RandNLA: Randomization in Numerical Linear Algebra: Theory and Practice

Petros Drineas
RPI

Ilse Ipsen (organizer)
NCSU

Michael W. Mahoney
UC Berkeley

To access our web pages use your favorite search engine.
Why RandNLA?

Randomization and sampling allow us to design provably accurate algorithms for problems that are:

- **Massive**
  (matrices so large that can not be stored at all, or can only be stored in slow memory devices)

- **Computationally expensive or NP-hard**
  (combinatorial optimization problems, such as the Column Subset Selection Problem)
Randomized algorithms

• By (carefully) sampling rows/columns of a matrix, we can construct new, smaller matrices that are close to the original matrix (w.r.t. matrix norms) with high probability.

\[
\begin{bmatrix}
A \\
\end{bmatrix}
\begin{bmatrix}
B \\
\end{bmatrix}
\approx
\begin{bmatrix}
C \\
\end{bmatrix}
\begin{bmatrix}
R \\
\end{bmatrix}
\]

• By preprocessing the matrix using “random projection” matrices, we can sample rows/columns much less carefully (uniformly at random) and still get nice bounds with high probability.

Matrix perturbation theory

• The resulting smaller matrices behave similarly (e.g., in terms of singular values and singular vectors) to the original matrices thanks to the norm bounds.
Interplay

Applications in BIG DATA
(Data Mining, Information Retrieval,
Machine Learning, Bioinformatics, etc.)

Theoretical Computer Science
Randomized and approximation algorithms

Numerical Linear Algebra
Matrix computations and linear algebra (i.e., perturbation theory)
(Petros) discrete probability, randomized matrix multiplication

(Ilse) randomized Gram matrix multiplication, least-squares problems

(Michael) RandNLA practice: implementations and evaluations

RandNLA work will also be covered in Haim Avron's plenary talk on "Sketching-Based Matrix Computations for Large-Scale Data Analysis."

(IP3, Monday October 26, 1:45 pm-2:30 pm)
Let's start with discrete probability.

I will actually skip most of the (fairly introductory) material in the next several slides and will focus on measure concentration inequalities that are particularly useful in the analysis of RandNLA algorithms.

**Overview**

- Probabilistic experiments (definition, state space, events, independence, probabilities, conditional probabilities, union bound, etc.)
- Random variables (definition, independence, expectation, variance, etc.)
- Measure concentration inequalities (Markov, Chebyshev, Bernstein/Chernoff/Hoeffding, etc.)
A random experiment is any procedure that can be infinitely repeated and has a well-defined set of possible outcomes.

Sample space $\Omega$: the finite set of all possible outcomes of the random experiment.

Bernoulli trial: a random experiment with exactly two possible outcomes (success and failure).

Event: any subset of the sample space $\Omega$.

Example: let the random experiment be: “toss a coin three times”. The sample space is

$\Omega = \{HHH, HHT, HTH, HTT, THH, THT, TTH, TTT\}$.

The event $E$ described by “the output of the experiment was either all heads or all tails” is the set $\{HHH, TTT\}$. 
Probability: the probability measure or probability function $Pr$ mapping the finite sample space to the interval $[0,1]$, e.g., the function $Pr$ mapping every element $ω$ in $Ω$ to $0 ≤ Pr(ω) ≤ 1$.

Additional property (finite $Ω$): $\sum_{ω∈Ω} Pr(ω) = 1$

Uniform probability: all elements of the sample space have the same probability.

Let $A$ be an event (a subset of $Ω$). The probability of $A$ is: $Pr(A) = \sum_{ω∈A} Pr(ω)$

Obviously, $Pr(∅) = 0$  
$Pr(Ω) = 1$.  


Discrete vs. continuous probability

Countably infinite sample spaces $\Omega$ must be treated with some care; one can define the discrete probability space to be the set of all possible subsets of $\Omega$, i.e., the powerset of $\Omega$.

In this case, the probability measure or probability function $Pr$ is defined over all possible events and satisfies the so-called positivity, normalization, and additivity properties.

If the set of all possible events is a proper subset of the powerset of $\Omega$ or if $\Omega$ is uncountably infinite, even more care is necessary; this is not discrete probability any more.
Properties of events

Events are sets, thus set operations (union, intersection, complementation) are applicable.

**Examples:**

\[ \Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B) \]
\[ \Pr(\overline{A}) = 1 - \Pr(A). \]

The first property follows from the inclusion-exclusion principle, which can be generalized to more than two sets and thus to more than two events.

Additionally, if \( A \) is a subset of \( B \), then \( \Pr(A) \leq \Pr(B) \).
**Union bound:** for any set of events $A_i$, $i=1...n$,

$$\Pr \left( \bigcup_{i=1}^{n} A_i \right) \leq \sum_{i=1}^{n} \Pr (A_i)$$

Simple inductive proof, using the inclusion-exclusion principle for two sets. The events (sets) $A_i$ do not need to satisfy any special property.
Disjoint and independent events

**Disjoint or mutually exclusive events**: events $A$ and $B$ are disjoint or mutually exclusive if their intersection is empty.

**Independent events**: events $A$ and $B$ are independent if the occurrence of one does not affect the probability of the other.

Formally, two events $A$ and $B$ are independent if

$$\Pr(A \cap B) = \Pr(A)\Pr(B)$$
Random variables: functions mapping the sample space $\Omega$ to real numbers. (They are called variables, even though they really are functions.)

Notice that given any real number $a$, the inverse

$$X^{-1}(a) = \{ \omega \in \Omega \mid X(\omega) = a \}$$

is a subset of $\Omega$ and thus is an event and has a probability.
Random variables: functions mapping the sample space $\Omega$ to real numbers. (They are called variables, even though they really are functions.)

Notice that given any real number $a$, the inverse

$$X^{-1}(a) = \{ \omega \in \Omega \mid X(\omega) = a \}$$

is a subset of $\Omega$ and thus is an event and has a probability.

Abusing notation: we use $\Pr(X = a)$ instead of $\Pr(X^{-1}(a))$.

This function of $a$ is of great interest. We can also define $\Pr(X \leq a)$ as follows:

$$\Pr(X \leq a) = \Pr(X^{-1}((-\infty, a])) = \Pr(\{ \omega \in \Omega \mid X(\omega) \leq a \})$$
Example: let \( \Omega \) be the sample space of two dice rolls with 36 elements and assume that both dice are fair.

The sum of the two numbers on the two dice (denoted by - say - \( S \)) is a random variable, taking values between 2 and 12.

We can compute the probabilities \( \Pr(S=s) \):

<table>
<thead>
<tr>
<th>( s )</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Pr(S=s) )</td>
<td>( \frac{1}{36} )</td>
<td>( \frac{2}{36} )</td>
<td>( \frac{3}{36} )</td>
<td>( \frac{4}{36} )</td>
<td>( \frac{5}{36} )</td>
<td>( \frac{6}{36} )</td>
<td>( \frac{5}{36} )</td>
<td>( \frac{4}{36} )</td>
<td>( \frac{3}{36} )</td>
<td>( \frac{2}{36} )</td>
<td>( \frac{1}{36} )</td>
</tr>
</tbody>
</table>
PMF and CDF

**Probability mass function or pmf:**

\[ f(a) = \Pr(X = a), \quad a \in \mathbb{R} \]

**Cumulative distribution function or cdf:**

\[ F(a) = \Pr(X \leq a), \quad a \in \mathbb{R} \]

Clearly, \[ F(a) = \sum_{x \leq a} f(x). \]

If the sample space is countably infinite some more care required; a lot more care is required if the sample space is uncountably infinite.
Independent random variables

Two random variables $X$ and $Y$ are independent if (for all reals $a$, $b$)

$$\Pr(X = a \text{ and } Y = b) = \Pr(X = a)\Pr(Y = b)$$

Joint mass function

$$f_{X,Y}(a,b) = \Pr(\omega \in \Omega \mid (X(\omega) = a) \land (Y(\omega) = b))$$
Given a random variable $X$, its expectation $E(X)$ is defined as

$$E(X) = \sum_{x \in X(\Omega)} x \cdot Pr(X = x)$$

$X(\Omega)$ is the image of $X$ (recall that $X$ is a function). For finite $\Omega$

$$E(X) = \sum_{\omega \in \Omega} X(\omega) Pr(\omega).$$

For countably infinite sample spaces care needs to be taken regarding the convergence of the appropriate sum.

The Lebesgue-Stieljes integral can be used for uncountable sample spaces.
Properties of expectation

**Linearity of expectation:** for any random variables $X$ and $Y$ and real $\lambda$

\[
E(X + Y) = E(X) + E(Y)
\]
\[
E(\lambda X) = \lambda E(X).
\]

The first property generalizes to any (finite) sum of random variables and holds even if the random variables are not independent.

If two random variables $X$ and $Y$ are independent then

\[
E(XY) = E(X)E(Y).
\]
Variance

The variance of a random variable $X$ is defined as

$$\text{Var}(X) = \mathbb{E}(X - \mathbb{E}(X))^2.$$ 

In words, it measures the average of the (square) of the difference $X - \mathbb{E}(X)$. The standard deviation is the square root of the variance.

Easy to prove:

$$\text{Var}(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 \leq \mathbb{E}(X^2).$$
If $X$ and $Y$ are independent random variables, then

$$\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y).$$

For any real $\lambda$

$$\text{Var}(\lambda X) = \lambda^2 \text{Var}(X).$$
Markov’s inequality

Let $X$ be a non-negative random variable; for any $a > 0$

$$\Pr(X \geq a) \leq \frac{E(X)}{a}$$

Really simple to apply: only needs a bound for the expectation of $X$. 
Markov’s inequality: equivalent formulas

Let $X$ be a **non-negative** random variable; for any $k > 1$

$$\Pr (X \geq k \cdot E(X)) \leq \frac{1}{k}$$

or, equivalently,

$$\Pr (X < k \cdot E(X)) \geq 1 - \frac{1}{k}$$
Markov’s inequality: proof hint

Let \( X \) be a \textbf{non-negative} random variable; for any \( a > 0 \)

\[
\Pr(X \geq a) \leq \frac{E(X)}{a}
\]

(Proof hint: define the random variable \( I_a \) that takes the value 1 if \( X \geq a \) and 0 otherwise; come up with an obvious upper bound for \( I_a \) and then bound its expectation.)
Let $X$ be a random variable with expectation $E(X)$ (finite) and variance $\text{Var}(X) = \sigma^2$ (also finite): Then, for any $a > 0$,

$$\Pr \left( |X - E(X)| \geq a\sigma \right) \leq \frac{1}{a^2}$$

We now need bounds for the expectation and the variance of $X$.

Proof hint: Apply Markov's inequality on $(X-E(X))^2$. 

\[ \text{Chebyshev's inequality} \]
Bernstein-type inequalities

We now focus on inequalities that measure how much a sum of random variables deviates from its expectation.

Given independent random variables $X_i$ for $i=1\ldots n$ we seek bounds for:

$$\left| \sum_{i=1}^{n} X_i - \sum_{i=1}^{n} \mathbb{E}(X_i) \right|$$

These bounds should hold with “high probability”.
Bernstein-type inequalities

We now focus on inequalities that measure how much a sum of random variables deviates from its expectation.

Given independent random variables $X_i$ for $i=1...n$ we seek bounds for:

$$\left| \sum_{i=1}^{n} X_i - \sum_{i=1}^{n} E(X_i) \right|$$

These bounds should hold with “high probability”.

**Remark 1:** We already know from the Central Limit Theorem that a sum of $n$ independent, identically distributed random variables with bounded, common, mean and variance converges to a normal distribution as $n$ approaches infinity. The inequalities that we will discuss deal with smaller, finite values of $n$.

**Remark 2:** Early versions of such inequalities are often attributed to Bernstein in the 1920’s, but have been rediscovered in various forms several times (starting in the 1950s with Chernoff bounds, Hoeffding’s inequality, Azuma’s inequality, etc.).
Bernstein-type inequalities

Given independent random variables $X_i$ such that $E(X_i) = 0$ and $-M \leq X_i \leq M$ for all $i=1...n$,

$$
\Pr \left( \left| \sum_{i=1}^{n} X_i \right| > a \right) \leq 2\exp \left( -\frac{a^2/2}{\sum_{i=1}^{n} E\left( X_i^2 \right) + Ma/3} \right)
$$
Bernstein-type inequalities

Given independent random variables $X_i$ such that $E(X_i) = 0$ and $-M \leq X_i \leq M$ for all $i=1...n$,

$$\Pr \left( \left| \sum_{i=1}^{n} X_i \right| > a \right) \leq 2\exp \left( -\frac{a^2/2}{\sum_{i=1}^{n} E\left(X_i^2\right) + Ma/3} \right)$$

$0 = \sum_{i=1}^{n} E(X_i)$
Bernstein-type inequalities

Given independent random variables $X_i$ such that $E(X_i) = 0$ and $-M \leq X_i \leq M$ for all $i=1...n$,

$$\Pr \left( \left| \sum_{i=1}^{n} X_i \right| > a \right) \leq 2\exp \left( -\frac{a^2/2}{\sum_{i=1}^{n} E\left( X_j^2 \right) + Ma/3} \right)$$

The above inequality holds for any $a > 0$.

- The expectation of the sum of the $X_i$'s is zero.

- In words, the above inequality states that the probability that the sum of the $X_i$'s deviates from its expectation by more than $a$ is bounded.

- Many variants can be derived.
Approximating Matrix Multiplication

(Drineas, Kannan, Mahoney SICOMP 2006)

Problem Statement

Given an \( m \)-by-\( n \) matrix \( A \) and an \( n \)-by-\( p \) matrix \( B \), approximate the product \( A \cdot B \),

OR, equivalently,

Approximate the sum of \( n \) rank-one matrices.

\[
A \cdot B = \sum_{i=1}^{n} \left( A_{*i} \right) \cdot \left( B_{i*} \right)
\]

Each term in the summation is a rank-one matrix

\( \in \mathbb{R}^{m \times p} \)

i-th column of \( A \)

i-th row of \( B \)
Algorithm

1. Fix a set of probabilities \( p_i, \ i = 1 \ldots n \), summing up to one.
2. For \( t = 1 \ldots c \),
   
   set \( j_t = i \), where \( \Pr(j_t = i) = p_i \).
   
   (Pick \( c \) terms of the sum, with replacement, with respect to the \( p_i \).)
3. Approximate the product \( AB \) by summing the \( c \) terms, after scaling.
Sampling (cont’d)

(DKM SICOMP 2006)

\[ A \cdot B = \sum_{i=1}^{n} \begin{pmatrix} A_{i*} \end{pmatrix} \cdot \begin{pmatrix} B_{i*} \end{pmatrix} \]

\[ \in \mathbb{R}^{m \times p} \]

Keeping the terms \( j_1, j_2, \ldots, j_c \).
The algorithm (matrix notation)

$$\begin{pmatrix} A \\ m \times n \end{pmatrix} \cdot \begin{pmatrix} B \\ n \times p \end{pmatrix} \approx \begin{pmatrix} C \\ m \times c \end{pmatrix} \cdot \begin{pmatrix} R \\ c \times p \end{pmatrix}$$

Algorithm

1. Pick $c$ columns of $A$ to form an $m$-by-$c$ matrix $C$ and the corresponding $c$ rows of $B$ to form a $c$-by-$p$ matrix $R$.

2. Approximate $A \cdot B$ by $C \cdot R$.

Notes

3. We pick the columns and rows with non-uniform probabilities.

4. We scale the columns (rows) prior to including them in $C$ ($R$).
The algorithm (matrix notation, cont’d)

\[
\begin{pmatrix}
A \\
m \times n
\end{pmatrix} \cdot \begin{pmatrix}
B \\
n \times p
\end{pmatrix} \approx \begin{pmatrix}
C \\
m \times c
\end{pmatrix} \cdot \begin{pmatrix}
R \\
c \times p
\end{pmatrix}
\]

- Create \( C \) and \( R \) by performing \( c \) i.i.d. trials, with replacement.
- For \( t = 1 \ldots c \), pick a column \( A(j_t) \) and a row \( B(j_t) \) with probability

\[
\Pr(j_t = i) = \frac{\|A_{*i}\|_2 \|B_{i*}\|_2}{\sum_{j=1}^n \|A_{*j}\|_2 \|B_{j*}\|_2}
\]

- Include \( A_{*j_t}/(cp_{j_t})^{1/2} \) as a column of \( C \), and \( B_{j_t*}/(cp_{j_t})^{1/2} \) as a row of \( R \).
We can also use the sampling matrix notation:

Let $S$ be an $n$-by-$c$ matrix whose $t$-th column (for $t = 1 \ldots c$) has a single non-zero entry, namely

$$S_{jt} = \frac{1}{\sqrt{cP_{jt}}}$$

Clearly:

$$A \cdot B \approx C \cdot R = (AS) \cdot (S^T B)$$
The algorithm (matrix notation, cont’d)

We can also use the sampling matrix notation:

Let \( S \) be an \( n \)-by-\( c \) matrix whose \( t \)-th column (for \( t = 1 \ldots c \)) has a single non-zero entry, namely

\[
S_{jt} = \frac{1}{\sqrt{c \cdot p_{jt}}}
\]

Clearly:

\[
A \cdot B \approx C \cdot R = (AS) \cdot (S^T B)
\]

Remark: \( S \) is sparse (has exactly \( c \) non-zero elements, one per column).
Simple Lemmas

• It is easy to implement this particular sampling in two passes, when the input matrices are given as a sequence of triplets \((i,j,A_{ij})\).

• The expectation of \(CR\) (element-wise) is \(AB\) (unbiased estimator), regardless of the sampling probabilities, i.e.,

\[
\mathbb{E}[(CR)_{i,j}] = (AB)_{i,j}
\]

• We can also bound the variance of the \((i,j)\)-th entry of \(CR\):

\[
\text{Var}[(CR)_{i,j}] = \frac{1}{c} \sum_{k=1}^{n} \frac{A_{ik}^2 B_{kj}^2}{p_k} - \frac{1}{c} (AB)_{i,j}^2
\]
Simple Lemmas

\[ \mathbf{E} [(CR)_{ij}] = (AB)_{ij} \]

\[ \text{Var} [(CR)_{ij}] = \frac{1}{c} \sum_{k=1}^{n} \frac{A_{ik}^2 B_{kj}^2}{p_k} - \frac{1}{c} (AB)_{ij}^2 \]

The above two bounds can now be used to bound

\[ \mathbf{E} \left[ \| AB - CR \|^2_F \right] \]

Our particular choice of sampling probabilities minimizes the above expectation.
A bound for the Frobenius norm

For the above algorithm,

\[ \mathbf{E} \| AB - CR \|_F = \mathbf{E} \| AB - ASS^T B \|_F \leq \frac{1}{\sqrt{c}} \| A \|_F \| B \|_F \]

- We can now use Markov’s inequality to directly get a “constant probability” bound.
- “High probability” follows from a martingale argument (we will skip this discussion).
- The above bound immediately implies an upper bound for the spectral norm of the error \( AB - CR \) (but better bounds can be derived).
Special case: $B = A^T$

If $B = A^T$, then the sampling probabilities are

$$\Pr(j_t = i) = \frac{\|A_{*i}\|_2^2}{\sum_{j=1}^n \|A_{*j}\|_2^2} = \frac{\|A_{*i}\|_2^2}{\|A\|_F^2}$$

Also, $R = C^T$, and the error bounds are:

$$\mathbb{E} \left\| AA^T - C C^T \right\|_F = \mathbb{E} \left\| AA^T - A S S^T A^T \right\|_F \leq \frac{1}{\sqrt{c}} \|A\|_F^2$$
Special case: $B = A^T$ (cont’d)

(Drineas et al. Num Math 2011, Theorem 4)

A better spectral norm bound via matrix Chernoff/Bernstein inequalities:

**Assumptions:**

- Spectral norm of $A$ is one (not important, just normalization)
- Frobenius norm of $A$ is at least 0.2 (not important, simplifies bounds).
- **Important:** Set

$$c = \Omega \left( \frac{\|A\|_F^2}{\varepsilon^2} \ln \left( \frac{\|A\|_F^2}{\varepsilon^2 \sqrt{\delta}} \right) \right)$$

Then: for any $0 < \varepsilon < 1$, with probability at least $1 - \delta$,

$$\|AA^T - CC^T\|_2 = \|AA^T - ASS^T A^T\|_2 \leq \varepsilon$$
Let $y^1, y^2, \ldots, y^c$ be independent identically distributed copies of the $m$-dimensional random vector $y$ with

$$\|y\|_2 \leq M \quad \text{and} \quad \|\mathbb{E}(yy^T)\|_2 \leq 1$$

Then, for any $a > 0$,

$$\left\| \frac{1}{c} \sum_{i=1}^c y^i y^i^T - \mathbb{E}(yy^T) \right\|_2 \leq a$$

holds with probability at least

$$1 - (2c)^2 \exp \left( -\frac{ca^2}{16M^2 + 8M^2a} \right)$$
Proof outline  
(Drineas et al. Num Math 2011, Theorem 4)

Define the random vector $y$ such that

$$\Pr \left( y = \frac{1}{\sqrt{p_i}} A_{*i} \right) = p_i$$

Now, it is easy to see that the matrix $C$ has columns

$$\frac{1}{\sqrt{c}} y^1, \frac{1}{\sqrt{c}} y^2, \ldots, \frac{1}{\sqrt{c}} y^c$$

where $y^1, y^2, \ldots, y^c$ are independent copies of $y$. It is easy to prove that

$$\mathbb{E} [yy^T] = AA^T \quad \text{and} \quad CC^T = ASS^T A^T = \frac{1}{c} \sum_{t=1}^{c} y^t y^{tT}.$$
Proof outline
(Drineas et al. Num Math 2011, Theorem 4)

We can now upper bound the norm of the vector $y$:

$$M = \|y\|_2 = \frac{1}{\sqrt{P_i}} \|A*_{i}\|_2$$

Also, given our assumption that the spectral norm of $A$ is at most one, we get:

$$\|E[yy^T]\|_2 = \|AA^T\|_2 \leq \|A\|_2 \|A^T\|_2 \leq 1.$$ 

We can now apply a matrix-Bernstein inequality (Lemma 1 of Oliveira 2010) to get the desired bound.
Using a dense $S$ (instead of a sampling matrix...)

We approximated the product $AA^T$ as follows:

$$A \cdot A^T \approx C \cdot C^T = (AS) \cdot (S^T A^T)$$

Recall that $S$ is an $n$-by-$c$ sparse matrix (one non-zero entry per column).

Let’s replace $S$ by a dense matrix, the random sign matrix:

$$S_{ij} = \begin{cases} 
+1 / \sqrt{c}, & \text{w.p. } 1/2 \\
-1 / \sqrt{c}, & \text{w.p. } 1/2
\end{cases}$$

Note that $S$ is oblivious to the input matrix $A$!
Using a dense $S$ (instead of a sampling matrix...) (and assuming $A$ is normalized)

Approximate the product $AA^T$ (assuming that the spectral norm of $A$ is one):

$$A \cdot A^T \approx C \cdot C^T = (AS) \cdot (S^T A^T)$$

Let $S$ be a dense matrix, the random sign matrix:

$$S_{ij} = \begin{cases} +1/\sqrt{c} & \text{w.p. } 1/2 \\ -1/\sqrt{c} & \text{w.p. } 1/2 \end{cases}$$

If

$$c = \Omega \left( \frac{\|A\|_F^2}{\epsilon^2 \ln m} \right)$$

then, with high probability:

$$\|AA^T - CC^T\|_2 = \|AA^T - ASS^T A^T\|_2 \leq \epsilon$$
Using a dense $S$ (instead of a sampling matrix...) (and assuming $A$ is normalized)

Approximate the product $AA^T$ (assuming that the spectral norm of $A$ is one):

$$A \cdot A^T \approx C \cdot C^T = (AS) \cdot (S^T A^T)$$

Let $S$ be a dense matrix, the random sign matrix:

$$S_{ij} = \begin{cases} +1/\sqrt{c} , \text{w.p. } 1/2 \\
-1/\sqrt{c} , \text{w.p. } 1/2 \end{cases}$$

If

$$c = \Omega \left( \frac{\|A\|_F^2}{\epsilon^2} \ln m \right)$$

then, with high probability:

$$\|AA^T - CC^T\|_2 = \|AA^T - ASS^T A^T\|_2 \leq \epsilon$$
Other choices for dense matrices $S$?

Why bother with a sign matrix?

(Computing the product $AS$ is somewhat slow, taking $O(mnc)$ time.)

Similar bounds are known for better, i.e., computationally more efficient, choices of “random projection” matrices $S$, most notably:

- When $S$ is the so-called subsampled Hadamard Transform Matrix.
  (much faster; avoids full matrix-matrix multiplication; see Sarlos FOCS 2006 and Drineas et al. (2011) Num Math)

- When $S$ is the input sparsity time projection matrix of Clarkson & Woodruff STOC 2013.
  (the matrix multiplication result appears in Mahoney & Meng STOC 2013 and was improved by Nelson and Huy FOCS 2013).
Recap: approximating matrix multiplication

We approximated the product $AB$ as follows:

$$A \cdot B \approx C \cdot R = (AS) \cdot (S^T B)$$

Let $S$ be a **sampling** matrix (actual columns from $A$ and rows from $B$ are selected):

We need to carefully sample columns of $A$ (rows of $B$) with probabilities that depend on their norms in order to get “good” bounds of the following form:

$$\mathbb{E} \left[ \| AB - CR \|_F \right] = \mathbb{E} \left[ \| AB - ASS^T B \|_F \right] \leq \frac{1}{\sqrt{c}} \| A \|_F \| B \|_F$$

$$\| AA^T - CC^T \|_2 = \| AA^T - ASS^T A^T \|_2 \leq \varepsilon$$

Holds with probability at least $1-\delta$ by setting

$$c = \Omega \left( \frac{\| A \|_F^2}{\varepsilon^2} \ln \left( \frac{\| A \|_F^2}{\varepsilon^2 \sqrt{\delta}} \right) \right)$$
Randomized Gram Matrix Multiplication and Least Squares
Outline

1. How to sample

2. Randomized matrix multiplication $A^T A$, again but now with uniform probabilities

3. Singular values of sampled orthonormal matrices

4. Randomized least squares

Assume: Exact arithmetic
How to Sample, Non-Uniformly

Given: Probabilities $0 \leq p_1 \leq \cdots \leq p_m$ with $\sum_{i=1}^{m} p_i = 1$

Want: Sample index $t = j$ from $\{1, \ldots, m\}$ with probability $p_j$

Inversion by sequential search [Devroye 1986]

1. Determine partial sums

$$S_k \equiv \sum_{i=1}^{k} p_i \quad 1 \leq k \leq m$$

2. Pick uniform $[0, 1]$ random variable $U$

3. Determine integer $j$ with $S_{j-1} < U \leq S_j$

4. Sampled index: $t = j$ with probability $p_j = S_j - S_{j-1}$
How to Sample, Uniformly

Want: Sample $c$ indices from $\{1, \ldots, m\}$ independently and identically, each with probability $1/m$

Matlab

\begin{align*}
\text{randi}(m, [c, 1])
\end{align*}

c $\times$ 1 vector of \textit{pseudo-random integers} between 1 and $m$
Randomized Gram Matrix Multiplication
DKM Algorithm

[Drineas, Kannan & Mahoney 2006]
DKM Algorithm (special case)

- Given: \( \mathbf{A} = \begin{pmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_m \end{pmatrix}^T \in \mathbb{R}^{m \times n} \) with rows \( \mathbf{a}_j \in \mathbb{R}^n \)
- Gram product = sum of outer products

\[
\mathbf{A}^T \mathbf{A} = \sum_{j=1}^{m} \mathbf{a}_j \mathbf{a}_j^T \in \mathbb{R}^{n \times n}
\]
DKM Algorithm (special case)

- Given: \( \mathbf{A} = (\mathbf{a}_1 \cdots \mathbf{a}_m)^T \in \mathbb{R}^{m \times n} \) with rows \( \mathbf{a}_j \in \mathbb{R}^n \)

- Gram product = sum of outer products

\[
\mathbf{A}^T \mathbf{A} = \sum_{j=1}^{m} \mathbf{a}_j \mathbf{a}_j^T \in \mathbb{R}^{n \times n}
\]

- DKM algorithm samples \( c \) rows of \( \mathbf{A} \), independently and identically, with uniform probabilities \( p_j = 1/m \)

\[
\mathbf{S} = \sqrt{\frac{m}{c}} (\mathbf{e}_{t_1} \cdots \mathbf{e}_{t_c})^T \in \mathbb{R}^{c \times m}
\]

\( \mathbf{e}_{t_1}, \ldots, \mathbf{e}_{t_c} \) not necessarily distinct
DKM Algorithm \textit{(special case)}

- **Given:** \( A = \begin{pmatrix} a_1 & \cdots & a_m \end{pmatrix}^T \in \mathbb{R}^{m \times n} \) with rows \( a_j \in \mathbb{R}^n \)

- **Gram product = sum of outer products**

\[
A^T A = \sum_{j=1}^{m} a_j a_j^T \in \mathbb{R}^{n \times n}
\]

- **DKM algorithm samples** \( c \) **rows of** \( A \), **independently and identically**, with uniform probabilities \( p_j = 1/m \)

\[
S = \sqrt{\frac{m}{c}} \begin{pmatrix} e_{t_1} & \cdots & e_{t_c} \end{pmatrix}^T \in \mathbb{R}^{c \times m}
\]

\( e_{t_1}, \ldots, e_{t_c} \) not necessarily distinct

- **Output = sum of** \( c \) **scaled outer products**

\[
(SA)^T (SA) = \frac{m}{c} \sum_{j=1}^{c} a_{t_j} a_{t_j}^T
\]
Expected Value of Output

- Output from DKM algorithm

\[ \mathbf{X} = (\mathbf{SA})^T (\mathbf{SA}) = \mathbf{X}_1 + \cdots + \mathbf{X}_c \]

- Sum of matrix-valued random variables

\[ \mathbf{X}_j = \frac{m}{c} \mathbf{a}_t \mathbf{a}_t^T \]

- Expected value

\[ \mathbb{E}[\mathbf{X}_j] = \sum_{i=1}^{m} \frac{1}{m} \frac{m}{c} \mathbf{a}_i \mathbf{a}_i^T = \frac{1}{c} \mathbf{A}^T \mathbf{A} \]

- Output is unbiased estimator

\[ \mathbb{E}[\mathbf{X}] = c \mathbb{E}[\mathbf{X}_j] = \mathbf{A}^T \mathbf{A} \]
Idea for Probabilistic Bounds

- Output from DKM algorithm
  \[ X = (SA)^T (SA) = X_1 + \cdots + X_c \]

- Independent matrix-valued random variables
  \[ X_j = \frac{m}{c} a_{t_j} a_{t_j}^T \quad \text{with} \quad \mathbb{E}[X_j] = \frac{1}{c} A^T A \]

- Change of variable to zero-mean
  \[ Y_j = X_j - \frac{1}{c} A^T A \quad \text{with} \quad \mathbb{E}[Y_j] = 0 \]

- Sum
  \[ Y = Y_1 + \cdots + Y_c \quad \text{with} \quad \mathbb{E}[Y] = 0 \]
  \[ = X - A^T A \]

Deviation of $Y$ from mean zero $\triangleq$ Absolute error in $X$
Idea for Probabilistic Bounds, ctd

- Apply matrix Bernstein concentration inequality to $Y$

$$\Pr \left[ \|Y\|_2 \geq \hat{\epsilon} \right] \leq \delta(\hat{\epsilon}, \ldots)$$

Deviation of $Y$ from mean since $\mathbb{E}[Y] = 0$

- Retrieve original random variables

$$Y = X - \mathbb{E}[X] = X - A^T A$$

- Relative error due to randomization for $X$

$$\Pr \left[ \|X - A^T A\|_2 \leq \hat{\epsilon} \right] \geq 1 - \delta(\hat{\epsilon}, \ldots)$$

with $\hat{\epsilon} = \|A^T A\|_2 \epsilon$
Matrix Bernstein Concentration Inequality [Tropp 2011]

1. Independent random symmetric $Y_j \in \mathbb{R}^{n \times n}$
2. $\mathbb{E}[Y_j] = 0 \quad \{\text{zero mean}\}$
3. $\|Y_j\|_2 \leq \beta \quad \{\text{bounded}\}$
4. $\left\| \sum_j \mathbb{E}[Y_j^2] \right\|_2 \leq \nu \quad \{"\text{variance}"\}$

Failure probability: For any $\hat{\epsilon} > 0$

$$\Pr \left[ \left\| \sum_j Y_j \right\|_2 \geq \hat{\epsilon} \right] \leq n \exp \left( -\frac{\hat{\epsilon}^2}{2\nu + \beta \hat{\epsilon}/3} \right)$$

$\{\text{Deviation from mean}\}$
Check Assumptions for Concentration Inequality

1. Independent symmetric $n \times n$ random variables

$$Y_j = X_j - \frac{1}{c} A^T A = \frac{1}{c} \left( m a_{t_j} a_{t_j}^T - A^T A \right)$$

2. Zero mean $\mathbb{E}[Y_j] = 0$

3. Bounded

$$\|Y_j\|_2 \leq \frac{1}{c} \max_{1 \leq i \leq m} \left\{ m \|a_i\|^2, \|A\|^2 \right\} = \frac{m}{c} \max_{1 \leq i \leq m} \|a_i\|^2_2 \leq \beta$$

4. Variance

$$\left\| \sum_{j=1}^c \mathbb{E}[Y_j^2] \right\|_2 \leq \beta \|A\|^2_2$$
Apply Concentration Inequality

- Failure probability
  \[ \delta = n \exp \left( -\frac{\hat{\epsilon}^2/2}{\nu + \beta \hat{\epsilon}/3} \right) \]
  \[ \nu = \beta \|A\|_2^2 \text{ and } \hat{\epsilon} = \epsilon \|A\|_2^2 \]
  \[ \delta = n \exp \left( -\frac{\epsilon^2 \|A\|_2^2}{2 \beta (1 + \epsilon/3)} \right) \]

- Row mass \( \mu = \max_{1 \leq i \leq m} \|a_i\|_2^2 / \|A\|_2^2 \)
  \[ \beta = \frac{m}{c} \|A\|_2^2 \mu \]
  \[ \delta = n \exp \left( -\frac{c \epsilon^2}{2 m \mu (1 + \epsilon/3)} \right) \]
Relative Error due to Randomization  [Holodnak & Ipsen 2015]

- **Success probability**

\[
\Pr \left[ \| (SA)^T (SA) - A^T A \|_2 \leq \epsilon \| A^T A \|_2 \right] \geq 1 - \delta
\]

where

\[
\delta = n \exp \left( -\frac{c \epsilon^2}{2 m \mu (1 + \epsilon/3)} \right)
\]

- **Row mass**  \( \mu = \max_{1 \leq i \leq m} \| a_i \|_2^2 / \| A \|_2^2 \)
Relative Error due to Randomization [Holodnak & Ipsen 2015]

- **Success probability**
  \[
  \Pr \left[ \| (SA)^T (SA) - A^T A \|_2 \leq \epsilon \| A^T A \|_2 \right] \geq 1 - \delta
  \]

  where
  \[
  \delta = n \exp \left( - \frac{c \epsilon^2}{2 m \mu (1 + \epsilon/3)} \right)
  \]

- **Row mass** \( \mu = \max_{1 \leq i \leq m} \| a_i \|^2 / \| A \|^2 \)

- **Sampling amount**: If
  \[
  c \geq \frac{8}{3} m \mu \frac{\ln(n/\delta)}{\epsilon^2}
  \]
  then with probability \( \geq 1 - \delta \)
  \[
  \| (SA)^T (SA) - A^T A \|_2 \leq \epsilon \| A^T A \|_2
  \]
Conclusion for DKM Algorithm

- \( \mathbf{A} = (a_1 \cdots a_m)^T \in \mathbb{R}^{m \times n} \)
  \[ \mu = \max_{1 \leq i \leq m} \frac{\|a_i\|_2^2}{\|\mathbf{A}\|_2^2} \]
- Failure probability \( 0 < \delta \leq 1 \)
- Relative error bound \( 0 < \epsilon < 1 \)
- Uniform probabilities \( p_j = 1/m, \ 1 \leq j \leq m \)
- Sampling amount
  \[ c \geq \frac{8}{3} m \mu \frac{\ln (n/\delta)}{\epsilon^2} \]

With probability \( \geq 1 - \delta \)
DKM algorithm samples \( c \) rows \( \mathbf{S} \mathbf{A} \) with

\[ \| (\mathbf{S} \mathbf{A})^T (\mathbf{S} \mathbf{A}) - \mathbf{A}^T \mathbf{A} \|_2 \leq \epsilon \| \mathbf{A}^T \mathbf{A} \|_2 \]
Matrices with Orthonormal Columns

- \( Q = \begin{pmatrix} q_1 & \cdots & q_m \end{pmatrix}^T \in \mathbb{R}^{m \times n} \) with \( Q^T Q = I_n \)

- Coherence

  \[ \mu = \max_{1 \leq i \leq m} \|q_i\|_2^2 \quad \frac{n}{m} \leq \mu \leq 1 \]
Coherence  [Donoho & Huo 2001]

\[ Q = (q_1 \cdots q_m)^T \in \mathbb{R}^{m \times n} \text{ with } Q^TQ = I_n \]

Coherence  \( \mu = \max_{1 \leq i \leq m} ||q_i||_2^2 \)

- \( \frac{n}{m} \leq \mu \leq 1 \)

- **Maximal** coherence:  \( \mu = 1 \)
  - At least one row \( q_j^T \) is a canonical vector

- **Minimal** coherence:  \( \mu = \frac{n}{m} \)
  - Columns of \( Q \) are columns of a \( m \times m \) Hadamard matrix

- Property of the column space  \( S = \text{range}(Q) \)
  - Independent of particular orthonormal basis for \( S \)

- Coherence measures
  - Correlation with canonical basis
  - Difficulty of *recovering* matrix from sampling
Matrices with Orthonormal Columns

- \( Q = (q_1 \cdots q_m)^T \in \mathbb{R}^{m \times n} \) with \( Q^T Q = I_n \)

- Coherence

\[
\mu = \max_{1 \leq i \leq m} \|q_i\|_2^2 \quad \frac{n}{m} \leq \mu \leq 1
\]
Matrices with Orthonormal Columns

- \( Q = (q_1 \cdots q_m)^T \in \mathbb{R}^{m \times n} \) with \( Q^T Q = I_n \)

- Coherence

\[
\mu = \max_{1 \leq i \leq m} \|q_i\|_2^2 \quad \frac{n}{m} \leq \mu \leq 1
\]

- Singular values of sampled matrix

\[
\sigma_1(SQ) \geq \cdots \geq \sigma_n(SQ)
\]

- Well conditioning of singular values

\[
\| (SQ)^T (SQ) - I_n \|_2 \leq \epsilon \quad \iff \quad \sqrt{1 - \epsilon} \leq \sigma_j(SQ) \leq \sqrt{1 + \epsilon} \quad 1 \leq j \leq n
\]
Singular Values of Sampled Orthonormal Matrices

[Holodnak & Ipsen 2015]

- Failure probability $0 < \delta \leq 1$
- Relative error bound $0 < \epsilon < 1$
- Sampling amount $c \geq \frac{8}{3} m \mu \frac{\ln(n/\delta)}{\epsilon^2}$

With probability $\geq 1 - \delta$

- Simultaneous bound for all singular values

$$\sqrt{1 - \epsilon} \leq \sigma_j(SQ) \leq \sqrt{1 + \epsilon} \quad 1 \leq j \leq n$$
Singular Values of Sampled Orthonormal Matrices

[Holodnak & Ipsen 2015]

- Failure probability $0 < \delta \leq 1$
- Relative error bound $0 < \epsilon < 1$
- Sampling amount $c \geq \frac{8}{3} m \mu \frac{\ln(n/\delta)}{\epsilon^2}$

With probability $\geq 1 - \delta$

- Simultaneous bound for all singular values
  \[ \sqrt{1 - \epsilon} \leq \sigma_j(\mathbf{SQ}) \leq \sqrt{1 + \epsilon} \quad 1 \leq j \leq n \]

- Extreme singular values
  \[ 0 < \sqrt{1 - \epsilon} \leq \sigma_n(\mathbf{SQ}) = 1/\|\mathbf{SQ}^\dagger\|_2 \]
  \[ \|\mathbf{SQ}\|_2 = \sigma_1(\mathbf{SQ}) \leq \sqrt{1 + \epsilon} \]
Singular Values of Sampled Orthonormal Matrices

[Holodnak & Ipsen 2015]

- Failure probability $0 < \delta \leq 1$
- Relative error bound $0 < \epsilon < 1$
- Sampling amount $c \geq \frac{8}{3} m \mu \frac{\ln(n/\delta)}{\epsilon^2}$

With probability $\geq 1 - \delta$

- Simultaneous bound for all singular values
  \[
  \sqrt{1 - \epsilon} \leq \sigma_j(SQ) \leq \sqrt{1 + \epsilon} \quad 1 \leq j \leq n
  \]
- Extreme singular values
  \[
  0 < \sqrt{1 - \epsilon} \leq \sigma_n(SQ) = \frac{1}{\|SQ\|_2} \\
  \|SQ\|_2 = \sigma_1(SQ) \leq \sqrt{1 + \epsilon}
  \]
- Full rank: $\text{rank}(SQ) = n$
Conditioning of Sampled Matrices

- \( Q = (q_1 \cdots q_m)^T \in \mathbb{R}^{m \times n} \)
- \( \mu = \max_{1 \leq i \leq m} \|q_i\|_2^2 \)
- Failure probability \( 0 < \delta \leq 1 \)
- Relative error bound \( 0 < \epsilon < 1 \)
- Sampling amount \( c \geq \frac{8}{3} m \mu \frac{\ln(n/\delta)}{\epsilon^2} \)
- DKM algorithm samples \( c \) rows \( SQ \) with uniform probabilities \( p_j = 1/m \)

With probability \( \geq 1 - \delta \)

\[
\text{rank}(SQ) = n \quad \text{and} \quad \kappa(SQ) = \|SQ\|_2 \| (SQ)^\dagger \|_2 \leq \sqrt{\frac{1 + \epsilon}{1 - \epsilon}}
\]
Strategy for DKM Algorithm

1. Choose sampling matrix $S$ to produce unbiased estimator

$$
\mathbb{E}[(SA)^T (SA)] = A^T A
$$

2. Absolute error equals deviation from the mean

$$
(SA)^T (SA) - A^T A = (SA)^T (SA) - \mathbb{E}[(SA)^T (SA)]
$$

3. Express deviation from mean as sum of matrix-valued (zero-mean) random variables

$$
(SA)^T (SA) - \mathbb{E}[(SA)^T (SA)] = Y_1 + \cdots + Y_c
$$

4. Apply matrix concentration inequality to $Y_1 + \cdots + Y_c$

For relative error use $\hat{\epsilon} = \|A^T A\|_2 \epsilon$
Analysis of DKM Algorithm

- Relative error bound
  \[\|(SA)^T (SA) - A^T A\|_2 \leq \epsilon \|A^T A\|_2\]

- Sampling with uniform probabilities \(p_j = 1/m\)

- Number of samples \(c = \Omega \left( m \mu \frac{\ln n}{\epsilon^2} \right)\) depends on
  - Large dimension \(m\)
  - Row mass \(\mu = \max_j \|a_j\|^2_2/\|A\|^2_2\)

- Orthonormal matrices \(Q\) with \(Q^T Q = I_n\)
  \[\|(SQ)^T (SQ) - I_n\|_2 \leq \epsilon \iff \sqrt{1 - \epsilon} \leq \sigma_j(SQ) \leq \sqrt{1 + \epsilon}\]

- Probability that \(SQ\) has full rank, bounds on condition number
Randomized Least Squares
DMMS Algorithm

[Drineas, Mahoney, Muthukrishnan & Sarlós 2011]
DMMS Algorithm (most basic version)

Given: \( A \in \mathbb{R}^{m \times n} \) with \( \text{rank}(A) = n \) and \( b \in \mathbb{R}^m \)

\[
\min_x \| Ax - b \|_2
\]

Unique solution \( x_o \equiv A^\dagger b \) with residual \( r \equiv Ax_o - b \)
DMMS Algorithm (most basic version)

- **Given:** \( \mathbf{A} \in \mathbb{R}^{m \times n} \) with \( \text{rank}(\mathbf{A}) = n \) and \( \mathbf{b} \in \mathbb{R}^m \)

  \[
  \min_{\mathbf{x}} \| \mathbf{A} \mathbf{x} - \mathbf{b} \|_2
  \]

- **Unique solution** \( \mathbf{x}_o \equiv \mathbf{A}^\dagger \mathbf{b} \) with residual \( \mathbf{r} \equiv \mathbf{A} \mathbf{x}_o - \mathbf{b} \)

- **DMMS algorithm** samples \( c \) rows of \( (\mathbf{A} \ \mathbf{b}) \) independently and identically, with uniform probabilities \( p_j = 1/m \)

  \[
  \mathbf{S} = \sqrt{\frac{m}{c}} \begin{pmatrix} \mathbf{e}_{t_1} & \cdots & \mathbf{e}_{t_c} \end{pmatrix}^T \in \mathbb{R}^{c \times m}
  \]

  \( \mathbf{e}_{t_1}, \ldots, \mathbf{e}_{t_c} \) not necessarily distinct

- **Output** = minimal norm solution of

  \[
  \min_{\tilde{\mathbf{x}}} \| \mathbf{S} (\mathbf{A} \tilde{\mathbf{x}} - \mathbf{b}) \|_2
  \]
Bounding the Error with regard to Randomization

- Perturbed solution \( \tilde{x}_o = (SA)^\dagger (Sb) \)
- Residual \( \tilde{r} = A\tilde{x}_o - b \) with regard to exact problem
- Want: Bounds for relative errors
  \[
  \frac{\|\tilde{x}_o - x_o\|_2}{\|x_o\|_2} \quad \text{and} \quad \frac{\|\tilde{r} - r\|_2}{\|r\|_2}
  \]
Bounding the Error with regard to Randomization

- Perturbed solution $\tilde{x}_o = (SA)^\dagger (Sb)$
- Residual $\tilde{r} = A\tilde{x}_o - b$ with regard to exact problem
- Want: Bounds for relative errors

$$\frac{\|\tilde{x}_o - x_o\|_2}{\|x_o\|_2} \quad \text{and} \quad \frac{\|\tilde{r} - r\|_2}{\|r\|_2}$$

Two-part analysis

1. **Structural part**
   - Multiplicative perturbation bounds for least squares
     \{Push deterministic perturbation analysis as far as possible\}

2. **Probabilistic part**
   - Apply concentration inequality
Preparing for the Structural Part

• Full and thin QR decompositions

\[
A = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R \\ 0 \end{pmatrix} = Q_1 R
\]

where

\[
Q_1^T Q_1 = I_n \quad Q_2^T Q_2 = I_{m-n} \quad Q_1^T Q_2 = 0
\]
Preparing for the Structural Part

- Full and thin QR decompositions

\[ \mathbf{A} = \begin{pmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \end{pmatrix} \begin{pmatrix} \mathbf{R} \\ 0 \end{pmatrix} = \mathbf{Q}_1 \mathbf{R} \]

where

\[ \mathbf{Q}_1^T \mathbf{Q}_1 = \mathbf{I}_n \quad \mathbf{Q}_2^T \mathbf{Q}_2 = \mathbf{I}_{m-n} \quad \mathbf{Q}_1^T \mathbf{Q}_2 = \mathbf{0} \]

- Exact solution and residual

\[ \mathbf{x}_o = \mathbf{R}^{-1} \mathbf{Q}_1^T \mathbf{b} \quad \mathbf{r} = -\mathbf{Q}_2 \mathbf{Q}_2^T \mathbf{b} \]
Preparing for the Structural Part

- Full and thin QR decompositions

\[
A = (Q_1 \quad Q_2) \begin{pmatrix} R \\ 0 \end{pmatrix} = Q_1 R
\]

where

\[
Q_1^T Q_1 = I_n \quad Q_2^T Q_2 = I_{m-n} \quad Q_1^T Q_2 = 0
\]

- Exact solution and residual

\[
x_o = R^{-1} Q_1^T b \quad r = -Q_2 Q_2^T b
\]

- Angle \( \theta \) between \( b \) and \( \text{range}(A) \)

\[
\tan \theta = \frac{\|Q_2^T b\|_2}{\|Q_1^T b\|_2} = \frac{\|r\|_2}{\|Ax_o\|_2}
\]
If \( \text{rank}(SA) = n \) then

\[
\|\hat{x}_o - x_o\|_2 \leq \eta \|A^\dagger\|_2 \|r\|_2
\]

\[
\|\tilde{r} - r\|_2 \leq \eta \|r\|_2
\]

where

\[
\eta = \|\left((SQ_1)^\dagger (SQ_2)\right)\|_2
\]

If also \( A^T b \neq 0 \) then

\[
\frac{\|\hat{x}_o - x_o\|_2}{\|x_o\|_2} \leq \eta \kappa_2(A) \tan \theta
\]
Proof Idea: Change of Variable [DMMS 2011]

- Shift $\tilde{x}_o$ to absolute error $z_o = \tilde{x}_o - x_o$
- Apply to perturbed problem
  \[ \min_{\tilde{x}} \| S (A\tilde{x} - b) \|_2 = \min_z \| S (Az + r) \|_2 \]
- Unique solution
  \[ z_o = -(SA)^\dagger (Sr) = R^{-1} \underbrace{(SQ_1)^\dagger (SQ_2)}_{\eta} (Q_2^T b) \]
- Absolute error bound
  \[ \| \tilde{x}_o - x_o \|_2 = \| z_o \|_2 \leq \| A^\dagger \|_2 \underbrace{\| (SQ_1)^\dagger (SQ_2) \|_2 \| r \|_2}_{\eta} \]
Proof Idea: Change of Variable \([\text{DMMS 2011}]\)

- Shift \(\tilde{x}_o\) to absolute error \(z_o = \tilde{x}_o - x_o\)
- Apply to perturbed problem
  \[
  \min_{\tilde{x}} \| S (A\tilde{x} - b) \|_2 = \min_{z} \| S (Az + r) \|_2
  \]
- Unique solution
  \[
  z_o = -(SA)^\dagger (Sr) = R^{-1} (SQ_1)^\dagger (SQ_2) (Q_2^T b)
  \]
- Absolute error bound
  \[
  \| \tilde{x}_o - x_o \|_2 = \| z_o \|_2 \leq \| A^\dagger \|_2 \| (SQ_1)^\dagger (SQ_2) \|_2 \| r \|_2
  \]
- Residual
  \[
  \tilde{r} - r = Az_o = Q_1 (SQ_1)^\dagger (SQ_2) (Q_2^T b)
  \]
Interpretation of $\eta$

Effect of randomization confined to

$$\eta = \|\tilde{X}_o\|_2 \quad \tilde{X}_o = (S Q_1)^\dagger (S Q_2)$$

- $\tilde{X}_o$ is solution of multiple rhs LS problem

$$\min_{\tilde{X}} \|S (Q_1 \tilde{X} - Q_2)\|_F$$

- Corresponding exact LS problem is

$$\min_X \|Q_1 X - Q_2\|_F$$

with solution $X_o = 0$

$\eta$ quantifies effect of $S$ on $\text{range}(A)$ and $\text{null}(A^T)$
Interpretation of Multiplicative Perturbation Bounds

If \( \text{rank}(SA) = n \) then

\[
\frac{\|\tilde{x}_o - x_o\|_2}{\|x_o\|_2} \leq \eta \kappa_2(A) \tan \theta
\]

\[
\|\tilde{r} - r\|_2 \leq \eta \|r\|_2
\]

- No amplification by \([\kappa_2(A)]^2\)
- Perfect conditioning of residual
- Non-asymptotic bounds

LS problems less sensitive to multiplicative perturbations \( S \) than to general additive perturbations
From Structural to Probabilistic

What to do about $\eta = \left\| (SQ_1)^\dagger (SQ_2) \right\|_2$?

Options

1. $\eta \leq \left\| (SQ_1)^\dagger \right\|_2 \left\| SQ_2 \right\|_2$
   
   Have: Singular value bound for $SQ_1$ ✓
   But: Need to know coherence of $Q_2$
From Structural to Probabilistic

What to do about $\eta = \|(SQ_1)^\dagger (SQ_2)\|_2$?

Options

1. $\eta \leq \|(SQ_1)^\dagger\|_2 \|SQ_2\|_2$
   - Have: Singular value bound for $SQ_1$ ✓
   - But: Need to know coherence of $Q_2$

2. $\eta \leq \left\| \left( (SQ_1)^T (SQ_1) \right)^{-1} \right\|_2 \|SQ_1\|^T (SQ_2)\|_2$
   - Have: Singular value bound for $SQ_1$ ✓
   - But: Need general matrix multiplication bound
From Structural to Probabilistic

What to do about \( \eta = \| (SQ_1)^\dagger (SQ_2) \|_2 \) ?

Options

1. \( \eta \leq \| (SQ_1)^\dagger \|_2 \| SQ_2 \|_2 \)
   - Have: Singular value bound for \( SQ_1 \) ✓
   - But: Need to know coherence of \( Q_2 \)

2. \( \eta \leq \left\| \left( (SQ_1)^T (SQ_1) \right)^{-1} \right\|_2 \| (SQ_1)^T (SQ_2) \|_2 \)
   - Have: Singular value bound for \( SQ_1 \) ✓
   - But: Need general matrix multiplication bound

Here \( \{ \text{see also [DMMS 2011]} \} \)

\[ \eta \leq \| (SQ_1)^\dagger \|_2 \| SQ_2 \|_2 \text{ and assume } \| SQ_2 \|_2 \leq \sqrt{\epsilon} \]
Probabilistic Part: Singular Value Bound for $S\!Q_1$

- $Q_1 \in \mathbb{R}^{m \times n}$ with $Q_1^T Q_1 = I_n$ and $\mu_1 = \max_{1 \leq j \leq m} \|e_j^T Q_1\|^2$

- Failure probability $0 < \delta \leq 1$

- Relative tolerance $0 < \epsilon < 1$

- Sampling amount $c \geq \frac{8}{3} m \mu_1 \frac{\ln(n/\delta)}{\epsilon^2}$

- DKM algorithm samples $c$ rows $S\!Q_1$ with uniform probabilities $p_j = 1/m$

With probability $\geq 1 - \delta$

$$\operatorname{rank}(S\!Q_1) = n \quad \text{and} \quad \|(S\!Q_1)^\dagger\|_2 \leq \frac{1}{\sqrt{1 - \epsilon}}$$
Recall the Ingredients for Least Squares

- \( A \in \mathbb{R}^{m \times n} \) with \( \text{rank}(A) = n \) and \( b \in \mathbb{R}^m \)
- \( \min_x \|Ax - b\|_2 \) has solution \( x_o \equiv A^\dagger b \) with \( r \equiv Ax_o - b \)
- QR decompositions: \( A = (Q_1 \quad Q_2) \begin{pmatrix} R \\ 0 \end{pmatrix} = Q_1 R \)
- Coherence of \( Q_1 \): \( \mu_1 = \max_{1 \leq i \leq m} \|e_i^T Q_1\|_2^2 \)

DMMS algorithm

1. Samples \( c \) rows \( S(A \quad b) \) with uniform probabilities \( p_j = 1/m \)
2. Solves \( \min_{\tilde{x}} \|S(A\tilde{x} - b)\|_2 \)
   - Minimal norm solution \( \tilde{x}_o = (SA)^\dagger (Sb) \)
   - Residual \( \tilde{r} = A\tilde{x}_o - b \) with regard to exact problem
Conclusion for DMMS Algorithm

- Failure probability $0 < \delta \leq 1$
- Relative tolerance $0 < \epsilon < 1$
- Assumption $\| SQ_2 \|_2 \leq \sqrt{\epsilon}$
- Sampling amount $c \geq \frac{8}{3} m \mu_1 \frac{\ln(n/\delta)}{\epsilon^2}$

With probability $\geq 1 - \delta$
DMMS algorithm computes $\text{rank}(SA) = n$

$$\frac{\| \tilde{x}_o - x_o \|_2}{\| x_o \|_2} \leq \kappa_2(A) \tan \theta \sqrt{\frac{\epsilon}{1 - \epsilon}}$$

$$\| \tilde{r} - r \|_2 \leq \sqrt{\frac{\epsilon}{1 - \epsilon}} \| r \|_2$$
Analysis of DMMS Algorithm

- Relative error bounds for $\tilde{x}_o$ and $\tilde{r} = A\tilde{x}_o - b$

- Sampling with uniform probabilities $p_j = 1/m$

- Number of samples $c = \Omega\left( m \mu_1 \frac{\ln n}{\epsilon^2} \right)$ depends on
  
  - Large dimension $m$
  - Coherence $\mu_1 = \max_i \| e_i^T Q_1 \|_2^2$

- LS problems less sensitive to randomization than to general, additive perturbations
Important Points

**Two-part** analysis of error due to randomization

1. **Structural:** Push linear algebra as far as possible
2. **Probabilistic:** Apply concentration inequality

Requirement of matrix concentration inequalities

Output of randomized algorithm

\[ = \text{sum} \] of matrix-valued random variables

Rule of thumb for **uniform sampling:**

Number of required samples depends on

- Large matrix dimension
- Coherence of column space
Implementations of RLA algorithms at small, medium, and large-scale

Michael W. Mahoney

ICSI and Dept of Statistics, UC Berkeley

(For more info, see:
http://www.stat.berkeley.edu/~mmahoney
or Google on “Michael Mahoney”)

ALA 2015

October 2015
Some recent observations

http://www.hpcwire.com/2015/10/19/numerical-algorithms-and-libraries-at-exascale/

Jack Dongarra, in HPC wire. Randomized algorithms for exascale computing:

- "... the other seismic force that’s shaking the road to exascale computing ... is the rise of large-scale data analytics as fundamental for a wide range of application domains."

- "... one of the most interesting developments in HPC math libraries is taking place at the intersection of numerical linear algebra and data analytics, where a new class of randomized algorithms is emerging ..."

- "... powerful tools for solving both least squares and low-rank approximation problems, which are ubiquitous in large-scale data analytics and scientific computing"

- "... these algorithms are playing a major role in the processing of the information that has previously lain fallow, or even been discarded, because meaningful analysis of it was simply infeasible—this is the so called “Dark Data phenomenon.”"

- "The advent of tools capable of usefully processing such vast amounts of data has brought new light to these previously darkened regions of the scientific data ecosystem.”
Some recent observations

http://www.hpcwire.com/2015/10/19/numerical-algorithms-and-libraries-at-exascale/

Jack Dongarra, in HPC wire. Randomized algorithms for exascale computing:

- “... randomized algorithms should be expected to play a major role in future convergence of HPC and data analytics”

- “Randomized algorithms are not only fast, they are, as compared to traditional algorithms, easier to code, easier to map to modern computer architectures, and display higher levels of parallelism.”

- “... they often introduce implicit regularization, which makes them more numerically robust.

- “... they may produce results that are easier to interpret and analyze in downstream applications.”

- “While not a silver bullet for large-scale data analysis problems, random sampling and random projection algorithms may dramatically improve our ability to solve a wide range of large matrix problems, especially when they are coupled with domain expertise.”
Outline

1. Overview of implementation issues
2. Communication issues in LA and RLA
3. High-quality implementations in RAM
4. A theoretical aside: low-rank methods
5. Solving $\ell_2$ regression in RAM
6. Solving $\ell_2$ regression using MPI
7. Solving $\ell_2$ regression in Spark
Lots of implementations

- Numerical linear algebra and scientific computing:
  - high-quality implementations, control over conditioning, precision, etc.,
  - often developed/parameterized for relatively “nice” input

- Machine learning and data analysis:
  - lower-quality implementations, applicable to much less “nice” input
  - conditioning, precision, etc. versus regularization or downstream objectives

- Tygert, Rokhlin, Martinsson, Halko, etc.
  - PCA, LS in (general or specialized) scientific computing applications

- Avron, Maymounkov, Toldeo, Meng, Mahoney, Saunders etc.
  - Blendenpik, LSRN

- Yang, Meng, Mahoney, Gittens, Avron, Bekas, etc.
  - L1 and then L2 regression in parallel/distributed environments
  - Skylark
  - PCA on $\geq$ terabyte sized problems

- Lots more ...
Low-precision versus high-precision issues

Who cares about 10 digits of precision?

- **SC/NLA**: all this RLA stuff is just to get a good starting point for iterations.
- **Everyone else**: low-precision is usually enough, otherwise iterate with some method.

Low versus high precision.

- Low precision ($\epsilon = 0.1$): get a sketch (typically subspace-preserving) and solve the subproblem with a black box
- High precision ($\epsilon = 10^{-10}$): get a sketch (downsampled more aggressively) and construct a preconditioner for the original problem.

Looking forward.

- **Many tradeoffs**: data-aware versus data-oblivious; communication versus computation; one machine versus distributed AEC2 versus supercomputer HPC; coupling with downstream objectives, etc.
- **Lots of theory-practice-gap questions** to be answered
Outline

1. Overview of implementation issues
2. Communication issues in LA and RLA
3. High-quality implementations in RAM
4. A theoretical aside: low-rank methods
5. Solving $\ell_2$ regression in RAM
6. Solving $\ell_2$ regression using MPI
7. Solving $\ell_2$ regression in Spark
Metrics for NLA and RLA implementations

Lots of metrics for implementations:

- Flops (dense versus sparse, etc.)
- Communication (single machine or many machines, HPC vs AEC2, for problem or for algorithm, etc.)
- Numerical stability (forward versus backward stability, etc.)
- Accuracy (objective vs certificate, $\epsilon = 10^{-1}$ vs $\epsilon = 10^{-10}$, etc.)

“Communication” means moving data—either between levels of memory hierarchy or between parallel processors over a network

- Flops is a good metric for small problems that fit in cache.
- Motivation for RLA is “massive data” problems.
- Communication costs orders of magnitude more per operation than a flop.
Communication in NLA

(Ballard, Demmel, Holtz, and Schwartz, 2011.)

- Performance model for sending one “message” of $n$ contiguous words is

$$\alpha + n\beta = \text{latency} + n/\text{bandwidth}$$

where a “message” could be a cache line in a shared memory system, an actual message in a distributed memory system or a disk access.

- Overall performance model (along critical path of an algorithm) is then

$$\gamma F + \beta W + \alpha M = \gamma \#\text{Flops} + \beta \#\text{Words.sent} + \alpha \#\text{Messages}$$

where $\gamma << \beta << \alpha$, with gaps growing over time (technological trends).

- Communication-avoiding LA: identify lower bounds (typically for dense input), and design algorithms that achieve lower bounds.
Communication in RLA

- **So far:** follow same high-level RLA theory as for RAM implementations but worry more about communication

- For very large problems: number of iterations is proxy for communication (Mapreduce versus Spark)

- Important difference: distributed data center (e.g., AEC2) versus parallel shared-memory (e.g., HPC)

- Fun fact: only communication-optimal algorithm for the non symmetric eigenvalue problem is a randomized algorithm

- Caveat: streaming model (developed in TCS for networking applications) is not a particularly good idealization for matrix problems
Large-scale environments and how they scale

- **Shared memory**
  - cores: \([10, 10^3]\) *
  - memory: \([100\text{GB}, 100\text{TB}]\)

- **Message passing**
  - cores: \([200, 10^5]\) †
  - memory: \([1\text{TB}, 1000\text{TB}]\)
  - CUDA cores: \([5 \times 10^4, 3 \times 10^6]\) ‡
  - GPU memory: \([500\text{GB}, 20\text{TB}]\)

- **MapReduce**
  - cores: \([40, 10^5]\) §
  - memory: \([240\text{GB}, 100\text{TB}]\)
  - storage: \([100\text{TB}, 100\text{PB}]\) ¶

- **Distributed computing**
  - cores: \([- , 3 \times 10^5]\) ∥

---

† [http://www.top500.org/list/2011/11/100](http://www.top500.org/list/2011/11/100)
‡ [http://i.top500.org/site/50310](http://i.top500.org/site/50310)
Outline

1. Overview of implementation issues
2. Communication issues in LA and RLA
3. High-quality implementations in RAM
4. A theoretical aside: low-rank methods
5. Solving $\ell_2$ regression in RAM
6. Solving $\ell_2$ regression using MPI
7. Solving $\ell_2$ regression in Spark
Least-squares and low-rank approximation via RLA

Low-rank approximation

- Tygert, Rokhlin, Martinsson, Halko, etc.
- high-quality numerical implementations
- Gaussian projections for structured matrices in scientific computing applications.

Least-squares approximation

- Tygert, Rokhlin: initial preconditioning iterative algorithm with $O(\log(1/\epsilon))$ non-worst-case complexity
- Avron, Maymounkov, and Toledo: Blendenpik (beats LAPACK clock time)
- Meng, Saunders, and Mahoney: LSRN (for parallel environments)
- Yang, Meng, and Mahoney: terabyte-sized (to low, medium, and high precision on nice and not nice data) in Spark

Many other problems, e.g., $\ell_1$ regression, Nystrom/SPSD low-rank approximation, etc.
Outline

1. Overview of implementation issues
2. Communication issues in LA and RLA
3. High-quality implementations in RAM
4. A theoretical aside: low-rank methods
5. Solving $\ell_2$ regression in RAM
6. Solving $\ell_2$ regression using MPI
7. Solving $\ell_2$ regression in Spark
A theoretical aside: low-rank (1 of 2)

(Halko, Martinsson, and Tropp, 2011.)

In scientific computing, goal is to find a good basis for the span of $A$ ...

Input: $m \times n$ matrix $A$, target rank $k$ and over-sampling parameter $p$
Output: Rank-$(k + p)$ factors $U$, $\Sigma$, and $V$ s.t. $A \approx U\Sigma V^T$.

1. Draw a $n \times (k + p)$ Gaussian random matrix $\Omega$.
2. Form the $n \times (k + p)$ sample matrix $Y = A\Omega$.
3. Compute an orthonormal matrix $Q$ s.t. $Y = QQ^T Y$.
4. Form the small matrix $B = Q^T A$.
5. Factor the small matrix $B = \hat{U}\Sigma V^T$.
6. Form $U = Q\hat{U}$.

Prove bounds of the form

$$
\|A - QQ^T A\|_F \leq \left(1 + \frac{k}{p - 1}\right)^{1/2} \left(\sum_{j=k+1}^{\min\{m,n\}} \sigma_j^2\right)^{1/2}
$$

$$
\|A - QQ^T A\|_2 \leq \left(1 + \sqrt{\frac{k}{p - 1}}\right) \sigma_{k+1} + \frac{e\sqrt{k + p}}{p} \left(\sum_{j=k+1}^{\min\{m,n\}} \sigma_j^2\right)^{1/2}
$$

Question: how does one prove bounds of this form?
A theoretical aside: low-rank (2 of 2)

(Boutsidis, Mahoney, Drineas, 2009.)

**Answer:** Basic structural result for RLA low-rank matrix approximation.

**Lemma**

Given $A \in \mathbb{R}^{m \times n}$, let $V_k \in \mathbb{R}^{n \times k}$ be the matrix of the top $k$ right singular vectors of $A$. Let $\Omega \in \mathbb{R}^{n \times r}$ ($r \geq k$) be any matrix such that $Y^T \Omega$ has full rank. Then, for any unitarily invariant norm $\xi$,

$$
\|A - P_{A\Omega}A\|_\xi \leq \|A - A_k\|_\xi + \|\Sigma_{k,\perp} (V_k^T \Omega) (V_k^T \Omega)^+ \|_\xi.
$$

Given this structural result, we obtain results for

- the Column Subset Selection Problem (BMD09)
- using random projections to approximate low-rank matrix approximations (RT10,HMT11,etc.)
- developing improved Nyström-based low-rank matrix approximations of SPSD matrices (GM13)
- developing improved feature selection methods (many)
- other low-rank matrix approximation methods
Outline

1. Overview of implementation issues
2. Communication issues in LA and RLA
3. High-quality implementations in RAM
4. A theoretical aside: low-rank methods
5. Solving $\ell_2$ regression in RAM
6. Solving $\ell_2$ regression using MPI
7. Solving $\ell_2$ regression in Spark
Two important notions: leverage and condition

(Mahoney, “Randomized Algorithms for Matrices and Data,” FnTML, 2011.)

- **Statistical leverage.** (Think: eigenvectors. Important for low-precision.)
  - The **statistical leverage scores** of $A$ (assume $m \gg n$) are the diagonal elements of the projection matrix onto the column span of $A$.
  - They equal the $\ell_2$-norm-squared of any orthogonal basis spanning $A$.
  - They measure:
    - how well-correlated the singular vectors are with the canonical basis
    - which constraints have largest “influence” on the LS fit
    - a notion of “coherence” or “outlierness”
  - Computing them exactly is as hard as solving the LS problem.

- **Condition number.** (Think: eigenvalues. Important for high-precision.)
  - The **$\ell_2$-norm condition number** of $A$ is $\kappa(A) = \sigma_{\text{max}}(A)/\sigma_{\text{min}}^+(A)$.
  - $\kappa(A)$ bounds the number of iterations; for ill-conditioned problems (e.g., $\kappa(A) \approx 10^6 \gg 1$), the convergence speed is very slow.
  - Computing $\kappa(A)$ is generally as hard as solving the LS problem.

These are for the $\ell_2$-norm. Generalizations exist for the $\ell_1$-norm.
Meta-algorithm for $\ell_2$-norm regression (1 of 3)

(Drineas, Mahoney, etc., 2006, 2008, etc., starting with SODA 2006; Mahoney FnTML, 2011.)

1: Using the $\ell_2$ statistical leverage scores of $A$, construct an importance sampling distribution $\{p_i\}_{i=1}^m$.

2: Randomly sample a small number of constraints according to $\{p_i\}_{i=1}^m$ to construct a subproblem.

3: Solve the $\ell_2$-regression problem on the subproblem.

A naïve version of this meta-algorithm gives a $1 + \epsilon$ relative-error approximation in roughly $O(mn^2/\epsilon)$ time (DMM 2006, 2008). (Ugh.)
Meta-algorithm for $\ell_2$-norm regression (2 of 3)

(Drineas, Mahoney, etc., 2006, 2008, etc., starting with SODA 2006; Mahoney FnTML, 2011.)

- Randomly sample high-leverage constraints
- Solve the subproblem

(In many moderately large-scale applications, one uses “$\ell_2$ objectives,” not since they are “right,” but since other things are even more expensive.)
Meta-algorithm for $\ell_2$-norm regression (2 of 3)

(Drineas, Mahoney, etc., 2006, 2008, etc., starting with SODA 2006; Mahoney FnTML, 2011.)

- Randomly sample high-leverage constraints
- Solve the subproblem

(In many moderately large-scale applications, one uses “$\ell_2$ objectives,” not since they are “right,” but since other things are even more expensive.)
Meta-algorithm for $\ell_2$-norm regression (2 of 3)

(Drineas, Mahoney, etc., 2006, 2008, etc., starting with SODA 2006; Mahoney FnTML, 2011.)

- Randomly sample high-leverage constraints
- Solve the subproblem

(In many moderately large-scale applications, one uses “$\ell_2$ objectives,” not since they are “right,” but since other things are even more expensive.)
We can make this meta-algorithm “fast” in RAM:**

- This meta-algorithm runs in $O(mn \log n/\epsilon)$ time in RAM if:
  - we perform a Hadamard-based random projection and sample uniformly in a randomly rotated basis, or
  - we quickly computing approximations to the statistical leverage scores and using those as an importance sampling distribution.

We can make this meta-algorithm “high precision” in RAM:††

- This meta-algorithm runs in $O(mn \log n \log(1/\epsilon))$ time in RAM if:
  - we use the random projection/sampling basis to construct a preconditioner and couple with a traditional iterative method.

** (Sarlós 2006; Drineas, Mahoney, Muthu, Sarlós 2010; Drineas, Magdon-Ismail, Mahoney, Woodruff 2011.)
†† (Rokhlin & Tygert 2008; Avron, Maymounkov, & Toledo 2010; Meng, Saunders, & Mahoney 2011.)
‡‡ (Mahoney, “Randomized Algorithms for Matrices and Data,” FnTML, 2011.)
Randomized regression in RAM: Implementations


Conclusions:

- **Randomized algorithms** “beats Lapack’s direct dense least-squares solver by a large margin on essentially any dense tall matrix.”
- **These results** “suggest that random projection algorithms should be incorporated into future versions of Lapack.”
Outline

1. Overview of implementation issues
2. Communication issues in LA and RLA
3. High-quality implementations in RAM
4. A theoretical aside: low-rank methods
5. Solving $\ell_2$ regression in RAM
6. Solving $\ell_2$ regression using MPI
7. Solving $\ell_2$ regression in Spark
Algorithm LSRN (for strongly over-determined systems)

(Meng, Saunders, and Mahoney 2011)

1: Perform a *Gaussian* random projection
2: Construct a preconditioner from the subsample
3: Iterate with a “traditional” iterative algorithm

Why a *Gaussian* random projection? Since it

- provides the strongest results for conditioning,
- uses level 3 BLAS on dense matrices and can be generated super fast,
- speeds up automatically on sparse matrices and fast operators,
- still works (with an extra “allreduce” operation) when $A$ is partitioned along its bigger dimension.

Although it is “slow” (compared with “fast” Hadamard-based projections i.t.o. FLOPS), it allows for better communication properties.
Implementation of LSRN

(Meng, Saunders, and Mahoney 2011)

- **Shared memory (C++ with MATLAB interface)**
  - Multi-threaded ziggurat random number generator (Marsaglia and Tsang 2000), generating $10^9$ numbers in less than 2 seconds on 12 CPU cores.
  - A naïve implementation of multi-threaded dense-sparse matrix multiplications.

- **Message passing (Python)**
  - Multi-threaded BLAS/LAPACK for SVD.
  - Using the Chebyshev semi-iterative method (Golub and Varga 1961) instead of LSQR.
Solving real-world problems

<table>
<thead>
<tr>
<th>matrix</th>
<th>m</th>
<th>n</th>
<th>nnz</th>
<th>rank</th>
<th>cond</th>
<th>DGELSD</th>
<th>A \ b</th>
<th>Blendenpik</th>
<th>LSRN</th>
</tr>
</thead>
<tbody>
<tr>
<td>landmark</td>
<td>71952</td>
<td>2704</td>
<td>1.15e6</td>
<td>2671</td>
<td>1.0e8</td>
<td>29.54</td>
<td>0.6498*</td>
<td>-</td>
<td>17.55</td>
</tr>
<tr>
<td>rail4284</td>
<td>4284</td>
<td>1.1e6</td>
<td>1.1e7</td>
<td>full</td>
<td>400.0</td>
<td>&gt; 3600</td>
<td>1.203*</td>
<td>OOM</td>
<td>136.0</td>
</tr>
<tr>
<td>tnimg_1</td>
<td>951</td>
<td>1e6</td>
<td>2.1e7</td>
<td>925</td>
<td>-</td>
<td>630.6</td>
<td>1067*</td>
<td>-</td>
<td>36.02</td>
</tr>
<tr>
<td>tnimg_2</td>
<td>1000</td>
<td>2e6</td>
<td>4.2e7</td>
<td>981</td>
<td>-</td>
<td>1291</td>
<td>&gt; 3600*</td>
<td>-</td>
<td>72.05</td>
</tr>
<tr>
<td>tnimg_3</td>
<td>1018</td>
<td>3e6</td>
<td>6.3e7</td>
<td>1016</td>
<td>-</td>
<td>2084</td>
<td>&gt; 3600*</td>
<td>-</td>
<td>111.1</td>
</tr>
<tr>
<td>tnimg_4</td>
<td>1019</td>
<td>4e6</td>
<td>8.4e7</td>
<td>1018</td>
<td>-</td>
<td>2945</td>
<td>&gt; 3600*</td>
<td>-</td>
<td>147.1</td>
</tr>
<tr>
<td>tnimg_5</td>
<td>1023</td>
<td>5e6</td>
<td>1.1e8</td>
<td>full</td>
<td>-</td>
<td>&gt; 3600</td>
<td>&gt; 3600*</td>
<td>OOM</td>
<td>188.5</td>
</tr>
</tbody>
</table>

**Table:** Real-world problems and corresponding running times. DGELSD doesn’t take advantage of sparsity. Though MATLAB’s backslash may not give the min-length solutions to rank-deficient or under-determined problems, we still report its running times. Blendenpik either doesn’t apply to rank-deficient problems or runs out of memory (OOM). LSRN’s running time is mainly determined by the problem size and the sparsity.
Iterating with LSQR

(Paige and Saunders 1982)

Code snippet (Python):

```python
u = A.matvec(v) - alpha*u
beta = sqrt(comm.allreduce(np.dot(u,u)))
...
v = comm.allreduce(A.rmatvec(u)) - beta*v
```

Cost per iteration:
- two matrix-vector multiplications
- two cluster-wide synchronizations
Iterating with Chebyshev semi-iterative (CS) method

(Golub and Varga 1961)

The strong concentration results on $\sigma_{\text{max}}(AN)$ and $\sigma_{\text{min}}(AN)$ enable use of the CS method, which requires an accurate bound on the extreme singular values to work efficiently.

Code snippet (Python):

```python
v = comm.allreduce(A.rmatvec(r)) - beta*v
x += alpha*v
r -= alpha*A.matvec(v)
```

Cost per iteration:

- two matrix-vector multiplications
- one cluster-wide synchronization
## LSQR vs. CS on an Amazon EC2 cluster

(Meng, Saunders, and Mahoney 2011)

<table>
<thead>
<tr>
<th>solver</th>
<th>$N_{\text{nodes}}$</th>
<th>$N_{\text{processes}}$</th>
<th>$m$</th>
<th>$n$</th>
<th>nnz</th>
<th>$N_{\text{iter}}$</th>
<th>$T_{\text{iter}}$</th>
<th>$T_{\text{total}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSRN w/ CS</td>
<td>2</td>
<td>4</td>
<td>1024</td>
<td>4e6</td>
<td>8.4e7</td>
<td>106</td>
<td>34.03</td>
<td>170.4</td>
</tr>
<tr>
<td>LSRN w/ LSQR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>84</td>
<td>41.14</td>
<td>178.6</td>
</tr>
<tr>
<td>LSRN w/ CS</td>
<td>5</td>
<td>10</td>
<td>1024</td>
<td>1e7</td>
<td>2.1e8</td>
<td>106</td>
<td>50.37</td>
<td>193.3</td>
</tr>
<tr>
<td>LSRN w/ LSQR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>84</td>
<td>68.72</td>
<td>211.6</td>
</tr>
<tr>
<td>LSRN w/ CS</td>
<td>10</td>
<td>20</td>
<td>1024</td>
<td>2e7</td>
<td>4.2e8</td>
<td>106</td>
<td>73.73</td>
<td>220.9</td>
</tr>
<tr>
<td>LSRN w/ LSQR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>84</td>
<td>102.3</td>
<td>249.0</td>
</tr>
<tr>
<td>LSRN w/ CS</td>
<td>20</td>
<td>40</td>
<td>1024</td>
<td>4e7</td>
<td>8.4e8</td>
<td>106</td>
<td>102.5</td>
<td>255.6</td>
</tr>
<tr>
<td>LSRN w/ LSQR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>84</td>
<td>137.2</td>
<td>290.2</td>
</tr>
</tbody>
</table>

**Table:** Test problems on an Amazon EC2 cluster and corresponding running times in seconds. Though the CS method takes more iterations, it actually runs faster than LSQR by making only one cluster-wide synchronization per iteration.
Outline

1. Overview of implementation issues
2. Communication issues in LA and RLA
3. High-quality implementations in RAM
4. A theoretical aside: low-rank methods
5. Solving $\ell_2$ regression in RAM
6. Solving $\ell_2$ regression using MPI
7. Solving $\ell_2$ regression in Spark
Distributed setting

“Apache Spark is a fast and general engine for large-scale data processing.”

Here, we assume that dataset is partitioned along the high dimension and stored in a distributed fashion.

The costs of computing in distributed settings:

- floating point operations
- bandwidth costs: $\propto$ total bits transferred
- latency costs: $\propto$ rounds of communication

$$\text{FLOPS}^{-1} \ll \text{bandwidth}^{-1} \ll \text{latency}.$$

We want to make as few passes over the dataset as possible.
Solvers for $\ell_2$ regression

- Low-precision solvers: compute a sketch (MapReduce, 1-pass)
  + solve the subproblem (local SVD)

- High-precision solvers: compute a sketch (MapReduce, 1-pass)
  + preconditioning (local QR)
  + invoke an iterative solver (only matrix-vector products are involved which can be well handled by Spark, $\#$ pass is proportional to $\#$ iteration)

Notes

- Methods for computing sketches are embarrassingly parallel and can be implemented under the MapReduce framework.

- Since the sketch is small, operations like SVD or QR can be performed exactly locally.

- Preconditioning is crucial because we want to reduce the number of iterations in the iterative solver.
Gaussian Transform

Here, $\Phi A = GA$ where $G \in \mathbb{R}^{r \times n}$ consisting of i.i.d. Gaussian random variables:

$$GA = (G_1 \cdots G_p) \begin{pmatrix} A_1 \\ \vdots \\ A_p \end{pmatrix} = \sum_{i=1}^{p} G_i A_i.$$ 

**Mapper:**
1. For a block of $A$, $A_i \in \mathbb{R}^{c \times d}$, generate $G_i \in \mathbb{R}^{r \times c}$ consisting of independent Gaussian random variables.
2. Compute $B = G_i A_i \in \mathbb{R}^{r \times d}$.
3. For $j = 1, \ldots, r$, emit $(j, B_j)$.

**Reducer:**
1. Reduce vectors associated with key $k$ to $v_k$ with addition operation.
2. Return $v_k$.

Other sketches are similar; but lots of work to do to figure out the best in general.
Experimental setup

Sketches

- **PROJ CW** — Random projection with the input-sparsity time CW method (sparse)
- **PROJ GAUSSIAN** — Random projection with Gaussian transform (dense)
- **PROJ RADEMACHER** — Random projection with Rademacher transform (dense)
- **PROJ SRDHT** — Random projection with Subsampled randomized discrete Hartley transform (dense, no longer fast)
- **SAMP APPR** — Random sampling based on approximate leverage scores (fast approximate leverage scores)
- **SAMP UNIF** — Random sampling with uniform distribution (for completeness)
Datasets

- We used synthetic datasets with *uniform* or *nonuniform* leverage scores, *low* or *high* condition number, and *coherent* or *incoherent*.
- Recall that leverage scores can be viewed as a measurement of the importance of the rows in the LS fit.
- Coherence is the largest leverage score.
- These properties of the matrix have a strong influence on the solution quality.

Resources

The experiments were performed in a cluster with 16 nodes (1 master and 15 slaves), each of which has 4 CPU cores at clock rate 2.5GHz with 25GB RAM.
Low-precision solvers: overall evaluation

(a) $\|x - x^*\|_2 / \|x^*\|_2$

(b) $|f - f^*| / f^*$

(c) Running time(sec)

1e7 × 1000 matrix with uniform leverage scores

(d) $\|x - x^*\|_2 / \|x^*\|_2$

(e) $|f - f^*| / f^*$

(f) Running time(sec)

1e7 × 1000 matrix with nonuniform leverage scores

Figure: Evaluation of all 6 of the algorithms on matrices with uniform and nonuniform leverage scores.
Low-precision solvers: on matrices with increasing $n$

\begin{align*}
\text{(a)} & \quad \| x - x^* \|_2 / \| x^* \|_2 \\
\text{(b)} & \quad | f - f^* | / f^* \\
\text{(c)} & \quad \text{Running time (sec)} \\
\text{(d)} & \quad \| x - x^* \|_2 / \| x^* \|_2 \\
\text{(e)} & \quad | f - f^* | / f^* \\
\text{(f)} & \quad \text{Running time (sec)}
\end{align*}

**sketch size $s = 5e3$**

**sketch size $s = 5e4$**

**Figure:** Performance of all the algorithms on matrices with nonuniform leverage scores, high condition number, varying $n$ from $2.5e5$ to $1e8$ and fixed $d = 1000$. 
Low-precision solvers: on matrices with increasing $d$

(a) $\|x - x^*\|_2 / \|x^*\|_2$

(b) $|f - f^*| / f^*$

(c) Running time (sec)

**sketch size $s = 2e3$**

(d) $\|x - x^*\|_2 / \|x^*\|_2$

(e) $|f - f^*| / f^*$

(f) Running time (sec)

**sketch size $s = 5e4$**

**Figure**: Performance of all the algorithms on matrices with *nonuniform* leverage scores, *high* condition number, varying $d$ from 10 to 2000 and fixed $n = 1e7$. 
High-precision solvers: preconditioning quality

<table>
<thead>
<tr>
<th>$r$</th>
<th>PROJ CW</th>
<th>PROJ GAUSSIAN</th>
<th>SAMP APPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>5E2</td>
<td>1.08E8</td>
<td>2.17E3</td>
<td>1.21E2</td>
</tr>
<tr>
<td>1E3</td>
<td>1.1E6</td>
<td>5.7366</td>
<td>75.0290</td>
</tr>
<tr>
<td>5E3</td>
<td>5.5E5</td>
<td>1.9059</td>
<td>25.8725</td>
</tr>
<tr>
<td>1E4</td>
<td>5.1E5</td>
<td>1.5733</td>
<td>17.0679</td>
</tr>
<tr>
<td>5E4</td>
<td>1.8E5</td>
<td>1.2214</td>
<td>6.9109</td>
</tr>
<tr>
<td>1E5</td>
<td>1.1376</td>
<td>1.1505</td>
<td>4.7573</td>
</tr>
</tbody>
</table>

Table: Quality of preconditioning, i.e., $\kappa(AR^{-1})$, on a matrix with size 1e6 by 500 and condition number 1e6 using several kinds of sketch.
High-precision solvers: on badly conditioned matrix

Figure: LSQR with randomized preconditioner on a matrix with size 1e8 by 1000 and condition number 1e6. (Running time is artificially slow and not accurate.)
High-precision solvers: on well conditioned matrix

Figure: LSQR with randomized preconditioner on a matrix with size 1e7 by 1000 and condition number 5.
Conclusions and Future Directions

Conclusions

- Short answer: take the basic RLA theory and couple with traditional NLA methods and you get high-quality implementations
  - “beat” LAPACK in terms of clock time
  - better (more robust, better communication, faster, etc.) in SC apps
  - least-squares approximation, low-rank approximation, etc. on terabyte (soon hundreds of terabytes) sized data in AEC2/HPC environments

- Long answer: is much longer ...

Future Directions.

- Communication-computation tradeoffs
- Numerical precision versus statistical precision tradeoffs
- Interacting in nontrivial ways with large-scale data analysis objectives
- Interacting in nontrivial with traditional/nontraditional continuous optimization
- Developing more relevant theory
- ...

More readings …


