Putting Randomized Matrix Algorithms in LAPACK, and Connections with Second-order Stochastic Optimization

Michael W. Mahoney

ICSI and Department of Statistics, UC Berkeley

November 2021
This work

It’s part of the larger *BALLISTIC* collaboration, with

Jim Demmel, Jack Dongarra, Mark Gates, Julie Langou, Julien Langou, Piotr Luszczek, and Riley Murray.

Recent contributors to the randomized linear algebra aspects of BALLISTIC include

Riley Murray, Jim Demmel, Laura Grigori, Ben Erichson, Michał Dereziński, Vivek Bharadwaj, Max Melnichenko, Hengrui Luo, Younghyun Cho, Haoyun Li, and me.

Beyond this talk, we’ll soon be sharing

- a much more detailed design document.
- a Python library, to illustrate design principles and facilitate experimentation.
What is LAPACK?

LAPACK (Linear Algebra PACKage)
- standard software library for numerical linear algebra
- routines for systems of linear equations and linear least squares, eigenvalue problems, and SVD
- also routines to implement associated matrix factorizations, LU, QR, Cholesky and Schur, etc.

“If you call a linear algebra routine in python, R, etc. . . . then you probably call something that calls something that calls LAPACK.”

BLAS (Basic Linear Algebra Subprograms)
- a specification that prescribes a set of low-level routines for common linear algebra operations
- vector addition, scalar multiplication, dot products, linear combinations, and matrix multiplication
- the de facto standard low-level routines for linear algebra libraries
What is randomized numerical linear algebra?

- early work from TCS achieved weak additive-error bounds for low-rank approximation [1, 2, 3, 4]
- relative-error guarantees for least-squares, low-rank using “subspace information” [5, 6, 7, 8]
- use as a preconditioner for iterative algorithms for least-squares [9, 10, 11]
- use two-step procedure for good low-rank approximation [12, 13]
- ... 
- use (SubSampled Newton, Iterative Hessian sketch, etc.) for convex optimization [14, 15]
- ... 
- stochastic second-order optimizers can beat first-order variants optimized for CV, NLP, RecSys AI/ML [16]
- ... 
- time to put these methods into LAPACK!
Randomized numerical linear algebra (RNLA)

Using randomized algorithms to solve deterministic problems.

For example: \[ \min_x \|Ax - b\|_2^2 \]

The algorithms use randomness *internally*

- Rely on a black-box random number generator.
- The generator needn’t be very high-quality.

The algorithms gamble with solution quality and/or computational cost

- Quality and cost vary from one run to another.
- Many RNLA algorithms have extremely small variations in performance.

Reviews from different perspectives: [17, 12, 18, 19, 20, 21, 22, 13, 23]
What does randomization buy us?

- Efficient algorithms for computing **approximate solutions**
  
  *Whole areas.* E.g., low-rank approximation [12], convex optimization [24].

- Efficient algorithms for computing **machine-precision solutions**
  
  *Specific problems.* E.g., strongly overdetermined least squares [9, 10, 11], block column-pivoted QR [25, 26].

- Robust algorithms for **intractable problems**
  
  E.g., nonnegative matrix factorization [27], interpolative decomposition [28]

- Solving problems under data-privacy constraints [29, 30, 31, 32]

Reviews from different perspectives: [17, 12, 18, 19, 20, 21, 22, 13, 23]
Two ingredients of RNLA algorithms

**Random sketching**

For overdetermined least squares with data \((A, b)\), obtain *sketched* data

\[
\hat{A} = S \tilde{A}, \quad \hat{b} = S \tilde{b}.
\]

**High-level deterministic NLA**

Next, solve the sketched problem

\[
\min_{\hat{x}} \| S (A \hat{x} - b) \|_2^2.
\]

For example, by SVD

\[
\hat{A} = U \Sigma V^T, \\
\Rightarrow \hat{x} = V \Sigma^\dagger U^T \tilde{b}.
\]
An architecture in two parts

Randomized LAPACK will be written in C++ and build on LAPACK++.

- **Configurable object-oriented API** and simplified procedural API
- Data model to focus on dense matrices in shared-memory.
- Accommodate sparse/abstract matrices with "linear operator" objects.

The Randomized BLAS will handle sketching dense data matrices.

- **Procedural API only**
- Hide all details of the random number generator (but preserve reproducibility)
- Support sketching operators drawn from a variety of distributions
- Opportunities to reorganize computation for big performance gains
Two regimes for sketching

Sketching can look like *embedding* or like *sampling*.

\[ \hat{A} = S \quad A \]

\[ Y = A \quad S \]

Distinguished by *relative sizes* of \((S, A)\).
A selection of sketching operators

Let’s say the operator is $d \times m$.

- **Dense iid Gaussian**

- **Haar matrices**: uniform over $d \times m$ matrices with orthonormal cols (or rows)

- **SRTTs**: subsampled randomized trig transforms. For $d \leq m$
  \[
  S = \underbrace{(\text{subsampling})}_{d \times m} \underbrace{(\text{fast trig transform})}_{\text{e.g., DFT or DCT}} \underbrace{(\text{diag(uniform } \pm 1))}_{m \times m}
  \]

- **Row sampling / column sampling**

- **SJLTs**: sparse Johnson-Lindenstrauss transforms.
  
  Example: $k$ independent uniform $\pm 1$’s per column (all others zero).
Two ways to structure an API for basic sketching

The straightforward approach:

1. Generate all random numbers which define $S$.
2. Invoke a deterministic algorithm to compute $SA$ or $AS$.

A more sophisticated approach:

- Generate pieces of $S$ on-the-fly while computing $SA$ or $AS$.
- Let ourselves store pieces of $S$ in cache, but not main memory.
- Discard / regenerate pieces of $S$ as needed.

To what extent will the Randomized BLAS support each approach?
Multi-sketching

1. Row-streaming sketch: [33, 34]
   \[ Y_1 = AS \quad \text{and} \quad Y_2 = A^T Y_1 \]

2. Double-sketch: [12, 33, 35]
   \[ Y_1 = AS_1 \quad \text{and} \quad Y_2 = S_2 A \]

3. Triple-sketch (four operators): [35, 36]
   \[ Y_1 = AS_1, \quad Y_2 = S_2 A, \quad \text{and} \quad Y_3 = S_3 A S_4. \]

What combinations of sketching operator distributions should the Randomized BLAS support for multi-sketching?
Levels in the Randomized BLAS

Levels 1 – 3 produce sketches. Possible organizations:

- One sketch at Level 1, two at Level 2, three or more at Level 3.
- One sketching operator at Level 1, two at Level 2, three or more at Level 3.

<table>
<thead>
<tr>
<th>Special examples</th>
<th>((AS, A^TAS))</th>
<th>((SA, Sb))</th>
<th>(S_1AS_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level by # sketches</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Level by # operators</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

**Level 0**: generate defining data for a sketching operator.
Least squares problems . . . and optimization

Data matrix $A$ is $m \times n$ and *tall* $(m \gg n)$.

**Overdetermined least squares**

\[
\min_{x \in \mathbb{R}^n} \|Ax - b\|_2^2
\]

**Underdetermined least squares**

\[
\min_{y \in \mathbb{R}^m} \|y\|_2^2 \quad \text{subject to} \quad A^T y = c.
\]

Randomized LAPACK:

- take a “primal-dual” perspective on these problems.
- include methods for solving to any desired accuracy.
- facilitate more general second-order optimization algorithms.
A saddle point perspective

Consider a simple saddle point system

\[
\begin{bmatrix}
I & A \\
A^T & 0 - H
\end{bmatrix}
\begin{bmatrix}
y \\
x
\end{bmatrix} =
\begin{bmatrix}
b \\
c
\end{bmatrix}.
\]  
(1)

Equation 1 (with \( H = 0 \)) characterizes optimal solutions to the primal-dual pair

\[
\begin{align*}
\min_{x \in \mathbb{R}^n} & \quad \|Ax - b\|_2^2 + 2c^T x \\
\min_{y \in \mathbb{R}^m} & \quad \|y - b\|_2^2 \quad \text{subject to} \quad A^T y = c.
\end{align*}
\]

Encounter sequences of saddle point systems in ...

- \( \ell_p \) regression for \( p \in (1, 2) \)
- minimizing a composite convex function via Newton’s method
- Interior-point methods for quadratic linear programming when \( H \) is psd

Michael W. Mahoney
A framework for saddle point systems

Problem data $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$.

1. If $m \gtrsim n$ (as opposed to $m \gg n$), call LAPACK instead. # tuning problem
2. Decide the distribution for $S \in \mathbb{R}^{d \times m}$ # tuning problem
3. Sketch $\hat{A}, \hat{b} = S[A, b]$
4. Factor $U, \Sigma, V^T = \text{svd}(\hat{A})$
5. sketch-and-solve: construct a solution to the sketched problem [8, 37].
6. Optional sketch-and-precondition:
   - Form the preconditioned linear operator $A \leftarrow A (V \Sigma^\dagger)$
   - Apply an iterative solver to Equation 1, with sketch-and-solve initialization
   - Previously used for least squares [10, 11] and linear programming [38].
Overdetermined least-squares example

Fixed data matrix:
\[ A \in \mathbb{R}^{100,000 \times 2,000} \]
\[ \text{cond}(A) = 100,000 \]

Fixed target vector:
\[ \| AA^\dagger b \|_2 = 0.95 \| b \|_2 \]

LAPACK time in seconds:
- GELSD: 26.3
- GELSS: 45.6

Laptop w/ Core i7-1065G7
- Battery power

Normal equation error vs time in seconds

Ten trials with SRCT \( S \in \mathbb{R}^{6,000 \times 100,000} \)
Low-rank approximation

Produce a suitably factored representation of a low-rank matrix $\hat{A}$, which stands in as an approximation for a target matrix $A$.

How to measure the quality of an approximation?

- Distance from the target $\|A - \hat{A}\|$
- Distance from an “optimal” approximation $\|A_* - \hat{A}\|$.

Algorithms in Randomized LAPACK

- can accept parameter $k$, produce $\hat{A}$ where $\text{rank} \, \hat{A} = \min\{k, \text{rank} \, A\}$.
- can (in some cases!) accept $\epsilon$ and ensure $\|A - \hat{A}\| \leq \epsilon$.
- come with theoretical guarantees for bounding $\|A - \hat{A}\|$ and/or $\|A_* - \hat{A}\|$.
Some representations for the approximation matrix

- **Singular value decomposition**
  \[ \hat{A} = \sum_{i=1}^{k} \sigma_i u_i v_i^T \]

- **Symmetric / Hermitian eigenvalue decomposition**
  \[ \hat{A} = \sum_{i=1}^{k} \lambda_i v_i v_i^T \]

- **Interpolative decompositions (row ID, column ID, two-sided ID)**
  \[ \hat{A} = CX, \text{ for } C = k \text{ columns of } A, \text{ suitable } X \]

- **CUR decompositions**
  \[ \hat{A} = CUR, \text{ for } R = k \text{ rows of } A, C \text{ as above, any } k \times k \text{ matrix } U \]

- **Nonnegative factorization**
  \[ \hat{A} = WH \text{ for } W \in \mathbb{R}^{m \times k}_+ \text{ and } H \in \mathbb{R}^{k \times n}_+ \]
A standard algorithm for randomized SVD

From [12]:

1. $Q = \text{orth}(AS)$
2. $B = Q^T A$ # Implicitly, $\hat{A} = QB = QQ^T A$.
3. $U, \Sigma, V^T = \text{svd}(B)$
4. $U = QU$ # Implicitly, $\hat{A} = U\Sigma V^T$.
5. return $(U, \Sigma, V^T)$

Many variations!

1. The sketching operator $S$ can be “data-aware.” (Leverage \textit{power iteration}.)
2. Alternative constructions of $(Q, B)$
   - Use only a single pass over $A$
   - Construct in blocks: add columns to $Q$ and rows to $B$.
   - Monitor $\|A - QB\|$ (typically Frobenius norm) as a stopping criterion.
Algorithms for low-rank approximation

- **Singular value decomposition**
  - QB algorithms, single-pass triple-sketch, row-extraction algorithms.

- **Symmetric / Hermitian eigenvalue decomposition**
  - QB algorithms, Nyström for psd matrices, row-extraction algorithms.

- **Interpolative decompositions**
  - “Carry over” algorithm, skeleton + pseudo-inverse algorithm.

- **CUR decompositions**
  - Convert any two-sided ID.

- **Nonnegative factorization**
  - QB-backed hierarchical alternating least-squares.
Feedback is welcome!

Many big-picture questions surround the Randomized BLAS.

One possible organization of development priorities for Randomized LAPACK:

- **Least squares and optimization**
  - Phase 1: saddle point solvers with $H = \delta I$.
  - Later: facilitate sequences of saddle point solves, support more general $H$.

- **Low-rank approximation**
  - Phase 1: QB and QB-backed algorithms.
  - Phase 2: Interpolative and CUR decompositions, Nyström approximations.
  - Later: algorithms focused almost exclusively on speed.

- **Full-rank factorizations.** ("Phase" uncertain.)

Stay tuned for a design document on the Randomized BLAS / Randomized LAPACK, and an associated Python package.
References I

Fast Monte-Carlo algorithms for finding low-rank approximations.

Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication.

Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix.

Fast Monte Carlo algorithms for matrices III: Computing a compressed approximate matrix decomposition.

Sampling algorithms for $\ell_2$ regression and applications.

Improved approximation algorithms for large matrices via random projections.
References II


References III

Randomized numerical linear algebra: Foundations and algorithms.

Sub-sampled Newton methods.

Newton Sketch: A Near Linear-Time Optimization Algorithm with Linear-Quadratic Convergence.

ADAHESSIAN: An adaptive second order optimizer for machine learning.

*Randomized algorithms for matrices and data.*

[18] David P. Woodruff.
Sketching as a tool for numerical linear algebra.
References IV

Lecture notes on randomized linear algebra.

RandNLA: Randomized numerical linear algebra.

Implementing randomized matrix algorithms in parallel and distributed environments.

[22] P. Drineas and M. W. Mahoney.
Lectures on randomized numerical linear algebra.

Determinantal point processes in randomized numerical linear algebra.

Iterative Hessian Sketch: Fast and Accurate Solution Approximation for Constrained Least-Squares.
References V

Randomized qr with column pivoting.


Randomized nonnegative matrix factorization.

[28] Sergey Voronin and Per-Gunnar Martinsson.
Efficient algorithms for CUR and interpolative matrix decompositions.

[29] Raghavendran Balu and Teddy Furon.
Differentially private matrix factorization using sketching techniques.

Privacy at scale.

[31] Seung Geol Choi, Dana Dachman-soled, Mukul Kulkarni, and Arkady Yerukhimovich.
Differentially-private multi-party sketching for large-scale statistics.
References VI

[32] Nitish Mital, Cong Ling, and Deniz Gunduz.

[33] Wenjian Yu, Yu Gu, Jian Li, Shenghua Liu, and Yaohang Li.
Single-pass pca of large high-dimensional data.
In Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, pages 3350–3356, 2017.

[34] Wenjian Yu, Yu Gu, and Yaohang Li.
Efficient randomized algorithms for the fixed-precision low-rank matrix approximation.

Practical sketching algorithms for low-rank matrix approximation.

Pass-efficient randomized algorithms for low-rank matrix approximation using any number of views.

Fast approximation of matrix coherence and statistical leverage.
References VII

Faster randomized infeasible interior point methods for tall/wide linear programs.

The powerurv algorithm for computing rank-revealing full factorizations, 2018.

Randutv: A blocked randomized algorithm for computing a rank-revealing utv factorization.

Blocked rank-revealing qr factorizations: How randomized sampling can be used to avoid single-vector pivoting, 2015.

[42] Christopher Melgaard and Ming Gu.
Gaussian elimination with randomized complete pivoting, 2015.

[43] Neil Lindquist, Piotr Luszczek, and Jack Dongarra.
Replacing pivoting in distributed gaussian elimination with randomized techniques.
Full-rank factorizations in Randomized LAPACK

- Overparameterized interpolative decomposition. E.g., $A = ZR$ where $R \in \mathbb{R}^{d \times n}$ has $d$ rows of $A$ and $m \gg d > n$.
- UTV (aka URV and QLP) [39, 40]
- QR with column pivoting [25, 41].
- LU [42] and [43].
- Symmetric indefinite systems (superset of saddle point systems) [44]
A framework for saddle point systems (detailed)

Problem data $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$.

1. Decide the distribution for $S \in \mathbb{R}^{d \times m}$  # tuning problem
2. Sketch $[\hat{A}, \hat{b}] = S[A, b]$  # e.g., $O(mn \log n)$ with SRTTs
3. Factor $Q, R = qr(\hat{A})$  # $O(dn^2)$.
4. sketch-and-solve: construct a solution to the sketched problem.
   - When $c = 0$, set $\hat{x} = R^{-1}Q^T\hat{b}$ [8]
   - We’re working on $c \neq 0$; one option in [37]
5. Optional sketch-and-precondition:
   - Form the preconditioned linear operator $A \leftarrow AR^{-1}$
   - Apply an iterative solver to Equation 1, with sketch-and-solve initialization
     - Conjugate gradients, LSQR, Chebyshev semi-iterative method
     - Run for a constant number of iterations (e.g., 60 iterations).
   - Previously used for least squares [10, 11] and linear programming [38].
Overdetermined least-squares example (wall power)

Fixed data matrix:

\[
A \in \mathbb{R}^{100,000 \times 2,000}
\]

\[
\text{cond}(A) = 100,000
\]

Fixed target vector:

\[
\|AA^\dagger b\|_2 = 0.95\|b\|_2
\]

LAPACK time in seconds:

- GELSD: 17.3
- GELSS: 34.1

Core i7-1065G7 (Wall power)

Ten trials with SRCT \( S \in \mathbb{R}^{6,000 \times 100,000} \)
Big questions for the Randomized BLAS

1. What pairs of functions will we offer for separately generating a sketching operator and applying such an operator?
   - Makes testing and initial development easier.
   - Probably be expected by many users.
   - Requires exposing a larger API (particularly for sparse sketching operators).
   - Will miss out on some opportunities for more efficient algorithms.

2. What individual functions will we offer that compute a sketch without accepting a sketching operator as input or returning one as output?
   - Gives us space to concoct extremely efficient non-obvious implementations.
   - Exposes a smaller API.
   - Makes testing harder.
   - Creates problems for algorithms that need the same operator several times.
Big questions for the Randomized BLAS

3. What combinations of sketching operators will be allowed with multi-sketching?
   - Multi-sketching is *critical* for single-pass / streaming algorithms.
   - Not practical to make optimized algorithms for all combinations.
   - Multi-sketching has to be exposed through a purely procedural API!

4. How exactly will we support coordinate subsampling as a type of sketching?
   - Applying such an operator is *nominally* trivial, but what about memory layout?
   - Yet another type of sketch for consideration in multi-sketching API.