel methods to large data sets.

Research program. Develop randomized algorithms to apply kernel methods at scale and resolve this strong kernel hypothesis.

Ongoing and future projects:

- Randomly pivoted Cholesky for infinite data sets (with E Epperly & J Tropp)
- Data-adaptive kernels for biochemistry (with D Aristoff & G Simpson).

Research vision

Randomized algorithms have many applications:

Research vision

Randomized algorithms have many applications:

1. Randomized low-rank approximation can accelerate ADMM and Newton's method in optimization.

Research vision

Randomized algorithms have many applications:

1. Randomized low-rank approximation can accelerate ADMM and Newton's method in optimization.

2. Random sparsification methods can solve large-scale Laplacian linear systems.

Research vision

Randomized algorithms have many applications:

1. Randomized low-rank approximation can accelerate ADMM and Newton's method in optimization.
2. Random sparsification methods can solve large-scale Laplacian linear systems.
3. "LoRA: Low-Rank Adaptation of Large Language Models" is used to fine-tune GPT-4. There is an opportunity for randomized low-rank approximation.

Research vision

Randomized algorithms have many applications:

1. Randomized low-rank approximation can accelerate ADMM and Newton's method in optimization.
2. Random sparsification methods can solve large-scale Laplacian linear systems.
3. "LoRA: Low-Rank Adaptation of Large Language Models" is used to fine-tune GPT-4. There is an opportunity for randomized low-rank approximation.

Research vision. Develop and analyze randomized algorithms to address 21st century computing challenges, while mathematically guaranteeing accuracy.

Research vision

Randomized algorithms have many applications:

1. Randomized low-rank approximation can accelerate ADMM and Newton's method in optimization.
2. Random sparsification methods can solve large-scale Laplacian linear systems.
3. "LoRA: Low-Rank Adaptation of Large Language Models" is used to fine-tune GPT-4. There is an opportunity for randomized low-rank approximation.

Research vision. Develop and analyze randomized algorithms to address 21st century computing challenges, while mathematically guaranteeing accuracy.

Ongoing and future projects:

- Fast, randomized algorithms for graph signal processing (with R Lu).
- Randomly sparsified PageRank solvers (with J Weare).

Research vision

Randomized algorithms have many applications:

1. Randomized low-rank approximation can accelerate ADMM and Newton's method in optimization.
2. Random sparsification methods can solve large-scale Laplacian linear systems.
3. "LoRA: Low-Rank Adaptation of Large Language Models" is used to fine-tune GPT-4. There is an opportunity for randomized low-rank approximation.

Research vision. Develop and analyze randomized algorithms to address 21st century computing challenges, while mathematically guaranteeing accuracy.

Ongoing and future projects:

- Fast, randomized algorithms for graph signal processing (with R Lu).
- Randomly sparsified PageRank solvers (with J Weare).

Thank you for your attention! Does anyone have questions?

RPCholesky theorem

Main theorem (Chen, Epperly, Tropp, & Webber, 2022)

Fix a psd matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ and let $\lfloor \boldsymbol{A} \rfloor_r$ denote an optimal rank-r approximation obtained from an r-truncated eigendecomposition.

RPCholesky generates a k-column Nyström approximation s.t.

$$\mathbb{E}\,\mathrm{tr}(\boldsymbol{A} - \hat{\boldsymbol{A}}) \le (1+\varepsilon) \cdot \mathrm{tr}(\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r).$$

as soon as

$$k \ge \frac{r}{\varepsilon} + r \log\left(\frac{1}{\varepsilon} \cdot \frac{\mathrm{tr}\,\boldsymbol{A}}{\mathrm{tr}(\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r)} \right).$$

Proof uses ideas of

- *matrix monotonicity*,
- *matrix concavity*,
- *dynamical systems.*

Let's go!

RPCholesky proof

Recall the recursive definition of RPCholesky:

- Set $\boldsymbol{A}^{(0)} = \boldsymbol{A}$.
- For $i = 1, \ldots, N$, sample $s_i \sim \operatorname{diag} \boldsymbol{A}^{(i-1)}$ and set

$$\boldsymbol{A}^{(i)} = \boldsymbol{A}^{(i-1)} - \frac{\boldsymbol{A}^{(i-1)}(:, s_i)\boldsymbol{A}^{(i-1)}(s_i, :)}{a^{(i-1)}_{s_i s_i}}.$$

Definition

Introduce the *expected residual function*

$$\boldsymbol{\Phi}(\boldsymbol{A}) := \mathbb{E}\left[\boldsymbol{A}^{(1)} \mid \boldsymbol{A}\right] = \boldsymbol{A} - \sum_{s_1=1}^{N} \frac{a_{s_1 s_1}}{\operatorname{tr}\boldsymbol{A}} \frac{\boldsymbol{A}(:, s_1)\boldsymbol{A}(s_1, :)}{a_{s_1 s_1}} = \boldsymbol{A} - \frac{\boldsymbol{A}^2}{\operatorname{tr}\boldsymbol{A}}.$$

Lemma

$\boldsymbol{\Phi}$ is monotone and concave with respect to the psd ordering.

RPCholesky proof

Lemma

The RPCholesky residual satisfies

$$\mathbb{E}\,\operatorname{tr} \boldsymbol{A}^{(k)} \le \operatorname{tr} \boldsymbol{\Phi}^{\circ k}(\boldsymbol{A}).$$

Proof.

Let $\boldsymbol{A}$ be a psd matrix. Using the matrix Jensen's inequality,

$$\mathbb{E}\,\boldsymbol{A}^{(i+1)} = \mathbb{E}\,\mathbb{E}\big[\boldsymbol{A}^{(i+1)}\big|\boldsymbol{A}^{(i)}\big] = \mathbb{E}\,\boldsymbol{\Phi}\big(\boldsymbol{A}^{(i)}\big) \preceq \boldsymbol{\Phi}\big(\mathbb{E}\boldsymbol{A}^{(i)}\big)$$

for $i = 0, 1, \ldots, k-1$. Applying the above display recursively,

$$\mathbb{E}\,\boldsymbol{A}^{(k)} \preceq \boldsymbol{\Phi}\big(\mathbb{E}\,\boldsymbol{A}^{(k-1)}\big) \preceq \boldsymbol{\Phi}\circ\boldsymbol{\Phi}\big(\mathbb{E}\,\boldsymbol{A}^{(k-2)}\big) \preceq \cdots \preceq \boldsymbol{\Phi}^{\circ k}(\boldsymbol{A}).$$

The trace is linear and respects psd ordering, hence

$$\mathbb{E}\,\operatorname{tr} \boldsymbol{A}^{(k)} = \operatorname{tr} \mathbb{E}\,\boldsymbol{A}^{(k)} \le \operatorname{tr} \boldsymbol{\Phi}^{\circ k}(\boldsymbol{A}).$$

□

RPCholesky proof

Proof of main theorem (6 steps):

1. We know $\mathbb{E}\operatorname{tr}\boldsymbol{A}^{(k)} \le \operatorname{tr}\boldsymbol{\Phi}^{\circ k}(\boldsymbol{A})$. The quantity $\operatorname{tr}\boldsymbol{\Phi}^{\circ k}(\boldsymbol{A})$ depends only on the eigenvalues of $\boldsymbol{A}$, so we take $\boldsymbol{A}$ to be diagonal.
2. By concavity, the worst-case matrix is

$$\boldsymbol{A} = \operatorname{diag}\Bigg(\underbrace{\frac{a}{r},\ldots,\frac{a}{r}}_{r \text{ times}},\underbrace{\frac{b}{N-r},\ldots,\frac{b}{N-r}}_{N-r\text{ times}}\Bigg).$$

3. By definition of $\boldsymbol{\Phi}$, we obtain

$$\boldsymbol{\Phi}^{\circ k}(\boldsymbol{A}) = \operatorname{diag}\Bigg(\underbrace{\frac{a^{(k)}}{r},\ldots,\frac{a^{(k)}}{r}}_{r \text{ times}},\underbrace{\frac{b^{(k)}}{N-r},\ldots,\frac{b^{(k)}}{N-r}}_{N-r\text{ times}}\Bigg),$$

where $\left(a^{(i)}\right)_{i=0,1,\ldots}$ and $\left(b^{(i)}\right)_{i=0,1,\ldots}$ are decreasing sequences with explicit formulae.

RPCholesky proof

Proof of main theorem (6 steps):

4. Moreover, we find an explicit upper bound

$$\operatorname{tr} \mathbf{\Phi}^{\circ k}(\boldsymbol{A}) \leq \overline{a}^{(k)} + b^{(0)}, \qquad \overline{a}^{(k)} - \overline{a}^{(k-1)} = \frac{-\left(\overline{a}^{(k-1)}\right)^2}{r\left(\overline{a}^{(k-1)} + b^{(0)}\right)}.$$

5. At each instant $t = 0, 1, 2, \ldots$, the discrete-time process $\overline{a}^{(t)}$ is bounded from above by the continuous-time process $x(t)$ satisfying

$$\frac{\mathrm{d}}{\mathrm{d}t} x(t) = -\frac{x(t)^2}{r\left(x(t) + b^{(0)}\right)} \quad \text{with initial condition } x(0) = a^{(0)}.$$

6. By direct calculation,

$$x(t) \leq \varepsilon b^{(0)} \quad \text{for any} \quad t \geq \frac{r}{\varepsilon} + r \log\left(\frac{1}{\varepsilon} \cdot \frac{\operatorname{tr} \boldsymbol{A}}{\operatorname{tr}(\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r)}\right).$$

RBKI proof

RBKI guarantees for psd matrices (Tropp & Webber, 2023)

For each block size $k \geq 2r+1$ and number of multiplications $m \geq 2$, RBKI satisfies

$$\mathbb{E}\|\boldsymbol{A} - \hat{\boldsymbol{A}}\| \leq \exp\left(\left[\frac{\log(4N+4)}{4m-6} \right]^2 \right) \|\boldsymbol{A} - \lfloor \boldsymbol{A} \rfloor_r\|.$$

Proof uses ideas of

- *Chebyshev polynomials,*
- *convexity and almost convexity,*
- *properties of Gaussian matrices.*

Let's go!

RBKI theorem

Tensions in the proof:

- The main idea is apply a filter $\phi(\boldsymbol{A})$ that increases the top eigenvalues and decreases the bottom eigenvalues.
- We need to ensure $\phi(\boldsymbol{A})\boldsymbol{\Omega}$ lies inside the approximation space, so we use Chebyshev polynomials.
- However, there is a **challenge**. The Chebyshev filtering argument only applies to

  $$\|(\mathbf{I} - \boldsymbol{\Pi}_{\phi(\boldsymbol{A})\boldsymbol{\Omega}})\phi(\boldsymbol{A})\|,$$

  where $\boldsymbol{\Pi}_{\phi(\boldsymbol{A})\boldsymbol{\Omega}}$ is the orthogonal projector onto the range of $\phi(\boldsymbol{A})\boldsymbol{\Omega}$, but the actual error attained by RBKI is

  $$\|(\mathbf{I} - \boldsymbol{\Pi}_{\phi(\boldsymbol{A})\boldsymbol{\Omega}})\boldsymbol{A}\|.$$

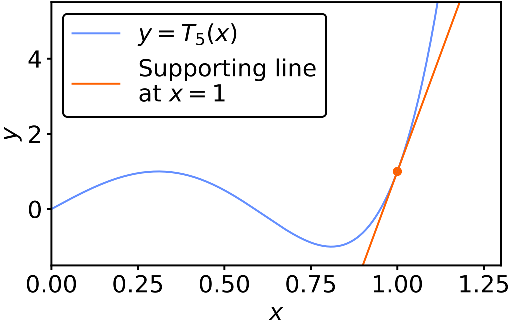
  How can we relate these two quantities?

RBKI theorem

Lemma (Jensen's inequality with "almost" convex functions, Tropp & Webber 2023)

Consider symmetric $\boldsymbol{A} \in \mathbb{R}^{N\times N}$, a random rank-$r$ orthogonal projection $\boldsymbol{Q} \in \mathbb{R}^{N\times N}$, and a function $f : [0,\infty) \to \mathbb{R}$ that has a supporting line at $(x, f(x))$ for $x \geq \sigma_{r+1}(\boldsymbol{A})^2$. Then,

$$f\left(\mathbb{E}\|(\mathbf{I}-\boldsymbol{Q})\boldsymbol{A}\|^2\right) \leq \mathbb{E}\|(\mathbf{I}-\boldsymbol{Q})f(\boldsymbol{A}^2)(\mathbf{I}-\boldsymbol{Q})\|.$$



Chebyshev polynomials $T_i(x)$ are almost convex. They admit supporting lines on the range $x \geq 1$.

RBKI theorem

Proof of main theorem (3 steps):

1. The RBKI approximation satisfies

$$\|\boldsymbol{A} - \hat{\boldsymbol{A}}\| \le \|(\mathbf{I} - \boldsymbol{\Pi}_{[\boldsymbol{A}^{1/2}\boldsymbol{\Omega} \; \cdots \; \boldsymbol{A}^{m-1/2}\boldsymbol{\Omega}]})\boldsymbol{A}\|.$$

2. We apply the majorization

$$\|\boldsymbol{A} - \hat{\boldsymbol{A}}\| \le \|(\mathbf{I} - \boldsymbol{Q})\boldsymbol{A}\|,$$

where

$$\boldsymbol{Q} = \phi(\boldsymbol{A})\boldsymbol{\Omega}\boldsymbol{\Omega}_1^{\dagger},$$

is a rank-r orthogonal projection. It is defined using the modified Chebyshev polynomial

$$\phi(x) = xT_{2m-1}\left(\sqrt{\frac{x}{\lambda_{r+1}(\boldsymbol{A})}}\right)$$

and the $N \times r$ matrix

$$\boldsymbol{\Omega}_1 = \begin{bmatrix} \boldsymbol{v}_1(\boldsymbol{A}) & \cdots & \boldsymbol{v}_r(\boldsymbol{A}) \end{bmatrix}^T \boldsymbol{\Omega}.$$

where $\boldsymbol{v}_i(\boldsymbol{A})$ are the dominant r eigenvectors of $\boldsymbol{A}$.

RBKI theorem

Proof of main theorem (3 steps):

3. Using the "almost convexity" lemma with $f(x) = \phi(\sqrt{x})^2$, calculate

$$\begin{aligned} f\left(\mathbb{E}\|(\mathbf{I}-\boldsymbol{Q})\boldsymbol{A}\|^2\right) &\le \mathbb{E}\|(\mathbf{I}-\boldsymbol{Q})f(\boldsymbol{A}^2)(\mathbf{I}-\boldsymbol{Q})\| \\ &= \mathbb{E}\|\phi(\boldsymbol{A})(\mathbf{I}-\boldsymbol{Q})\|^2 \\ &\le \sigma_{r+1}(\phi(\boldsymbol{A}))^2 + \frac{r}{k-r-1}\sum\nolimits_{i=r+1}^{N} \sigma_i(\phi(\boldsymbol{A}))^2 \\ &\le \sigma_{r+1}(\boldsymbol{A})^2 + \frac{r}{k-r-1}\sum\nolimits_{i=r+1}^{N} \sigma_i(\boldsymbol{A})^2. \end{aligned}$$

The third line uses a standard formula for the moments of an inverse Wishart matrix. The fourth line uses the fact that $\lambda_i(\phi(\boldsymbol{A})) \le \lambda_i(\boldsymbol{A})$ for $i > r$. This gives a stronger version of the result. □

Kernel spectral clustering

Kernel spectral clustering: formulation

Find a low-dimensional embedding $\boldsymbol{V} \in \mathbb{R}^{N \times k}$ of N data points into $\mathbb{R}^k$ that minimizes distortion

$$\frac{1}{2} \sum_{i,j=1}^{N} \phi(\boldsymbol{x}_i, \boldsymbol{x}_j) \| \boldsymbol{V}(i, \cdot) - \boldsymbol{V}(j, \cdot) \|^2$$

while satisfying the isotropy condition

$$\sum_{i=1}^{N} \left(\sum_{j=1}^{N} \phi(\boldsymbol{x}_i, \boldsymbol{x}_j) \right) \boldsymbol{V}(i, \cdot) \boldsymbol{V}(i, \cdot)^T = \mathbf{I}$$

Then apply k-means clustering to the rows of $\boldsymbol{V}$.

Kernel spectral clustering

Kernel spectral clustering leads to an exact solution (Belkin & Niyogi, 2003).

Kernel spectral clustering: algorithm

1. Form the adjacency matrix $\boldsymbol{W} \in \mathbb{R}^{N \times N}$ with entries $w_{ij} = \phi(\boldsymbol{x}_i, \boldsymbol{x}_j)$.
2. Form the diagonal matrix $\boldsymbol{D} \in \mathbb{R}^{N \times N}$ containing the row sums $d_{ii} = \sum_{j=1}^{N} w_{ij}$.
3. Find the top eigenvectors $\tilde{\boldsymbol{V}} = \begin{bmatrix} \tilde{\boldsymbol{V}}_1 & \cdots & \tilde{\boldsymbol{V}}_r \end{bmatrix}$ of $\boldsymbol{D}^{-1/2} \boldsymbol{W} \boldsymbol{D}^{-1/2}$.
4. Set $\boldsymbol{V} = \boldsymbol{D}^{-1/2} \tilde{\boldsymbol{V}}$ and apply k-means to the rows of $\boldsymbol{V}$.