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 $\underline{\text{randomized algorithms}}$ to solve $\underline{\text{deterministic problems}}$.

For example: $\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$

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]

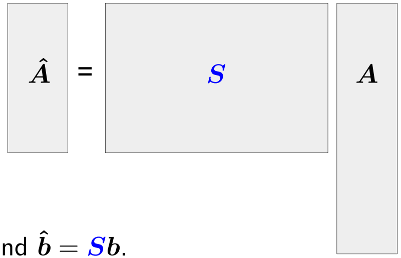
- Solving problems under data-privacy constraints [29, 30, 31, 32]

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 $(\boldsymbol{A}, \boldsymbol{b})$, obtain *sketched* data



and $\hat{\boldsymbol{b}} = \boldsymbol{S}\boldsymbol{b}$.

High-level deterministic NLA

Next, solve the sketched problem

$$\min_{\boldsymbol{x}} \|\boldsymbol{S}\left(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\right)\|_2^2.$$

For example, by SVD

$$\hat{\boldsymbol{A}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathsf{T}},$$

$$\Rightarrow \quad \hat{\boldsymbol{x}} = \boldsymbol{V}\boldsymbol{\Sigma}^{\dagger}\boldsymbol{U}^{\mathsf{T}}\hat{\boldsymbol{b}}.$$

An architecture in two parts

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

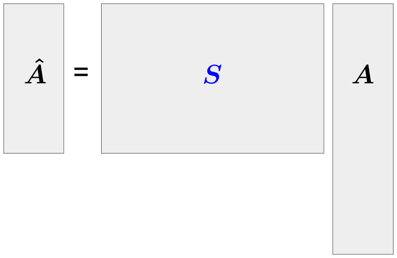
- Configurable object-oriented API and simplified procedural API
- Data model to focus on dense matrices in shared-memory.
- Accommodate sparse/abstract matrices with "linear operator" objects.

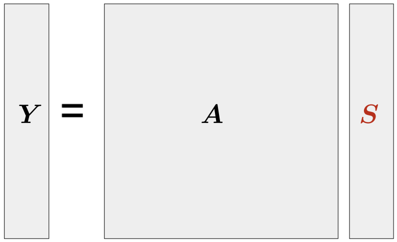
The Randomized BLAS will handle sketching dense data matrices.

- 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*.





Distinguished by *relative sizes* of $(\boldsymbol{S}, \boldsymbol{A})$.

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 $d \leq m$

$$\boldsymbol{S} = \underbrace{\text{(subsampling)}}_{d \times m}\underbrace{\text{(fast trig transform)}}_{\text{e.g., DFT or DCT}}\underbrace{(\text{diag}(\text{uniform } \pm 1))}_{m \times m}$$

- Row sampling / column sampling
- SJLTs: sparse Johnson-Lindenstrauss transforms.

  Example: k independent uniform ± 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 $\boldsymbol{S}$.
2. Invoke a deterministic algorithm to compute $\boldsymbol{SA}$ or $\boldsymbol{AS}$.

A more sophisticated approach:

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

To what extent will the Randomized BLAS support each approach?

Multi-sketching

1. Row-streaming sketch: [33, 34]

$$\boldsymbol{Y}_1 = \boldsymbol{A}\boldsymbol{S} \quad \text{and} \quad \boldsymbol{Y}_2 = \boldsymbol{A}^{\mathsf{T}}\boldsymbol{Y}_1$$

2. Double-sketch: [12, 33, 35]

$$\boldsymbol{Y}_1 = \boldsymbol{A}\boldsymbol{S}_1 \quad \text{and} \quad \boldsymbol{Y}_2 = \boldsymbol{S}_2\boldsymbol{A}$$

3. Triple-sketch (four operators): [35, 36]

$$\boldsymbol{Y}_1 = \boldsymbol{A}\boldsymbol{S}_1, \quad \boldsymbol{Y}_2 = \boldsymbol{S}_2\boldsymbol{A}, \quad \text{and} \quad \boldsymbol{Y}_3 = \boldsymbol{S}_3\boldsymbol{A}\boldsymbol{S}_4.$$

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.

Special examples	$(\boldsymbol{AS}, \boldsymbol{A}^{\mathsf{T}}\boldsymbol{AS})$	$(\boldsymbol{SA}, \boldsymbol{Sb})$	$\boldsymbol{S}_1\boldsymbol{AS}_2$
Level by # sketches	2	2	1
Level by # operators	1	1	2

Level 0: generate defining data for a sketching operator.

Least squares problems . . . and optimization

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

Overdetermined least squares

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$$

Underdetermined least squares

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

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*

$$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{A} \\ \boldsymbol{A}^\mathsf{T} & \boldsymbol{0} - \boldsymbol{H} \end{bmatrix} \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{c} \end{bmatrix}. \tag{1}$$

Equation 1 (with $\boldsymbol{H} = \boldsymbol{0}$) characterizes optimal solutions to the *primal-dual pair*

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + 2\boldsymbol{c}^\mathsf{T}\boldsymbol{x}$$

$$\min_{\boldsymbol{y} \in \mathbb{R}^m} \|\boldsymbol{y} - \boldsymbol{b}\|_2^2 \text{ subject to } \boldsymbol{A}^\mathsf{T}\boldsymbol{y} = \boldsymbol{c}.$$

Encounter sequences of saddle point systems in ...

- ℓ_p regression for $p \in (1, 2)$
- minimizing a composite convex function via Newton's method
- Interior-point methods for quadratic linear programming when $\boldsymbol{H}$ is psd

A framework for saddle point systems

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

1. If $m \gtrsim n$ (as opposed to $m \gg n$), call LAPACK instead. # tuning problem
2. Decide the distribution for $\boldsymbol{S} \in \mathbb{R}^{d \times m}$ # tuning problem
3. Sketch $[\hat{\boldsymbol{A}}, \hat{\boldsymbol{b}}] = \boldsymbol{S}[\boldsymbol{A}, \boldsymbol{b}]$
4. Factor $\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}^{\mathsf{T}} = \mathtt{svd}(\hat{\boldsymbol{A}})$
5. sketch-and-solve: construct a solution to the sketched problem [8, 37].
6. Optional sketch-and-precondition:
   - Form the preconditioned linear operator $\boldsymbol{A} \leftarrow \boldsymbol{A}\left(\boldsymbol{V}\boldsymbol{\Sigma}^{\dagger}\right)$
   - 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:

$$\boldsymbol{A} \in \mathbb{R}^{100,000 \times 2,000}$$

$$\operatorname{cond}(\boldsymbol{A}) = 100,000$$

Fixed target vector:

$$\|\boldsymbol{A}\boldsymbol{A}^{\dagger}\boldsymbol{b}\|_2 = 0.95\|\boldsymbol{b}\|_2$$

LAPACK time in seconds:

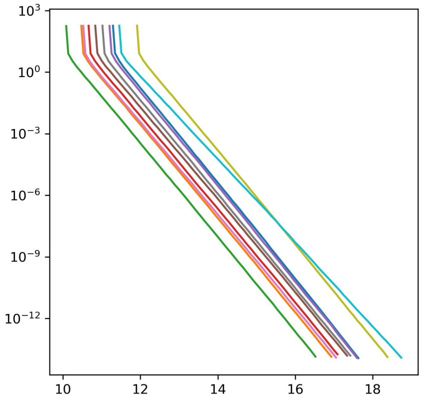
`GELSD`: 26.3

`GELSS`: 45.6

Laptop w/ Core i7-1065G7

Battery power

Normal equation error vs time in seconds



Ten trials with SRCT $\boldsymbol{S} \in \mathbb{R}^{6,000 \times 100,000}$

Low-rank approximation

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

How to measure the quality of an approximation?

- Distance from the target $\|\boldsymbol{A} - \hat{\boldsymbol{A}}\|$
- Distance from an "optimal" approximation $\|\boldsymbol{A}_\star - \hat{\boldsymbol{A}}\|$.

Algorithms in Randomized LAPACK

- can accept parameter k, produce $\hat{\boldsymbol{A}}$ where $\operatorname{rank} \hat{\boldsymbol{A}} = \min\{k, \operatorname{rank} \boldsymbol{A}\}$.
- can (in some cases!) accept ϵ and ensure $\|\boldsymbol{A} - \hat{\boldsymbol{A}}\| \leq \epsilon$.
- come with theoretical guarantees for bounding $\|\boldsymbol{A} - \hat{\boldsymbol{A}}\|$ and/or $\|\boldsymbol{A}_\star - \hat{\boldsymbol{A}}\|$.

Some representations for the approximation matrix

- Singular value decomposition

  $\hat{\boldsymbol{A}} = \sum_{i=1}^{k} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^{\mathsf{T}}$

- Symmetric / Hermitian eigenvalue decomposition

  $\hat{\boldsymbol{A}} = \sum_{i=1}^{k} \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^{\mathsf{T}}$

- Interpolative decompositions (row ID, *column ID*, two-sided ID)

  $\hat{\boldsymbol{A}} = \boldsymbol{C}\boldsymbol{X}$, for $\boldsymbol{C} = k$ columns of $\boldsymbol{A}$, suitable $\boldsymbol{X}$

- CUR decompositions

  $\hat{\boldsymbol{A}} = \boldsymbol{C}\boldsymbol{U}\boldsymbol{R}$, for $\boldsymbol{R} = k$ rows of $\boldsymbol{A}$, $\boldsymbol{C}$ as above, any $k \times k$ matrix $\boldsymbol{U}$

- Nonnegative factorization

  $\hat{\boldsymbol{A}} = \boldsymbol{W}\boldsymbol{H}$ for $\boldsymbol{W} \in \mathbb{R}_{+}^{m \times k}$ and $\boldsymbol{H} \in \mathbb{R}_{+}^{k \times n}$

A standard algorithm for randomized SVD

From [12]:

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

Many variations!

1. The sketching operator $\boldsymbol{S}$ can be "data-aware." (Leverage *power iteration*.)
2. Alternative constructions of $(\boldsymbol{Q}, \boldsymbol{B})$
   - Use only a single pass over $\boldsymbol{A}$
   - Construct in blocks: add columns to $\boldsymbol{Q}$ and rows to $\boldsymbol{B}$.
   - Monitor $\|\boldsymbol{A} - \boldsymbol{QB}\|$ (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 $\boldsymbol{H} = \delta \boldsymbol{I}$.
  - Later: facilitate sequences of saddle point solves, support more general $\boldsymbol{H}$.
- 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

[1] A. Frieze, R. Kannan, and S. Vempala.
Fast Monte-Carlo algorithms for finding low-rank approximations.
Journal of the ACM, 51(6):1025–1041, 2004.

[2] P. Drineas, R. Kannan, and M. W. Mahoney.
Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication.
SIAM Journal on Computing, 36:132–157, 2006.

[3] P. Drineas, R. Kannan, and M. W. Mahoney.
Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix.
SIAM Journal on Computing, 36:158–183, 2006.

[4] P. Drineas, R. Kannan, and M. W. Mahoney.
Fast Monte Carlo algorithms for matrices III: Computing a compressed approximate matrix decomposition.
SIAM Journal on Computing, 36:184–206, 2006.

[5] P. Drineas, M. W. Mahoney, and S. Muthukrishnan.
Sampling algorithms for ℓ_2 regression and applications.
In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1127–1136, 2006.

[6] Tamas Sarlos.
Improved approximation algorithms for large matrices via random projections.
In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '06, page 143–152, USA, 2006. IEEE Computer Society.

References II

[7] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan.
Relative-error CUR matrix decompositions.
SIAM Journal on Matrix Analysis and Applications, 30(2):844–881, January 2008.

[8] P. Drineas, M. W. Mahoney, S. Muthukrishnan, and T. Sarlós.
Faster least squares approximation.
Numerische Mathematik, 117(2):219–249, 2011.

[9] V Rokhlin and M Tygert.
A fast randomized algorithm for overdetermined linear least-squares regression.
Proceedings of the National Academy of Sciences, 105(36):13212–13217, sep 2008.

[10] Haim Avron, Petar Maymounkov, and Sivan Toledo.
Blendenpik: Supercharging LAPACK's Least-Squares Solver.
SIAM Journal on Scientific Computing, 32(3):1217–1236, jan 2010.

[11] Xiangrui Meng, Michael A. Saunders, and Michael W. Mahoney.
LSRN: A parallel iterative solver for strongly over- or underdetermined systems.
SIAM Journal on Scientific Computing, 36(2):C95–C118, January 2014.

[12] N. Halko, P. G. Martinsson, and J. A. Tropp.
Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions.
SIAM Review, 53(2):217–288, January 2011.

References III

[13] Per-Gunnar Martinsson and Joel A. Tropp.
Randomized numerical linear algebra: Foundations and algorithms.
Acta Numerica, 29:403–572, 2020.

[14] F. Roosta-Khorasani and M. W. Mahoney.
Sub-sampled Newton methods.
Mathematical Programming, 174(1-2):293–326, 2019.

[15] Mert Pilanci and Martin J Wainwright.
Newton Sketch: A Near Linear-Time Optimization Algorithm with Linear-Quadratic Convergence.
SIAM Journal on Optimization, 27(1):205–245, jan 2017.

[16] Z. Yao, A. Gholami, S. Shen, K. Keutzer, and M. W. Mahoney.
ADAHESSIAN: An adaptive second order optimizer for machine learning.
Technical Report Preprint: arXiv:2006.00719, 2020.

[17] M. W. Mahoney.
Randomized algorithms for matrices and data.
Foundations and Trends in Machine Learning. NOW Publishers, Boston, 2011.

[18] David P. Woodruff.
Sketching as a tool for numerical linear algebra.
Found. Trends Theor. Comput. Sci., 10(1–2):1–157, October 2014.

References IV

[19] M. W. Mahoney.
Lecture notes on randomized linear algebra.
Technical Report Preprint: arXiv:1608.04481, 2016.

[20] P. Drineas and M. W. Mahoney.
RandNLA: Randomized numerical linear algebra.
Communications of the ACM, 59:80–90, 2016.

[21] J. Yang, X. Meng, and M. W. Mahoney.
Implementing randomized matrix algorithms in parallel and distributed environments.
Proceedings of the IEEE, 104(1):58–92, 2016.

[22] P. Drineas and M. W. Mahoney.
Lectures on randomized numerical linear algebra.
In M. W. Mahoney, J. C. Duchi, and A. C. Gilbert, editors, *The Mathematics of Data*, IAS/Park City Mathematics Series, pages 1–48. AMS/IAS/SIAM, 2018.

[23] M. Derezinski and M. W. Mahoney.
Determinantal point processes in randomized numerical linear algebra.
Notices of the AMS, 68(1):34–45, 2021.

[24] Mert Pilanci and Martin J Wainwright.
Iterative Hessian Sketch: Fast and Accurate Solution Approximation for Constrained Least-Squares.
J. Mach. Learn. Res., 17(1):1842–1879, jan 2016.

References V

[25] Jed A. Duersch and Ming Gu.
Randomized qr with column pivoting.
SIAM Journal on Scientific Computing, 39(4):C263–C291, Jan 2017.

[26] Jed A. Duersch and Ming Gu.
Randomized projection for rank-revealing matrix factorizations and low-rank approximations, 2020.

[27] N Benjamin Erichson, Ariana Mendible, Sophie Wihlborn, and J Nathan Kutz.
Randomized nonnegative matrix factorization.
Pattern Recognition Letters, 104:1–7, 2018.

[28] Sergey Voronin and Per-Gunnar Martinsson.
Efficient algorithms for CUR and interpolative matrix decompositions.
Advances in Computational Mathematics, 43(3):495–516, November 2016.

[29] Raghavendran Balu and Teddy Furon.
Differentially private matrix factorization using sketching techniques.
In *Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security*. ACM, June 2016.

[30] Graham Cormode, Somesh Jha, Tejas Kulkarni, Ninghui Li, Divesh Srivastava, and Tianhao Wang.
Privacy at scale.
In *Proceedings of the 2018 International Conference on Management of Data*. ACM, May 2018.

[31] Seung Geol Choi, Dana Dachman-soled, Mukul Kulkarni, and Arkady Yerukhimovich.
Differentially-private multi-party sketching for large-scale statistics.
Proceedings on Privacy Enhancing Technologies, 2020(3):153–174, July 2020.

References VI

[32] Nitish Mital, Cong Ling, and Deniz Gunduz.
Secure distributed matrix computation with discrete fourier transform, 2020.

[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.
SIAM Journal on Matrix Analysis and Applications, 39(3):1339–1359, January 2018.

[35] Joel A. Tropp, Alp Yurtsever, Madeleine Udell, and Volkan Cevher.
Practical sketching algorithms for low-rank matrix approximation.
SIAM Journal on Matrix Analysis and Applications, 38(4):1454–1485, January 2017.

[36] Elvar K. Bjarkason.
Pass-efficient randomized algorithms for low-rank matrix approximation using any number of views.
SIAM Journal on Scientific Computing, 41(4):A2355–A2383, January 2019.

[37] P. Drineas, M. Magdon-Ismail, M. W. Mahoney, and D. P. Woodruff.
Fast approximation of matrix coherence and statistical leverage.
Journal of Machine Learning Research, 13:3475–3506, 2012.

References VII

[38] Agniva Chowdhury, Palma London, Haim Avron, and Petros Drineas.
Faster randomized infeasible interior point methods for tall/wide linear programs.
In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 8704–8715. Curran Associates, Inc., 2020.

[39] Abinand Gopal and Per-Gunnar Martinsson.
The powerurv algorithm for computing rank-revealing full factorizations, 2018.

[40] P. G. Martinsson, G. Quintana-Ortí, and N. Heavner.
Randutv: A blocked randomized algorithm for computing a rank-revealing utv factorization.
ACM Trans. Math. Softw., 45(1), March 2019.

[41] P. G. Martinsson.
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.
In *2020 IEEE/ACM 11th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*, pages 35–43, 2020.

References VIII

[44] Yuehua Feng, Jianwei Xiao, and Ming Gu.
Randomized complete pivoting for solving symmetric indefinite linear systems.
SIAM Journal on Matrix Analysis and Applications, 39(4):1616–1641, January 2018.

Extra Slides

Full-rank factorizations in Randomized LAPACK

- Overparameterized interpolative decomposition. E.g., $\boldsymbol{A} = \boldsymbol{Z}\boldsymbol{R}$ where $\boldsymbol{R} \in \mathbb{R}^{d \times n}$ has d rows of $\boldsymbol{A}$ 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 $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, and $\boldsymbol{c} \in \mathbb{R}^n$.

1. Decide the distribution for $\boldsymbol{S} \in \mathbb{R}^{d \times m}$ # tuning problem
2. Sketch $[\hat{\boldsymbol{A}}, \hat{\boldsymbol{b}}] = \boldsymbol{S}[\boldsymbol{A}, \boldsymbol{b}]$ # e.g., $O(mn \log n)$ with SRTTs
3. Factor $\boldsymbol{Q}, \boldsymbol{R} = \mathtt{qr}(\hat{\boldsymbol{A}})$ # $O(dn^2)$.
4. sketch-and-solve: construct a solution to the sketched problem.
   - When $\boldsymbol{c} = \boldsymbol{0}$, set $\hat{\boldsymbol{x}} = \boldsymbol{R}^{-1}\boldsymbol{Q}^{\mathsf{T}}\hat{\boldsymbol{b}}$ [8]
   - We're working on $\boldsymbol{c} \neq \boldsymbol{0}$; one option in [37]
5. Optional sketch-and-precondition:
   - Form the preconditioned linear operator $\boldsymbol{A} \leftarrow \boldsymbol{A}\boldsymbol{R}^{-1}$
   - 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:

$$\boldsymbol{A} \in \mathbb{R}^{100,000\times 2,000}$$
$$\operatorname{cond}(\boldsymbol{A}) = 100,000$$

Fixed target vector:

$$\|\boldsymbol{A}\boldsymbol{A}^{\dagger}\boldsymbol{b}\|_2 = 0.95\|\boldsymbol{b}\|_2$$

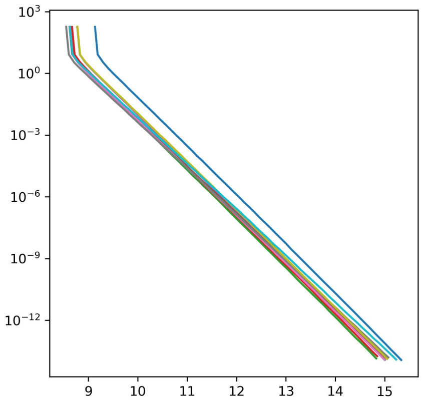
LAPACK time in seconds:

`GELSD`: 17.3

`GELSS`: 34.1

Core i7-1065G7 (Wall power)

Normal equation error vs time in seconds



Ten trials with SRCT $\boldsymbol{S} \in \mathbb{R}^{6,000\times 100,000}$

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.