Distributed Estimation of Generalized Matrix Rank: Efficient Algorithms and Lower Bounds

Yuchen Zhang, Martin Wainwright, Michael Jordan

UC Berkeley
Multi-Party Linear Algebra

Given a matrix $A \in \mathbb{R}^{n \times n}$, compute an algebraic function $f(A)$. 

Linear algebra problems:

1. $f(A) = I$ (if $A$ is singular).
2. $f(A) = \text{rank of } A$.
3. $f(A) = \text{minimum singular value of } A$.
4. $f(A) = A^{-1}b$ for vector $b \in \mathbb{R}^n$.
5. $f(A) = \arg \min_{x \in \mathbb{R}^n} x^T Ax + b^T x$.

Multi-party Game: how to compute $f(A)$ if $A$ is held by two (or more) parties?

Alice has $A_1 \in \mathbb{R}^{n \times n}$, Bob has $A_2 \in \mathbb{R}^{n \times n}$, $A = A_1 + A_2$.
Multi-Party Linear Algebra

Given a matrix $A \in \mathbb{R}^{n \times n}$, compute an algebraic function $f(A)$.

Linear algebra problems:
- $f(A) = \mathbb{I}(A \text{ in singular})$.
- $f(A) = \text{rank of } A$.
- $f(A) = \text{minimum singular of } A$.
- $f(A) = A^{-1}b$ for vector $b \in \mathbb{R}^n$.
- $f(A) = \arg \min_{x \in \mathbb{R}^n} x^T Ax + b^T x$
Multi-Party Linear Algebra

Given a matrix $A \in \mathbb{R}^{n \times n}$, compute an algebraic function $f(A)$.

Linear algebra problems:

- $f(A) = \mathbb{I} (A \text{ in singular})$.
- $f(A) = \text{rank of } A$.
- $f(A) = \text{minimum singular of } A$.
- $f(A) = A^{-1}b$ for vector $b \in \mathbb{R}^n$.
- $f(A) = \arg \min_{x \in \mathbb{R}^n} x^T Ax + b^T x$

Multi-party Game: how to compute $f(A)$ if $A$ is held by two (or more) parties?

- Alice has $A_1 \in \mathbb{R}^{n \times n}$, Bob has $A_2 \in \mathbb{R}^{n \times n}$, $A = A_1 + A_2$
- Alice has $A_1 \in \mathbb{R}^{n \times n/2}$, Bob has $A_2 \in \mathbb{R}^{n \times n/2}$, $A = [A_1 \ A_2]$. 
Singularity Test

\[ f(A) = \mathbb{I}(A \text{ in singular}). \]

Question: How many bits should be communicated to compute \( f(A) \)?

Upper bound: \( O(n^2) \) (Bob sends everything to Alice)
Singularity Test

\[ f(A) = \mathbb{I}(A \text{ in singular}). \]

Question: How many bits should be communicated to compute \( f(A) \)?

Upper bound: \( O(n^2) \) (Bob sends everything to Alice)

Lower bound:
- \( \Omega(n^2) \) for deterministic algorithm. (Chu and Schnitger, 1991)
Singularity Test

\[ f(A) = \mathbb{I}(A \text{ in singular}). \]

Question: How many bits should be communicated to compute \( f(A) \)?

Upper bound: \( O(n^2) \) (Bob sends everything to Alice)

Lower bound:
- \( \Omega(n^2) \) for deterministic algorithm. (Chu and Schnitger, 1991)
- \( \Omega(1) \) for randomized algorithm.
Non-perfect Singularity Test

\[ f(A) = \mathbb{I}(A \text{ in singular}). \]

Question: How many bits should be communicated to compute \( f(A) \)?
Non-perfect Singularity Test

\[ f(A) = \mathbb{I}(A \text{ in singular}). \]

Question: How many bits should be communicated to compute \( f(A) \)?

Two classes of matrices are extremely difficult to distinguish:

- singular v.s. non-singular with arbitrarily small singular value.
Non-perfect Singularity Test

\[ f(A) = \mathbb{I}(A \text{ in singular}). \]

Question: How many bits should be communicated to compute \( f(A) \)?

Two classes of matrices are extremely difficult to distinguish:
- singular v.s. non-singular with arbitrarily small singular value.

In practice, do we really need perfect classification?
Non-perfect algorithm: allow \( f(A) \) making mistakes if the minimum singular value is between \((0, t)\). For example: \( t = n^{-10} \).
Non-perfect Singularity Test

\[ f(A) = \mathbb{I}(A \text{ in singular}). \]

Question: How many bits should be communicated to compute \( f(A) \)?

Two classes of matrices are extremely difficult to distinguish:
- singular v.s. non-singular with arbitrarily small singular value.

In practice, do we really need perfect classification?
Non-perfect algorithm: allow \( f(A) \) making mistakes if the minimum singular value is between \((0, t)\). For example: \( t = n^{-10} \).

Non-perfect algorithms:
Upper bound: \( O(n^2) \) (Bob sends everything to Alice)
Non-perfect Singularity Test

\[ f(A) = \mathbb{I}(A \text{ in singular}). \]

Question: How many bits should be communicated to compute \( f(A) \)?

Two classes of matrices are extremely difficult to distinguish:
   singular v.s. non-singular with arbitrarily small singular value.

In practice, do we really need perfect classification?
Non-perfect algorithm: allow \( f(A) \) making mistakes if the minimum singular value is between \((0, t)\). For example: \( t = n^{-10} \).

Non-perfect algorithms:
Upper bound: \( O(n^2) \) (Bob sends everything to Alice)
Lower bound: \( O(1) \) (deterministic or randomized).
Communication Complexity of Non-perfect Algorithm

For a broad class of problems:

- \( f(A) = \mathbb{I}(A \text{ in singular}) \).
- \( f(A) = \text{rank of } A \).
- \( f(A) = \text{minimum singular of } A \).
- \( f(A) = A^{-1} b \) for vector \( b \in \mathbb{R}^n \).
- \( f(A) = \arg \min_{x \in \mathbb{R}^n} x^T Ax + b^T x \)

By allowing:

- mistake in ambiguous cases
- approximation errors

We have:

- No communication-efficient algorithm.
- No non-trivial lower bound.
Communication Complexity of Non-perfect Algorithm

For a broad class of problems:

- $f(A) = \mathbb{I}(A \text{ in singular}).$
- $f(A) = \text{rank of } A.$
- $f(A) = \text{minimum singular of } A.$
- $f(A) = A^{-1}b$ for vector $b \in \mathbb{R}^n.$
- $f(A) = \arg \min_{x \in \mathbb{R}^n} x^T Ax + b^T x$

By allowing:

- mistake in ambiguous cases
- approximation errors
Communication Complexity of Non-perfect Algorithm

For a broad class of problems:

- $f(A) = \mathbb{I}(A \text{ in singular})$.
- $f(A) = \text{rank of } A$.
- $f(A) = \text{minimum singular of } A$.
- $f(A) = A^{-1}b$ for vector $b \in \mathbb{R}^n$.
- $f(A) = \arg\min_{x \in \mathbb{R}^n} x^T Ax + b^T x$

By allowing:

- mistake in ambiguous cases
- approximation errors

We have:

- No communication-efficient algorithm.
- No non-trivial lower bound.
Generalized Matrix Rank Estimation
Problem Set-up

Alice has PSD matrix $A_1 \in \mathbb{R}^{n \times n}$, Bob has PSD matrix $A_2 \in \mathbb{R}^{n \times n}$; $A := A_1 + A_2$

**Goal:** find $\text{rank}(A, c) = \text{“the number of eigenvalues of } A \text{ that are greater than } c\text{”}$.
Problem Set-up

Alice has PSD matrix $A_1 \in \mathbb{R}^{n \times n}$, Bob has PSD matrix $A_2 \in \mathbb{R}^{n \times n}$; $A := A_1 + A_2$

**Goal:** find $\text{rank}(A, c) = \text{“the number of eigenvalues of } A \text{ that are greater than } c\text{”}$.

- $c = 0 \Rightarrow \text{rank}(A, c) = \text{rank of } A$
- $c > 0 \Rightarrow \text{rank}(A, c) = \text{generalized rank of } A$
Problem Set-up

Alice has PSD matrix $A_1 \in \mathbb{R}^{n \times n}$, Bob has PSD matrix $A_2 \in \mathbb{R}^{n \times n}$; $A := A_1 + A_2$

**Goal:** find $\text{rank}(A, c) = \text{“the number of eigenvalues of } A \text{ that are greater than } c\text{”}$.

- $c = 0 \Rightarrow \text{rank}(A, c) = \text{rank of } A$
- $c > 0 \Rightarrow \text{rank}(A, c) = \text{generalized rank of } A$

**Equivalent Alternative Set-up:** Alice has matrix $X_1 \in \mathbb{R}^{n \times m}$, Bob has matrix $X_2 \in \mathbb{R}^{n \times m}$; $X := [X_1 \ X_2]$

**Goal:** find the number of singular values of $X$ that are greater than $c$. 
Applications

Many algorithm requires the knowledge of generalized rank in order to set hyper-parameters:

- **PCA**: set the number of principle components.
- **Personalized Recommendation**: user $v_u \in \mathbb{R}^d$, movie $v_m \in \mathbb{R}^d$.
  - Recommend movies by sorting $\text{affinity} = \langle v_u, v_m \rangle$.
  - Set $d$ by the generalized rank of user-movie matrix.
- **Matrix Completion**: set the nuclear-norm regularization coefficient.
- **Spectral Clustering**: set the number of clusters.
Applications

Many algorithm requires the knowledge of generalized rank in order to set hyper-parameters:

- **PCA**: set the number of principle components.
- **Personalized Recommendation**: user $v_u \in \mathbb{R}^d$, movie $v_m \in \mathbb{R}^d$.
  - Recommend movies by sorting affinity $= \langle v_u, v_m \rangle$.
  - Set $d$ by the generalized rank of user-movie matrix.
- **Matrix Completion**: set the nuclear-norm regularization coefficient.
- **Spectral Clustering**: set the number of clusters.

Power method has $\tilde{\Omega}(n^2)$ communication complexity.
Our algorithm has $\tilde{\Omega}(n)$ communication complexity.
Approximate Algorithm

In many applications, we don’t need exact result.

A rank estimator $\hat{r}$ is $(\epsilon, \delta)$-approximate if

$$(1 - \delta)\text{rank}(A, c + \epsilon) \leq \hat{r}(A, c) \leq (1 + \delta)\text{rank}(A, c - \epsilon)$$

- $\epsilon$: allow mistakes on ambiguous eigenvalues between $(c - \epsilon, c + \epsilon)$.
- $\delta$: relative approximation error.
For \((\epsilon, \delta)\)-approximate algorithms, we can define:

**Deterministic Complexity**: minimum amount of communication (bits) to guarantee that \(\hat{r}\) is always \((\epsilon, \delta)\)-approximate.

**Randomized Complexity**: minimum amount of communication (bits) to guarantee that \(\hat{r}\) is \((\epsilon, \delta)\)-approximate with high probability.
Communication Complexity of Approximate Algorithms

For \((\epsilon, \delta)\)-approximate algorithms, we can define:

**Deterministic Complexity**: minimum amount of communication (bits) to guarantee that \(\hat{r}\) is always \((\epsilon, \delta)\)-approximate.

**Randomized Complexity**: minimum amount of communication (bits) to guarantee that \(\hat{r}\) is \((\epsilon, \delta)\)-approximate with high probability.

**Deterministic v.s. Randomized**

**Example**: Alice has \(x \in \{0, 1\}^n\), Bob has \(y \in \{0, 1\}^n\); judge if \(x = y\).

- Deterministic: \(\Theta(n)\) communication.
- Randomized: \(\tilde{\Theta}(1)\) communication.
Communication Complexity of Approximate Algorithms

For \((\epsilon, \delta)\)-approximate algorithms, we can define:

**Deterministic Complexity**: minimum amount of communication (bits) to guarantee that \(\hat{r}\) is always \((\epsilon, \delta)\)-approximate.

**Randomized Complexity**: minimum amount of communication (bits) to guarantee that \(\hat{r}\) is \((\epsilon, \delta)\)-approximate with high probability.

**Deterministic v.s. Randomized**

**Example**: Alice has \(x \in \{0, 1\}^n\), Bob has \(y \in \{0, 1\}^n\); judge if \(x = y\).

- **Deterministic**: \(\Theta(n)\) communication.
- **Randomized**: \(\tilde{\Theta}(1)\) communication.

Randomized algorithms could be substantially more efficient!
Deterministic Algorithms for Generalized Rank Estimation
Upper Bound

Upper bound is trivial: Bob sends $A_2$ to Alice (after discretization).

Communication cost $= \tilde{O}(n^2)$. 
Upper Bound

Upper bound is trivial: Bob sends $A_2$ to Alice (after discretization).

**Communication cost** = $\tilde{O}(n^2)$.

**Is there a more efficient algorithm?**

Probably yes, because $O(\log n)$ bits are sufficient to encode the answer.
Theorem (lower bound for deterministic algorithm)

To implement a deterministic \((\frac{1}{40}, \frac{1}{12})\)-approximate algorithm, \(\Omega(n^2)\) bits must be communicated.

- No deterministic algorithm is substantially better than the trivial algorithm.
Randomized Algorithms for Generalized Rank Estimation
Polynomial Operator on Matrices

Assume:

- The eigenvalues of $A$ are in $[0, 1]$. (by normalization)
- $f(x) = \sum_{i=0}^{p} a_i x^i : \mathbb{R} \rightarrow \mathbb{R}$ is a polynomial function.

Treat $f$ as an operator on matrix $A$:

$$f(A) = \sum_{i=0}^{p} a_i A^i$$
Polynomial Operator on Matrices

Properties of $f(A)$:

- If $\lambda$ is an eigenvalue of $A$, then $f(\lambda)$ is an eigenvalue of $f(A)$. 

---

Yuchen Zhang  (UC Berkeley)  Generalized Matrix Rank Estimation  April 2015  16 / 21
Polynomial Operator on Matrices

Properties of $f(A)$:

- If $\lambda$ is an eigenvalue of $A$, then $f(\lambda)$ is an eigenvalue of $f(A)$.
- Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of $A$. If $g \sim N(0, I)$, then:

$$
\mathbb{E}[\|f(A)g\|_2^2] = \text{trace}[f(A)\mathbb{E}[gg^T]] = \text{trace}[f(A)] = \sum_{i=1}^{n} f(\lambda_i)
$$
Polynomial Operator on Matrices

Properties of $f(A)$:

- If $\lambda$ is an eigenvalue of $A$, then $f(\lambda)$ is an eigenvalue of $f(A)$.
- Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of $A$. If $g \sim N(0, I)$, then:

$$
\mathbb{E}[\|f(A)g\|_2^2] = \text{trace}[f(A)\mathbb{E}[gg^T]] = \text{trace}[f(A)] = \sum_{i=1}^{n} f(\lambda_i)
$$

- If $f$ is of degree-$p$, then $f(A)g$ can be computed by communicating $\tilde{O}(pn)$ bits:
  
  Round 1 : compute $Ag = A_1g + A_2g$
  Round 2 : compute $A^2g = A_1(Ag) + A_2(Ag)$
  
  ...... 

  Round $p$ : compute $A^pg = A_1(A^{p-1}g) + A_2(A^{p-1}g)$

Finally, compute $f(A)g = \sum_{i=0}^{p} a_i(A^i g)$. 
Polynomial Approximation

Construct a polynomial function \( f \) of order \( p \sim \log(n) \) and satisfies:

\[
\mathbb{I}(x > c + \epsilon) \lesssim f(x) \lesssim \mathbb{I}(x > c - \epsilon) \quad \text{for all} \quad x \in [0, 1]
\]
Polynomial Approximation

Construct a polynomial function $f$ of order $p \sim \log(n)$ and satisfies:

$$
I(x > c + \epsilon) \preceq f(x) \preceq I(x > c - \epsilon) \quad \text{for all } x \in [0, 1]
$$

Then:

- $\mathbb{E}[\|f(A)g\|_2^2] = \sum_{i=1}^{n} f(\lambda_i)$ is a $(\epsilon, 0)$-approximate estimator because:

  $$
  \text{rank}(A, c + \epsilon) = \sum_{i=1}^{n} I(\lambda_i > c + \epsilon) \preceq \sum_{i=1}^{n} f(\lambda_i) \preceq \sum_{i=1}^{n} I(\lambda_i > c - \epsilon) = \text{rank}(A, c - \epsilon)
  $$
**Polynomial Approximation**

Construct a polynomial function $f$ of order $p \sim \log(n)$ and satisfies:

$$\mathbb{I}(x > c + \epsilon) \preceq f(x) \preceq \mathbb{I}(x > c - \epsilon) \quad \text{for all } x \in [0, 1]$$

Then:

- $\mathbb{E}[\|f(A)g\|_2^2] = \sum_{i=1}^{n} f(\lambda_i)$ is a $(\epsilon, 0)$-approximate estimator because:

  $$\text{rank}(A, c + \epsilon) = \sum_{i=1}^{n} \mathbb{I}(\lambda_i > c + \epsilon) \preceq \sum_{i=1}^{n} f(\lambda_i)$$

  $$\preceq \sum_{i=1}^{n} \mathbb{I}(\lambda_i > c - \epsilon) = \text{rank}(A, c - \epsilon)$$

- $\|f(A)g\|_2^2$ is a good approximate to $\mathbb{E}[\|f(A)g\|_2^2]$ ($(\epsilon, \delta)$-approximate).
Polynomial Approximation

Construct a polynomial function $f$ of order $p \sim \log(n)$ and satisfies:

$$\mathbb{I}(x > c + \epsilon) \preceq f(x) \preceq \mathbb{I}(x > c - \epsilon) \quad \text{for all } x \in [0, 1]$$

Then:

- $\mathbb{E}[\|f(A)g\|_2^2] = \sum_{i=1}^{n} f(\lambda_i)$ is a $(\epsilon, 0)$-approximate estimator because:
  
  $$\text{rank}(A, c + \epsilon) = \sum_{i=1}^{n} \mathbb{I}(\lambda_i > c + \epsilon) \preceq \sum_{i=1}^{n} f(\lambda_i)$$
  
  $$\preceq \sum_{i=1}^{n} \mathbb{I}(\lambda_i > c - \epsilon) = \text{rank}(A, c - \epsilon)$$

- $\|f(A)g\|_2^2$ is a good approximate to $\mathbb{E}[\|f(A)g\|_2^2]$ ($(\epsilon, \delta)$-approximate).
- The communication cost for computing $\|f(A)g\|_2^2$ is $\tilde{O}(n)$.
Algorithm and Upper Bound

Algorithm:

1. Find polynomial function $f$ satisfying the inequality of the last slide.
2. Sample a random Gaussian vector $g \sim N(0, I_{n \times n})$.
3. Communicate $p$ rounds to compute $f(A)g$, then return $\|f(A)g\|_2^2$.

Upper Bound:

Theorem (upper bound for randomized algorithm)

For any constant $\epsilon, \delta > 0$, by appropriately choosing degree of function $f$, the proposed algorithm is $(\epsilon, \delta)$-approximate with high probability, and its communication cost is $\tilde{O}(n)$. 
Algorithm and Upper Bound

Algorithm:
1. Find polynomial function $f$ satisfying the inequality of the last slide.
2. Sample a random Gaussian vector $g \sim N(0, I_{n \times n})$.
3. Communicate $p$ rounds to compute $f(A)g$, then return $\|f(A)g\|_2^2$.

Upper Bound:

Theorem (upper bound for randomized algorithm)
For any constant $\epsilon, \delta > 0$, by appropriately choosing degree of function $f$, the proposed algorithm is $(\epsilon, \delta)$-approximate with high probability, and its communication cost is $\tilde{O}(n)$. 
Is the $\tilde{O}(n)$ upper bound improvable?
Probably yes, because $O(\log n)$ bits are sufficient to encode the answer.
Is the $\tilde{O}(n)$ upper bound improvable?
Probably yes, because $O(\log n)$ bits are sufficient to encode the answer.

**Theorem (lower bound for randomized algorithm)**

For $\epsilon = \delta = 0.2$, any randomized $(\epsilon, \delta)$-approximate algorithm must communicate $\Omega(n)$ bits.
Is the $\tilde{O}(n)$ upper bound improvable?
Probably yes, because $O(\log n)$ bits are sufficient to encode the answer.

**Theorem (lower bound for randomized algorithm)**

For $\epsilon = \delta = 0.2$, any randomized $(\epsilon, \delta)$-approximate algorithm must communicate $\Omega(n)$ bits.

- The $\tilde{O}(n)$ algorithm cannot be substantially improved.
- Lower bound proved by reducing from the 2-SUM problem (Woodruff and Zhang)
Summary

- Estimate generalized matrix rank: the number of eigenvalues that are greater than a threshold.
- Allow the algorithm making mistake on ambiguous cases; allow approximation errors.
- Deterministic algorithm suffers $\Omega(n^2)$ communication complexity.
- Randomized algorithm enjoys $\tilde{O}(n)$ communication cost. There is $\Omega(n)$ lower bound.
Open Questions

- **Generalize to $m$ players**: the upper bound is multiplied by $m$. Can we prove lower bound with this multiplicative factor?
- **Zero approximation error**: for $(\epsilon, 0)$-approximate algorithms, there is $\tilde{O}(n^2)$ upper bound and $\Omega(n)$ lower bound. Better algorithm? Tighter lower bound?
- Prove tight communication complexity bound for other linear algebra problems.