Possible futures for Randomized Numerical Linear Algebra

Michael W. Mahoney
ICSI, LBNL, and Dept of Statistics, UC Berkeley

May 2026

Overview

Some thoughts on the past

Some thoughts on the present

Some thoughts on the future

Overview

Some thoughts on the past

Some thoughts on the present

Some thoughts on the future

Recent and Upcoming Developments in Randomized Numerical Linear Algebra for ML

Michał Dereziński and Michael W. Mahoney

University of Michigan and ICSI / LBNL / UC Berkeley

December 11, 2023

Outline

Part I
Foundations of RandNLA

1. Initial thoughts
   - Overview
2. Foundations of "classical" RandNLA
   - Matrix Multiplication
   - Least-squares Approximation
   - Low-rank Approximation
3. Foundations of "modern" RandNLA
   - Algorithmic Gaussianization via Random Matrix Theory
   - RMT for Sampling via DPPs

Part II
Recent and Upcoming Advances

4. Advances in RandNLA for Optimization
   - Gradient Sketch
   - Hessian Sketch
   - Sketch-and-Project
5. Advances in RandNLA for ML
   - Statistical Learning Approaches
   - Statistical Inference Approaches
   - Random Matrix Theory Approaches
6. Putting Randomness into LAPACK
   - RandBLAS/RandLAPACK
7. Concluding thoughts

RandNLA: Randomized Numerical Linear Algebra

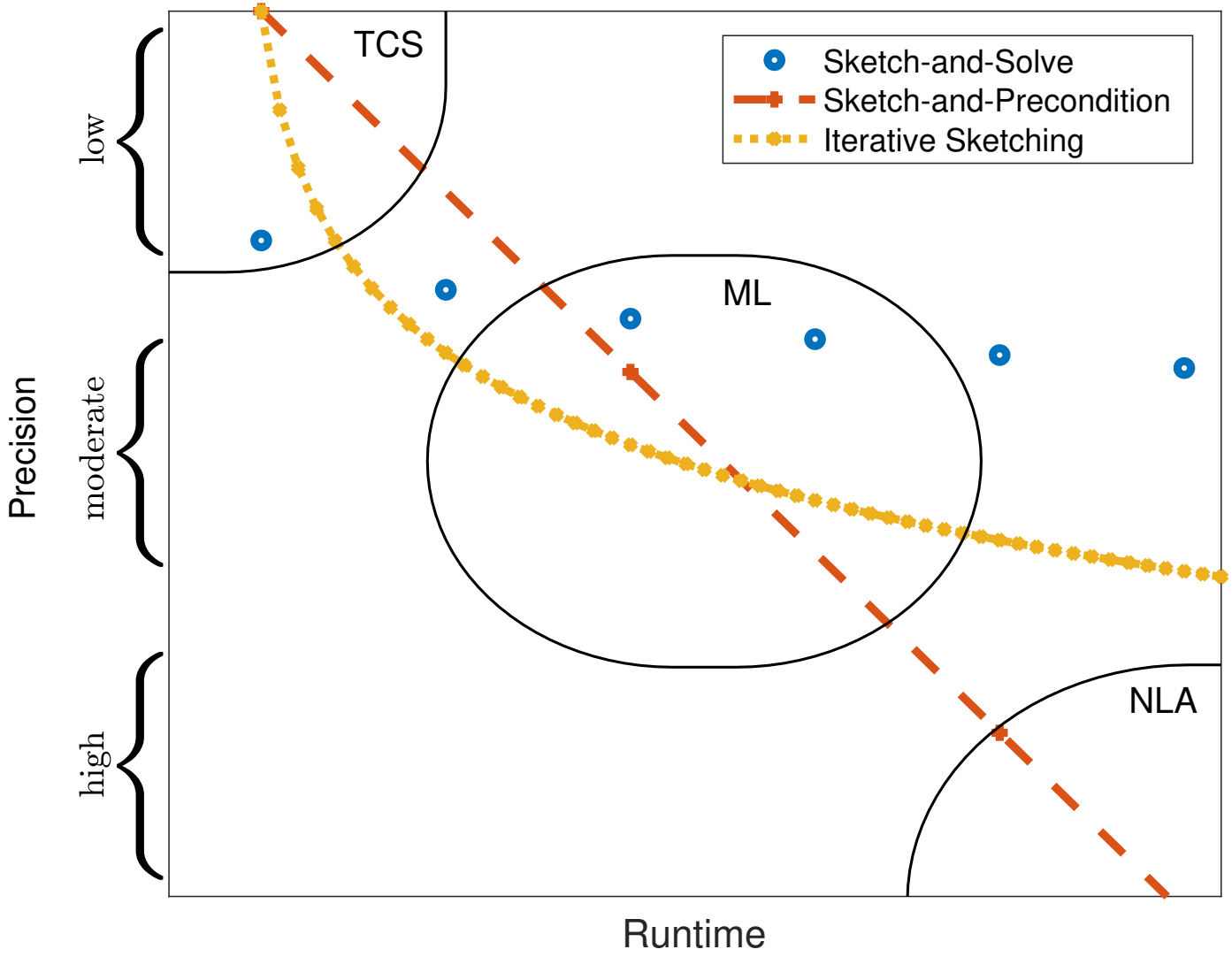
- **"Classical" RandNLA**:
  - Sample/project and then solve subproblem or construct preconditioner
  - Theory from TCS/NLA, typically based on JL / subspace embeddings
  - Lots of data/ML and scientific computing applications
  - Initial proof-of-principle implementations (low-rank approximation, least-squares, optimization, etc.)
  - Relatively large theory-practice gap (esp. when used in ML pipelines)
- **"Modern" RandNLA**:
  - More sophisticated theory going beyond worst-case JL / subspace embeddings, with stronger connections to RMT
  - Improved statistical analysis and improved optimization algorithms
  - Implementations in **RandBLAS/RandLAPACK**, and more demands from GPU-based ML model training and scientific computing
  - Smaller theory-practice gap
- Opens up door to new theory, new implementations, new applications, ...

Using RandNLA methods more generally ...

Three paradigms that apply more broadly than least squares:

1. Sketch-and-solve: Construct a *smaller* least squares problem; then solve it using a direct method.
   - Low-precision estimate, e.g., $\epsilon = 0.1$
   - Simplest to highlight structure of the theory
2. Iterative sketching: Repeatedly sketch/sub-sample the problem; and iteratively refine the estimate.
   - Medium (to high, depending on method) precision estimate, e.g., $\epsilon = 10^{-3}$
   - SGD, SGD++, sketch-and-project, preconditioned weighted SGD
3. Sketch-and-precondition: Construct an *equivalent* but well-conditioned problem; then use a deterministic iterative method.
   - High-precision solution, e.g., $\epsilon = 10^{-10}$
   - Best (usually) for high-quality numerical solutions

Using RandNLA methods more generally ...



Part I: Foundations of RandNLA

1. Initial thoughts
   - Overview
2. Foundations of "classical" RandNLA
   - Matrix Multiplication
   - Least-squares Approximation
   - Low-rank Approximation
3. Foundations of "modern" RandNLA
   - Algorithmic Gaussianization via Random Matrix Theory
   - RMT for Sampling via DPPs

The proportional limit

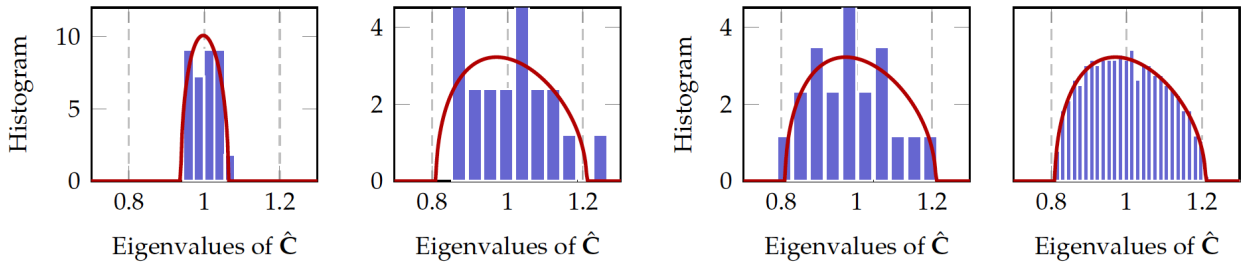


Figure: Histogram of the eigenvalues of $\hat{\mathbf{C}}$ (**blue**) versus the Marčenko-Pastur law (**red**), for $\mathbf{X}$ having standard Gaussian entries in different settings: (**left**: small versus large dimensional intuition) $p = 20, n = 1\,000p$ versus $p = 20, n = 100p$; and (**right**: non-asymptotic versus asymptotic MP law) $p = 20, n = 100p$ versus $p = 500, n = 100p$.

Consider $A \in \mathbb{R}^{n \times d}$ and iid Gaussian sketching matrix $S \in \mathbb{R}^{l \times d}$

Quality of $\tilde{A} = SA$ is often measured by $\mathrm{cond}(SU)$ for $U = \texttt{orth}(A)$ (e.g., subspace embedding, quality of a preconditioner, etc.)

Thanks to the rotation invariance of Gaussian distribution, SU is also Gaussian, so we can use the Marchenko-Pastur law:

$$\sigma_{\min}(SU) \sim 1 - \sqrt{\frac{d}{l}}, \qquad \sigma_{\max}(SU) \sim 1 + \sqrt{\frac{d}{l}}$$

Question: Can we obtain similar results with non-Gaussian sketches?

RMT analysis in RandNLA

Consider sketching matrix $S \in \mathbb{R}^{l \times n}$ with iid Gaussian entries.

- Sketch-and-precondition: Construct R^{-1} from the QR of SA

$$\mathrm{cond}(AR^{-1}) \leq 6 \quad \text{with high probability for} \quad l \geq 2d.$$

- Sketch-and-solve: $\hat{x} = \mathrm{argmin}_x \, \|S(Ax - b)\|_2^2$

$$\mathbb{E}\|A(\hat{x} - x^*)\|_2^2 = \frac{d}{l - d - 1}\|Ax^* - b\|_2^2 \quad \text{for} \quad l \geq d + 2.$$

- Low-rank approximation: Compute $Q = \texttt{orth}(AS)$

$$\mathbb{E}\|A - QQ^\top A\|_F^2 \leq \left(1 + \frac{k}{l - k - 1}\right) \cdot \|A - A_k\|_F^2 \quad \text{for} \quad l \geq k + 2.$$

These are all easy to show for iid Gaussian matrices.

Inversion bias: the key challenge [DM19, DLDM21]

Given $n \times d$ data matrix A of rank d, where $n \geq d$,

approximate $F((A^\top A)^{-1})$, where $F(\cdot)$ is a linear functional.

- $(A^\top A)^{-1}b$, for a vector b:
  - Is the OLS solution (multivariate statistical analysis, Newton's method in numerical optimization, etc.)
- $x^\top (A^\top A)^{-1}x$, for a vector x:
  - If $x = a_i$ is one of the rows of A, then it is leverage scores
  - If $x = \mathbf{e}_i$ is a standard basis vector, then this is the squared length of the confidence interval for the i-th coefficient in OLS
- $\operatorname{tr} C(A^\top A)^{-1}$ for a matrix C:
  - Used to quantify uncertainty
  - Used for experimental design criteria, e.g., A-designs and V-designs

Inversion bias: $\mathbb{E}[(\tilde{A}^\top \tilde{A})^{-1}] \neq (A^\top A)^{-1}$, even though $\mathbb{E}[\tilde{A}^\top \tilde{A}] = A^\top A$

Why focus on the inverse?

- Consider $S \in \mathbb{R}^{l \times n}$ having i.i.d. zero-mean rows statistically.
- $A^\top S^\top S A$ is a *sample covariance estimator* of the "population covariance matrix" $A^\top A \in \mathbb{R}^{d \times d}$.
- How does the spectrum differ between *sample* and *population* covariance?
- RMT answers this by looking at the *resolvent matrix*:

$$(A^\top S^\top S A - zI)^{-1} \quad \text{for} \quad z \in \mathbb{C} \setminus \mathbb{R}_+.$$

- The Stieltjes transform (normalized trace of the resolvent) exhibits *inversion bias*, leading to discrepancy between sample and population.
- Traditional RMT studies limiting eigenvalue distribution as $l, n, d \to \infty$.
- Our goal: *precise* and *non-asymptotic* results on resolvent matrices for sketching, e.g., $(A^\top S^\top S A)^{-1}$, leading to RMT analysis for RandNLA.

Correcting the bias (for Gaussian sketching matrices)

Consider $\hat{H} = \tilde{A}^\top \tilde{A} \approx A^\top A = H$,

(where $\tilde{A} = SA$ is an $l \times d$ sketch of an $n \times d$ matrix A)

Simple correction for a Gaussian sketching matrix S:

- Rescale by a dimensional factor: $\mathbb{E}\left[(\gamma \hat{H})^{-1}\right] = H^{-1}$ for $\gamma = \frac{l}{l-d-1}$

This is **not** true for other sketching methods. Other sketches:

- are *not perfectly rotationally symmetric*, etc.
- could *lose rank*, with very small probability
- suffer from *"coupon collector" problems*

In general, the *bias occurs differently in each direction*,

(so you cannot correct it with a single rescaling)

Q: Can we quickly correct the inversion bias, exactly or approximately?

Near-unbiasedness: an (ϵ, δ)-unbiased estimator

This motivates the following definition.

Definition

A random p.s.d. matrix $\tilde{C}$ is an (ϵ, δ)-unbiased estimator of C if there is an event $\mathcal{E}$ that holds with probability $1 - \delta$ such that

$$\mathbb{E}_{\mathcal{E}}[\tilde{C}] \approx_{1+\epsilon} C, \quad \text{and} \quad \tilde{C} \preceq O(1) \cdot C \quad \text{when conditioned on } \mathcal{E}.$$

Sub-gaussian sketches have small inversion bias

Consider a full rank $n \times d$ matrix A with $n \gg d$.

Proposition (Near-unbiasedness of sub-gaussian sketches)

Let S be an $m \times n$ random matrix such that $\sqrt{m}\,S$ has i.i.d. $O(1)$-sub-gaussian entries with mean zero and unit variance.

$$\textit{If} \qquad m \geq C\big(d + \sqrt{d}/\epsilon + \log(1/\delta)\big), \qquad \textit{then}$$

$(\frac{m}{m-d} A^\top S^\top S A)^{-1}$ is an (ϵ, δ)-unbiased estimator of $(A^\top A)^{-1}$.

So, there is an event $\mathcal{E}$ that holds with probability $1 - \mathrm{e}^{-cm}$, s.t.

$$\mathbb{E}_{\mathcal{E}}\left[(\tfrac{m}{m-d} A^\top S^\top S A)^{-1}\right] \approx_\epsilon (A^\top A)^{-1}, \quad \text{for} \quad \epsilon = O\left(\frac{\sqrt{d}}{m}\right).$$

[DLDM21]

Comparison with JL / subspace embeddings

Condition: Subspace embedding

Sketching matrix S with probability $1 - \delta$ satisfies

$$A^\top S^\top S A \approx_\eta A^\top A \qquad \text{for} \quad \eta = O(1).$$

Subspace embedding: w.h.p. $(A^\top S^\top S A)^{-1} \approx_\eta (A^\top A)^{-1}$

Near-unbiasedness: $\mathbb{E}_{\mathcal{S}}\left[(\frac{m}{m-d} A^\top S^\top S A)^{-1}\right] \approx_\epsilon (A^\top A)^{-1}$

For sub-gaussian sketches, we have:

$$\eta = \Theta\left(\sqrt{\frac{d}{m}}\right) \quad \text{and} \quad \epsilon = O\left(\frac{\sqrt{d}}{m}\right)$$

Subspace embedding is not enough to show near-unbiasedness!

Corollary for model averaging

Effectively, we showed that for sub-gaussian sketches:

$$\text{Bias}^2 \quad \ll \quad \text{Variance}$$

Corollary (Model averaging)

For $q = \tilde{O}(m)$ sub-gaussian sketches of size $m = O(d + \sqrt{d}/\epsilon)$,

$$\frac{1}{q}\sum_{i=1}^{q}\left(\tfrac{m}{m-d}A^\top S_i^\top S_i A\right)^{-1} \approx_\epsilon (A^\top A)^{-1}.$$

Applies to distributed averaging of linear functionals, e.g.:

$$\operatorname{tr} C\left(\tfrac{m}{m-d}A^\top S_i^\top S_i A\right)^{-1}.$$

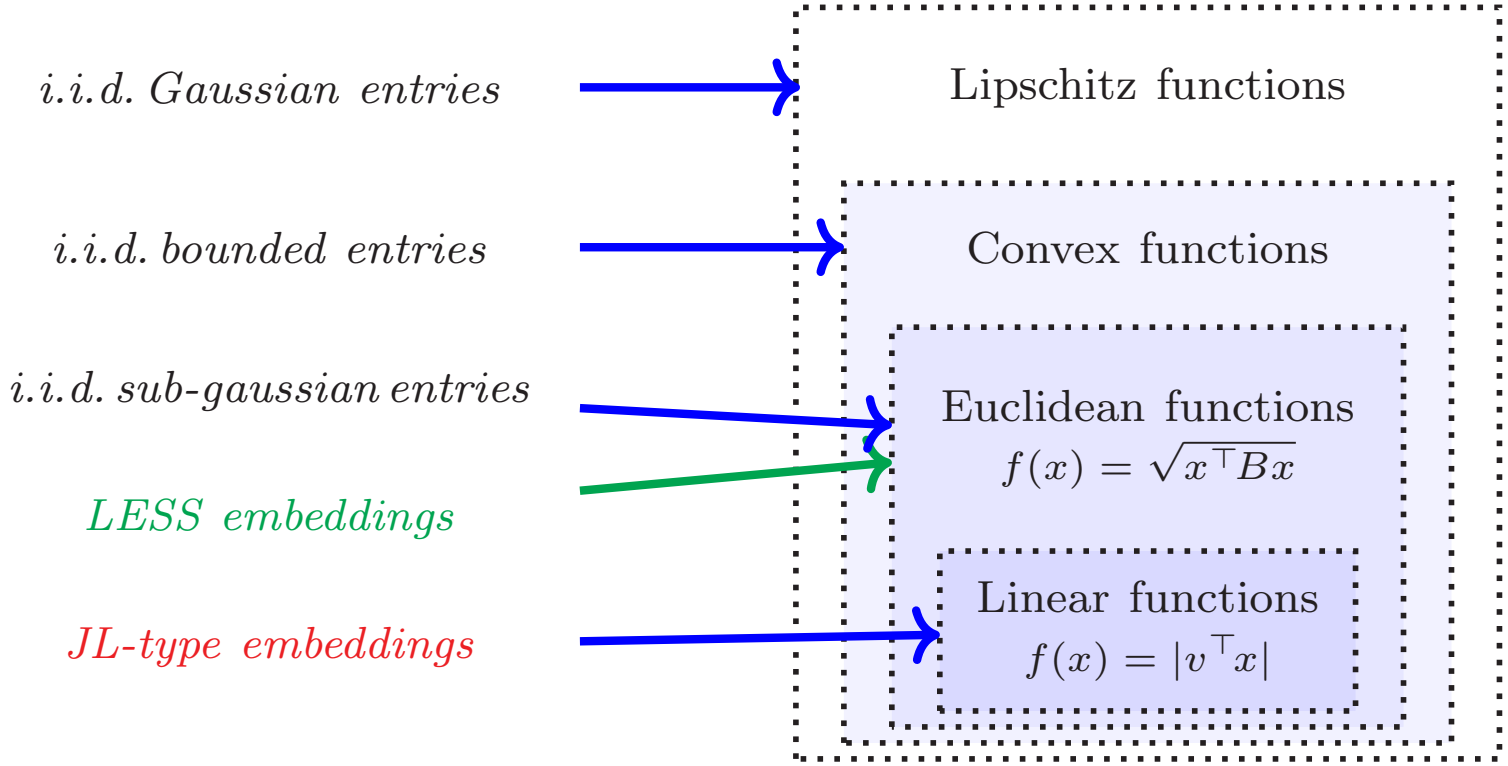
Landscape of Algorithmic Gaussianization

Sub-gaussian concentration of $x \in \mathbb{R}^d$ w.r.t. a set of functions $\mathcal{F}$

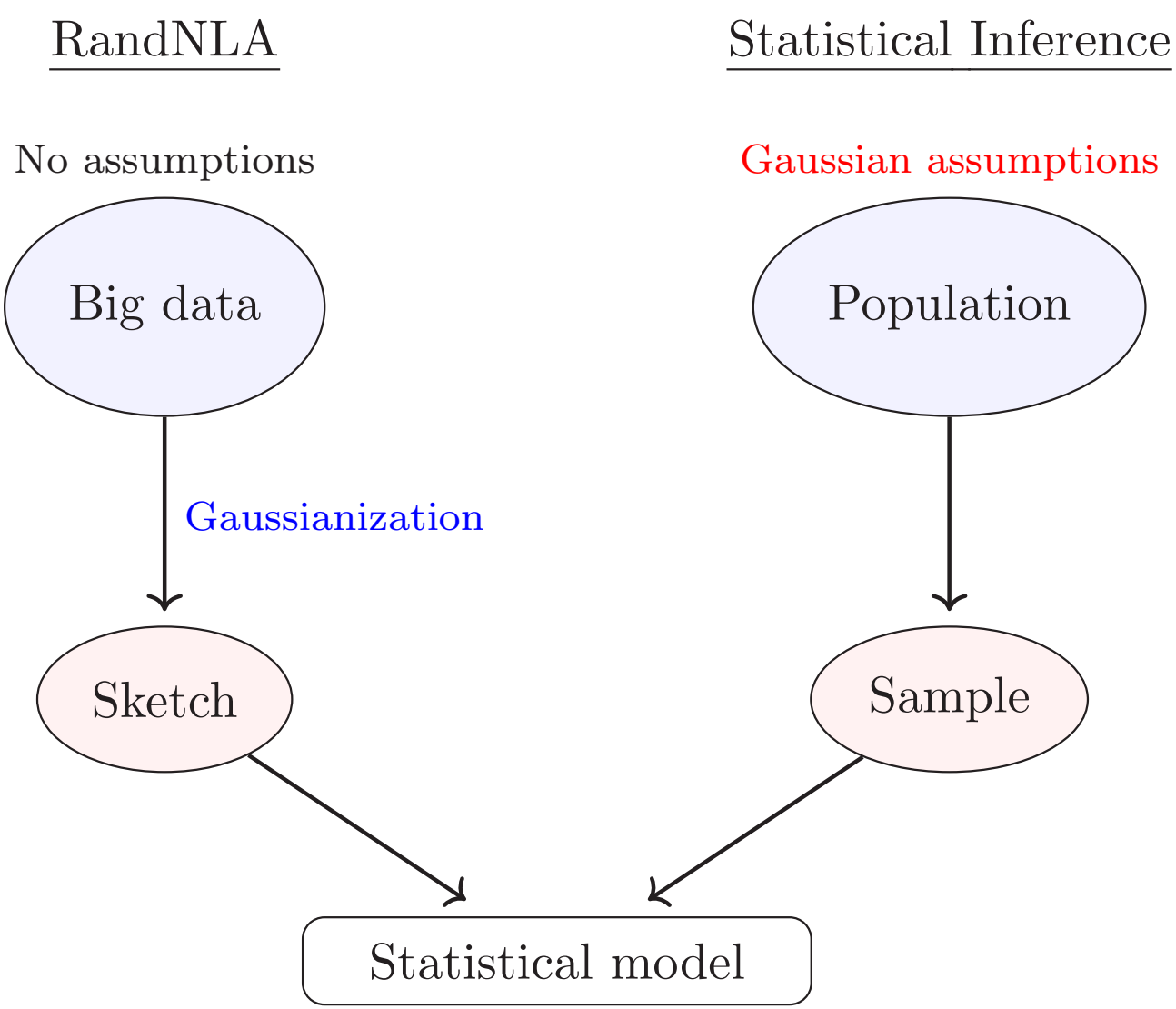
$$\forall f \in \mathcal{F}: \qquad X = f(x) - \mathbb{E}\, f(x) \quad \text{is} \quad \underbrace{O(\|f\|_{\text{Lip}})\text{-sub-gaussian}}_{\mathbb{E}\, \exp(cX^2/\|f\|_{\text{Lip}}) \leq 2}$$

Examples
$x \in \mathbb{R}^d$

Concentration
$\mathcal{F} \subseteq \{\mathbb{R}^d \to \mathbb{R}\}$

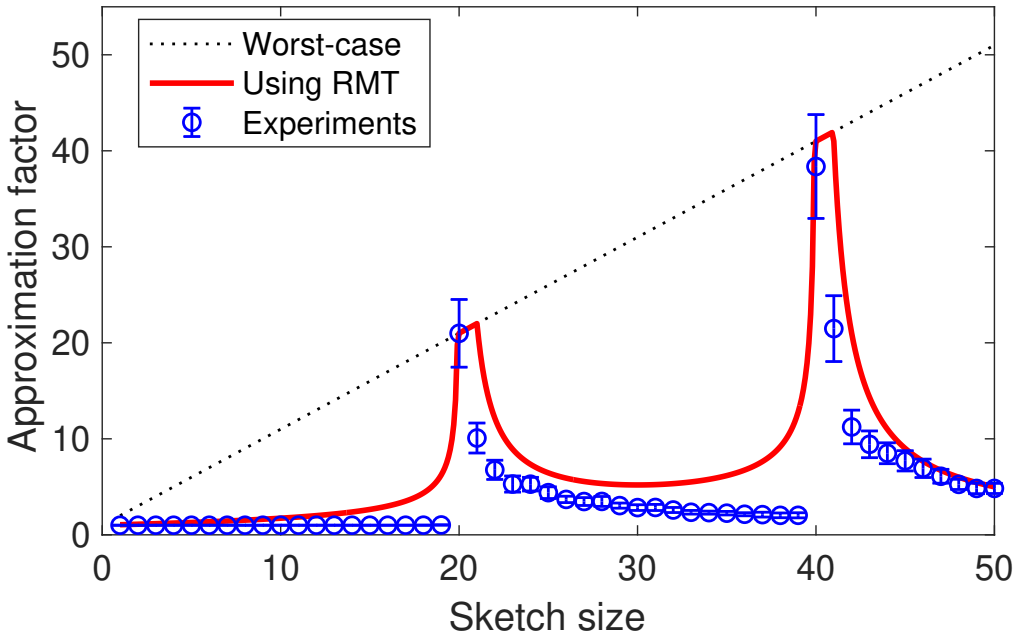


Gaussianization in RandNLA vs Statistical Inference



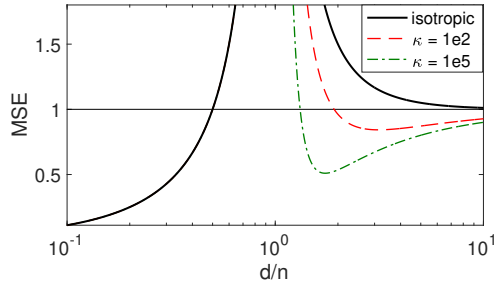
Multiple-descent in low-rank approximation

Theory: Characterizing the approximation factor using RMT [DKM20]



Connection: double descent in over-parameterized ML models [DLM20b]

"Classical" ML:	$\textit{parameters} \ll \textit{data}$
"Modern" ML:	$\textit{parameters} \gg \textit{data}$
Phase transition:	$\textit{parameters} \sim \textit{data}$



Developing standard libraries for RandNLA

RandBLAS

- Library that concerns basic sketching for dense data matrices.
- Reference implementation in C++.
- Hope: it grows to become a community standard for RandNLA, in the sense that its API would see wider adoption than any single implementation.

RandLAPACK

- Library that concerns algorithms for solving traditional linear algebra problems and advanced sketching functionality.
- To be written in C++, build on BLAS++/LAPACK++ portability layer
- Main drivers:
  - Least squares and optimization.
  - Low-rank approximation.
  - Full-rank decompositions.

"The RandLAPACK book"

Mathematics > Numerical Analysis

[Submitted on 22 Feb 2023]

Randomized Numerical Linear Algebra : A Perspective on the Field With an Eye to Software

Riley Murray, James Demmel, Michael W. Mahoney, N. Benjamin Erichson, Maksim Melnichenko, Osman Asif Malik, Laura Grigori, Piotr Luszczek, Michał Dereziński, Miles E. Lopes, Tianyu Liang, Hengrui Luo, Jack Dongarra

Randomized numerical linear algebra – RandNLA, for short – concerns the use of randomization as a resource to develop improved algorithms for large-scale linear algebra computations.

The origins of contemporary RandNLA lay in theoretical computer science, where it blossomed from a simple idea: randomization provides an avenue for computing approximate solutions to linear algebra problems more efficiently than deterministic algorithms. This idea proved fruitful in the development of scalable algorithms for machine learning and statistical data analysis applications. However, RandNLA's true potential only came into focus upon integration with the fields of numerical analysis and "classical" numerical linear algebra. Through the efforts of many individuals, randomized algorithms have been developed that provide full control over the accuracy of their solutions and that can be every bit as reliable as algorithms that might be found in libraries such as LAPACK. Recent years have even seen the incorporation of certain RandNLA methods into MATLAB, the NAG Library, NVIDIA's cuSOLVER, and SciPy.

For all its success, we believe that RandNLA has yet to realize its full potential. In particular, we believe the scientific community stands to benefit significantly from suitably defined "RandBLAS" and "RandLAPACK" libraries, to serve as standards conceptually analogous to BLAS and LAPACK. This 200-page monograph represents a step toward defining such standards. In it, we cover topics spanning basic sketching, least squares and optimization, low-rank approximation, full matrix decompositions, leverage score sampling, and sketching data with tensor product structures (among others). Much of the provided pseudo-code has been tested via publicly available Matlab and Python implementations.

Comments: v1: this is the first arXiv release of LAPACK Working Note 299

Subjects: **Numerical Analysis (math.NA)**; Mathematical Software (cs.MS); Optimization and Control (math.OC)

Cite as: arXiv:2302.11474 **[math.NA]**

(or arXiv:2302.11474v1 **[math.NA]** for this version)

https://doi.org/10.48550/arXiv.2302.11474

Overview

Some thoughts on the past

Some thoughts on the present

Some thoughts on the future

Motivation

- **NLA*, the hidden engine of modern computing, faces fundamental transition**
  - Built for the earlier **CPU** era
  - Must be reimagined for the **GPU** era: heterogeneous accelerators, and low-precision arithmetic
- **Arithmetic precision is now an algorithmic design parameter**
  - **Mixed precision** and **quantization** (FP8, FP4, beyond) drive performance
  - But lack principled theory and reliability guarantees
- **Foundational theory lags behind computing practice**
  - Recent advances on LLMs are produced by industry
  - Broader numerical methods operate under outdated assumptions
  - This is an increasingly **critical bottleneck**

Thesis: The AI era requires a new numerical linear algebra stack, where randomness and precision are first-class algorithmic objects.

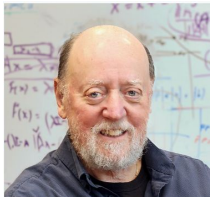
*"NLA" = "Numerical linear algebra"

Historical Context

- **Post World War II**
  - Development of the digital* computer
  - **NLA** emerged at the intersection of EE & Mathematics -> **led to CS**
- **Big Problem**
  - Potentially **catastrophic, "roundoff" errors**** → All matrix computations are "wrong"
- **Intellectual Foundations**
  - von Neumann, Turing, Forsythe, etc.
  - Wilkinson (Turing Award 1970): **roundoff** error analysis
  - Kahan (Turing Award 1989) IEEE 754 **floating point** specification
  - **Dongarra** (Turing Award 2021) HPC **libraries**



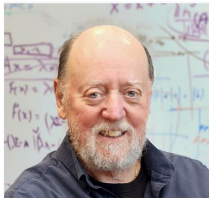




Before the electronic age, "computer" was a job title -- human computers performed calculations by hand.
**Errors are catastrophic when problems are ill-conditioned, etc.; but that is the point!*

Historical Context

- **Post World War II**
  - Development of the digital* computer
  - **NLA** emerged at the intersection of EE & Mathematics -> **led to CS**
- **Big Problem**
  - Potentially **catastrophic, "roundoff" errors**** → All matrix computations are "wrong"
- **Numerical libraries: gold standard for high-performance matrix computations**
  - LINPACK and LAPACK
  - **Dongarra**, **Demmel**, Moler, Stewart, Bunch, etc.)
  - ScaLAPACK, MAGMA, etc.






*Before the electronic age, "computer" was a job title -- human computers performed calculations by hand.
**Errors are catastrophic when problems are ill-conditioned, etc.; but that is the point!

Our Vision

- **Foundations:**
  - Develop a *principled mathematical foundation* for **reliable NLA/RandNLA in the AI/ML era***
- **Implementation:**
  - Design *next-gen NLA libraries* by leveraging **randomization, random matrix theory, and mixed precision** with bindings for PyTorch and TensorFlow
- **Applications:**
  - **Validate AI/ML libraries** on *scientific ML and generative AI models.*

Software goal: an AI-era NLA engine that automatically selects sketching, precision, decomposition, and hardware kernels for a target accuracy, latency, memory, and energy budget.

**We will have succeeded if you (and future generations) never have to think about this stuff …*

DeepSeek-R1 as a motivating example

- **Deepseek-R1** achieves sophisticated reasoning via:
  - **Clever algorithmic design,** not just massive increases in compute & data
  - Custom methods, mixed-precision training, low-rank approximations
- But:
  - These powerful approaches remain ***ad hoc, incremental and poorly understood*.**
- Industry and researchers face *severe resource constraints* (export controls, university-scale budgets, industry demand for AGI):
  - Must **innovate at a fundamental level**
  - More data and more compute is not enough
  - Must **look "under the hood"** of algorithms and hardware

Our proposal addresses this gap!

Why now? Three converging forces…

1. Hardware shift

- **Mixed precision** is no longer optional; it is the default design principle
- A full spectrum of arithmetic:
  - from **high precision** down to **ultra-low precision** formats (MXFP4 and NVFP4*)
  - rapidly maturing 2-bit and even 1-bit compute regimes
- Hardware is **no longer homogeneous:**
  - modern systems exploit a hierarchy of precisions
  - performance depends critically on algorithms

*Implemented in Nvidia's NVFP4 standard: https://developer.nvidia.com/blog/introducing-nvfp4-for-efficient-and-accurate-low-precision-inference/

Why now? Three converging forces…

2. Demand shift

- **AI/ML workloads now dominate** computation
  - Previously CSE workloads dominated
- AI/ML systems are driven by **large-scale NLA**
  - NLA is now the true bottleneck
  - NLA is invisible to end users
  - NLA is fundamental for performance.
- Strong shift from classical CSE to **AI/ML-centric workloads**
  - Efficiency, throughput, and scaling laws are dictated by NLA

**"CSE" = "computational science & engineering" = solves complex problems in the physical, biological, social sciences, and in engineering!*

Why now? Three converging forces…

3. Foundational shift

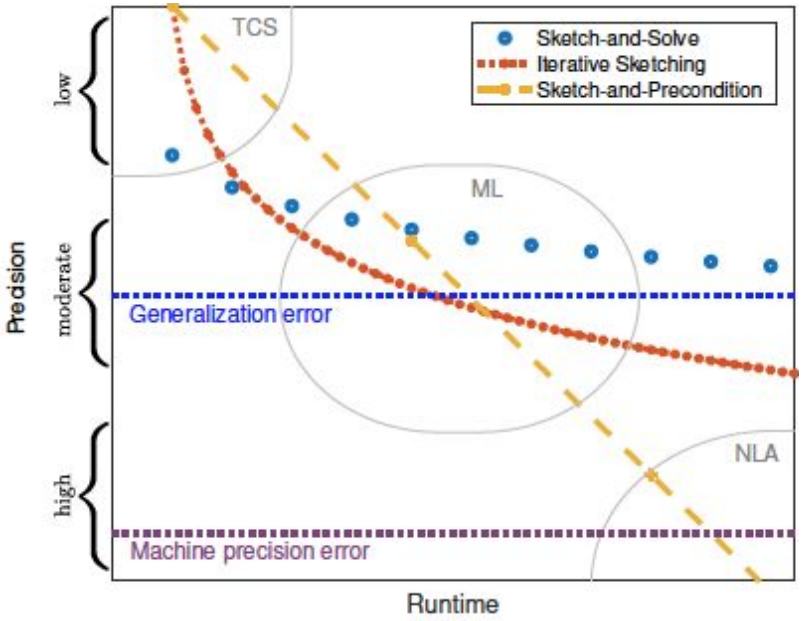
- **Theory is catching up to practice** AND practice is catching up to theory.

- Recent theoretical developments → much smaller theory–practice gap
  - Traditional RandNLA (based on Johnson-Lindenstrauss) is “too coarse”
  - **Non-asymptotic Random Matrix Theory (RMT)**: finer insight for “moderate precision”

- **Growing theoretical readiness** → the time is right
    - Traditional RandNLA:
      - Led to hiqh-quality efficient CSE code and RandBLAS/RandLAPACK
    - Modern RandNLA:
      - Randomized rounding, Randomized Hadamard Transforms, etc.
      - Are increasingly implemented in hardware and at low precision

RandNLA Theory: Three Paradigms

- Sketch-and-solve (classical RandNLA)
  - Construct sketch and solve subproblem
  - Low precision ($\varepsilon \approx 10^{-1}$)
  - Appropriate for **theory**
- Sketch-and-precondition (classical RandNLA)
  - Use sketch to construct preconditioner for traditional iterative algorithm
  - High precision: ($\varepsilon \approx 10^{-16}$)
  - Appropriate for **CSE**
- Iterative sketching (modern RandNLA)
  - Iteratively sketch
  - Medium precision: (e.g., $\varepsilon \approx 10^{-4}$)
  - Appropriate for **AI/ML and quantization**



RandNLA Impact on Industry/Hardware

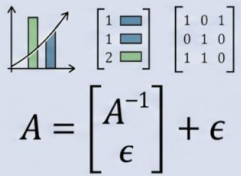
- Random Projections (Sketching)
  - Core technique behind Google's **TurboQuant*** LLM quantization
- Randomized Matrix Multiplication
  - Fast approximate products via sampling / sketching
  - Reduces communication and memory bottlenecks
- Randomized SVD
  - Efficient low-rank approximation at scale. Implemented in **scikit-learn**
  - Core tool for PCA, compression, and many LLM compression methods
- Randomized Hadamard Transforms
  - Fast structured dimension reduction
  - Implemented in **Nvidia's NVFP4 quantization**
- Randomized Least Squares Solvers
  - Sketch-and-solve, iterative sketching
- Randomized Rounding
  - Used in many quantized LLMs
  - Implemented in **Nvidia's LLM inference libraries**

*See https://research.google/blog/turboquant-redefining-ai-efficiency-with-extreme-compression/

Research structure

FOUNDATIONS

Foundational NLA & AI-Ready Math



$$A = \begin{bmatrix} A^{-1} \\ \epsilon \end{bmatrix} + \epsilon$$

- Mixed Precision & Quantization Theory.
- Rounding, Perturbation & Numerical Stability
- Next-Generation Sketching Methods for NLA.
- Randomized Matrix Theory for AI/ML-Era NLA

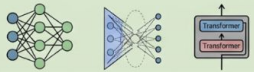
IMPLEMENTATIONS

Scalable Software & Automated Workflows

- Quantization & Mixed-Precision Simulation.
- Rapid Prototyping for Randomized NLA.
- Application-Driven Validation & Benchmarking.
- Maturation into BALLISTIC / RandLAPACK.

AI APPLICATIONS

Empowering AI Models



- Efficient Generative AI.
- Efficient Transformers.

SCIENCE APPLICATIONS

Empowering Computational Science

- Scalable Foundation Models.
- Scalable Solvers for Science.

BROADER IMPACTS & PARTNERSHIPS

Key Industrial Partners

IBM Google Microsoft Mathworks JuliaHub

Key National Labs

LBNL Sandia ORNL

Key Educational Impacts

- Multi-Disciplinary Training in Foundations of NLA & AI.
- Online Education Videos.
- Public Code Repositories.

Julia (co-PI Edelman) enables rapid prototyping; **BALLISTIC** (PI Mahoney) delivers mature, optimized implementations; **Python** bindings deliver application impact.

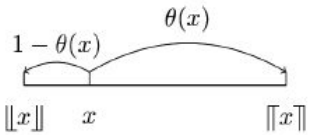
Deep Dive on 3 Pillars

- **Deep Dive A: Foundations**
  - Stochastic rounding: a model of computation
- **Deep Dive B: Implementation**
  - Rapid experimentation on attention
- **Deep Dive C: Application**
  - LLMs and Generative AI
  - Operator inference in SciML

Deep Dive A: Stochastic rounding, a model of computation

What is Stochastic Rounding (SR)?

- $\theta(x) = \dfrac{x - \lfloor x \rfloor}{\lceil x \rceil - \lfloor x \rfloor}$
- SR: round *up* with probability $\theta(x)$ or round *down* with probability $1 - \theta(x)$



- SR is unbiased, i.e., $\mathbb{E}\left[\mathrm{SR}(x)\right] = x$

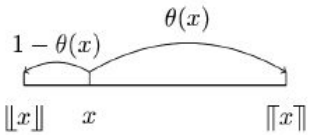
Why view SR as a model of computation?

- Analogous to **smoothed analysis** → injects small random perturbations
- Captures realistic behavior of **low-precision hardware**
- Bridges:
  - Worst-case analysis: too pessimistic
  - Average-case analysis: too idealized
  - **SR-based analysis: practical + implementable**

Deep Dive A: Stochastic rounding, a model of computation

What is Stochastic Rounding (SR)?

- $\theta(x) = \dfrac{x - \lfloor x \rfloor}{\lceil x \rceil - \lfloor x \rfloor}$
- SR: round *up* with probability $\theta(x)$ or round *down* with probability $1 - \theta(x)$



- SR is unbiased, i.e., $\mathbb{E}\left[\mathrm{SR}(x)\right] = x$

Advantages:

- Prevents error accumulation in low precision
- Acts as **implicit regularization** (improves conditioning)
- Enables **stable mixed-precision algorithms**
- Naturally aligns with **modern accelerators (GPUs, IPUs, neuromorphic hardware)**

Key Takeaway:

Stochastic rounding is new computational paradigm for **analyzing and designing NLA algorithms in the AI/ML era**.

Deep Dive B: Rapid Experimentation on Attention

- **Question:** are we actually able to rapidly prototype in Julia?
- **Yes!** Demo: speeding up the attention mechanism (bottleneck) in modern LLMs.
- RandLinearAlgebra.jl on GPU or CPU can do this in **3 Lines of Code**

```
15  function compressed_systems(Wq, Wk, Wv; compression_dim)
16      specs = Gaussian(cardinality=Right(), compression_dim=compression_dim, type=eltype(Wq))
17      compressors = [complete_compressor(specs, transpose(W)) for W in [Wq, Wk, Wv]]
18      return [W'*comp for (comp, W) in zip(compressors, [Wq, Wk, Wv])]
19  end
```

- **The result:** RandNLA gives 10x speed up on CPU, and 17x speed up on A100 GPU

```
32  println("=== CPU ===")
33  t1 = @elapsed attention(X_cpu, Weights_cpu...); "Standard:    $(round(t1, digits=4))s"  "Standard:    6.7911s"
34  t2 = @elapsed attention(X_cpu, Weights_cpu_c...); "Compressed:  $(round(t2, digits=4))s"  "Compressed:  0.6569s"
35
36  println("\n=== GPU (Metal) ===")
37  t3 = @elapsed attention(X_gpu, Weights_gpu...); "Standard:    $(round(t3, digits=4))s"  "Standard:    3.3345s"
38  t4 = @elapsed attention(X_gpu, Weights_gpu_c...); "Compressed:  $(round(t4, digits=4))s"  "Compressed:  0.1923s"
```

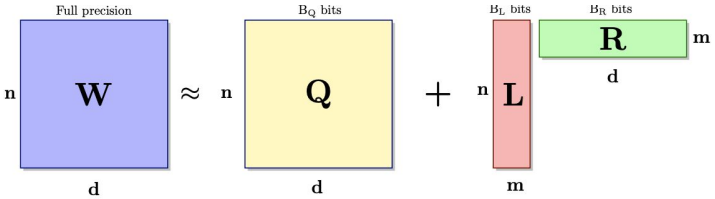
Deep Dive B: Rapid Experimentation on Attention

- **Message:**
  - We need only **38 lines** of code for the entire experiment on attention

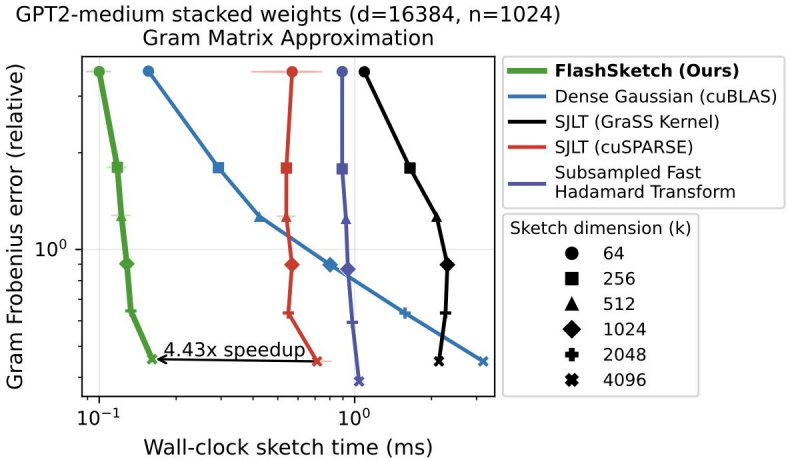
- **What do we still need to do?**
  - Produce end-to-end application libraries to validate the **speed-ups and contextually-meaningful results** of our NLA
  - Equip RandLinearAlgebra.jl with new NLA techniques, (simulated) FP/quantization choices, and newer hardware backends (beyond GPUs)
  - Automate benchmarking and parameter tuning determine the best tools for priority implementations in BALLISTIC (C++) and for **Python bindings to realize application advancement**

Deep Dive C: LLMs and Generative AI

- **LLM inference:**
  - Randomized decompositions + low-bit quantization enable structure-aware compression
  - State-of-the-art accuracy–efficiency tradeoffs
  - Heuristic compression → **principled, analyzable algorithms**



- **LLM interpretability:**
  - Interpretability as structured NLA with specialized kernels
  - **>4× speedup** over NVIDIA inference libraries
  - Establish interpretability as a **systems-level optimization opportunity**



- **LLM Training: Spectral Methods**
  - Newton–Schulz iterations → fast spectral preconditioning
  - Efficient matrix inverse/root approximations at scale
  - **Accelerate training** with stable, low-overhead spectral updates

Deep Dive C: Operator Inference and Scientific ML

- **Context:** control, data assimilation, & PINNs require inner loops that repeatedly solve complex DEs
- **Operator Inference:**
  - A low-fidelity surrogate model can be used in place of full solution
  - The state is first embedded to lower dimensions with an SVD
- **Problem:** SVD cannot scale to realistic discretizations
- **RandNLA Solution:** RandomizedSVD scales to realistic discretizations

- **Example:** Solving simple model* (2048 dimensions and 4000 time steps) takes **79.87 seconds** with an explicit method, which is slow for ~50 to 100 iterations
- **Operator Inference**

	SVD	G-RSVD	SS-RSVD
Estimation Time (s)	20.03 s	0.136 s	0.16 s
Forward Solve Time (s)	13.78 s	13.56 s	13.62 s
Rel. Trajectory Error	< 0.1%	< 0.1%	< 0.1%

- **Message:** Estimating model is **100x faster** with RandomizedSVD compared to SVD; operator inference produces **4-5x** speed up compared to direct solve.

Patel et al. "Scientific Applications Leveraging Randomized Linear Algebra" arXiv:2506.16457

Overview

Some thoughts on the past

Some thoughts on the present

Some thoughts on the future

Towards the future …

- "Spectral Estimation with Free Decompression," Ameli et al. arXiv:2506.11994, NeurIPS 2025.
- "Random Matrix Theory for Deep Learning: Beyond Eigenvalues of Linear Models," Liao et al. arXiv:2506.13139, IEEE SPM (2026).
- "PRISM: Distribution-free Adaptive Computation of Matrix Functions for Accelerating Neural Network Training," Yang et al. arXiv:2601.22137, ICML 2026.
- "Learning to Discover Iterative Spectral Algorithms," Liu et al. arXiv:2602.09530.

Towards the future …

- "Spectral Estimation with Free Decompression," Ameli et al. arXiv:2506.11994, NeurIPS 2025.
- "Random Matrix Theory for Deep Learning: Beyond Eigenvalues of Linear Models," Liao et al. arXiv:2506.13139, IEEE SPM (2026).
- "PRISM: Distribution-free Adaptive Computation of Matrix Functions for Accelerating Neural Network Training," Yang et al. arXiv:2601.22137, ICML 2026.
- "Learning to Discover Iterative Spectral Algorithms," Liu et al. arXiv:2602.09530.

Tiers of Matrix Difficulty

Explicit: the whole matrix fits in memory

Implicit: can make use of matrix-vector products (e.g. CG, SLQ)

Out-of-core: parts of the matrix can be loaded into memory a piece at a time

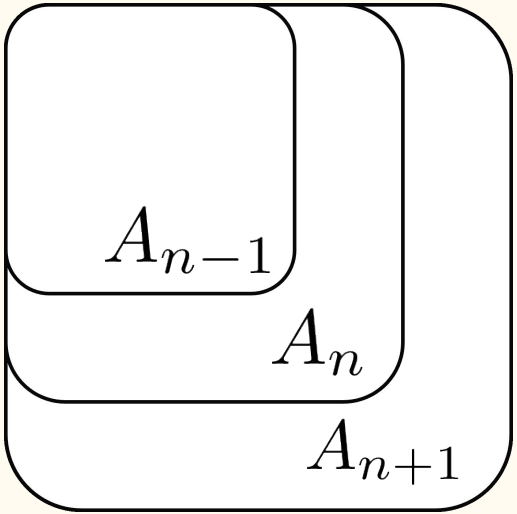
Impalpable: most matrix entries are inaccessible, matrix-vector products are unavailable (e.g. distributed or enormous datasets)

	Access in Memory		
Type	Matrix	Matrix–Vector Product	Any Subblock
Explicit	✓	✓	✓
Implicit	✗	✓	✗
Out-of-core	✗	∼	✓
Impalpable	✗	✗	✗

Extrapolating Matrices

Suppose our matrix of interest is embedded in an infinite sequence of nested matrices

$$A_1, A_2, A_3, \ldots \qquad A_n \in \mathbb{R}^{n \times n}$$



so that $(A_n)_{ij} = (A_{n+1})_{ij}$

Objective: Find eigenspectrum of A_n using eigenspectrum of A_{n_s} for $n_s \ll n$

Free Probability

How do we ensure the eigenvalues of submatrices represent the whole matrix?

An important topic in random matrix theory involving random matrices with uniformly random eigenvectors, so that probability distributions of matrix dependents (including submatrices) *depend only on the eigenspectra*.

Theorem (Nica, 1993): Any sequence of matrices can be turned into an (asymptotically) free sequence of random matrices by applying random permutations σ to the rows and columns:

$$\tilde{A}_{ij} = A_{\sigma(i)\sigma(j)}$$

Free Decompression

Let $m(t, \cdot)$ be the Stieltjes transform of the enlargement of A by a factor of e^t
Under the large matrix limit, $m(t, \cdot)$ satisfies the *partial differential equation:*

$$\frac{\partial m}{\partial t} = -m + \frac{1}{m}\frac{\partial m}{\partial z}$$

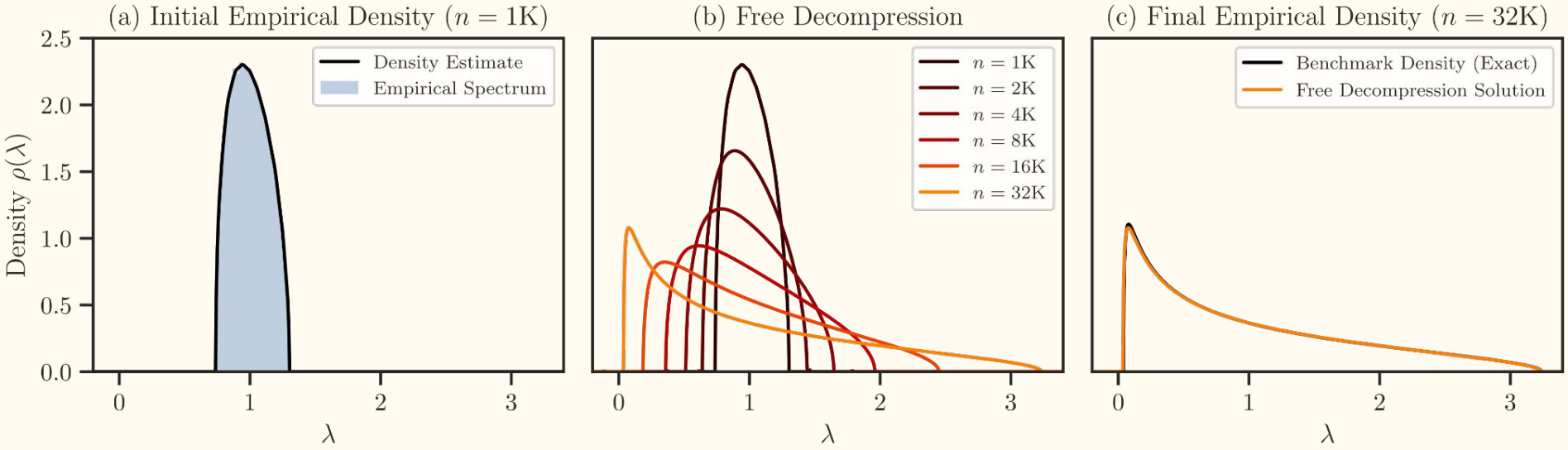
Proof: Random matrix theory arguments involving the R-transform and the celebrated theorem of (Nica & Speicher, 1996).

To our knowledge, this operation has always been considered in reverse (*free compression*), finding eigenspectra of submatrices, given the eigenspectrum of the full matrix. We are the first to attempt ***free decompression.***

Free decompression of a random submatrix $\mathbf{A}_n$ to a larger matrix $\mathbf{A}$ requires:

1. **estimation** of its Stieltjes transform $m_{\mathbf{A}_n}$;
2. **evolution** of $m_{\mathbf{A}_n}$ in n using PDE;
3. **evaluation** of the spectral distribution of $\mathbf{A}$.

Wishart Matrices (Marchenko-Pastur Law)

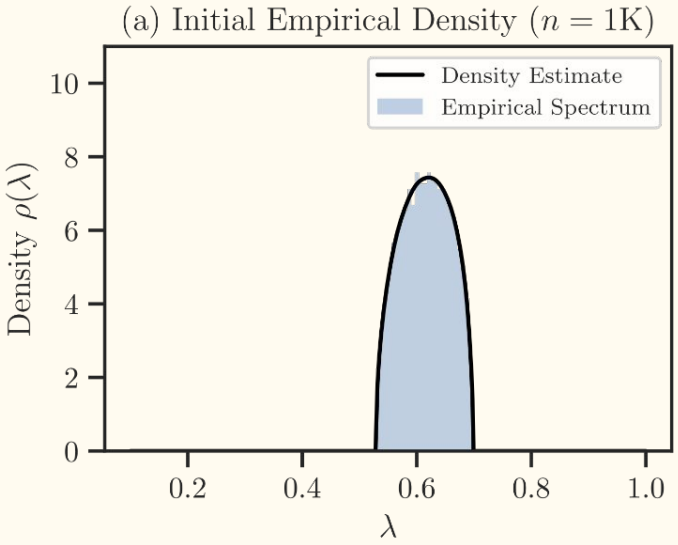


Histogram of eigenvalues of small matrix & density estimate

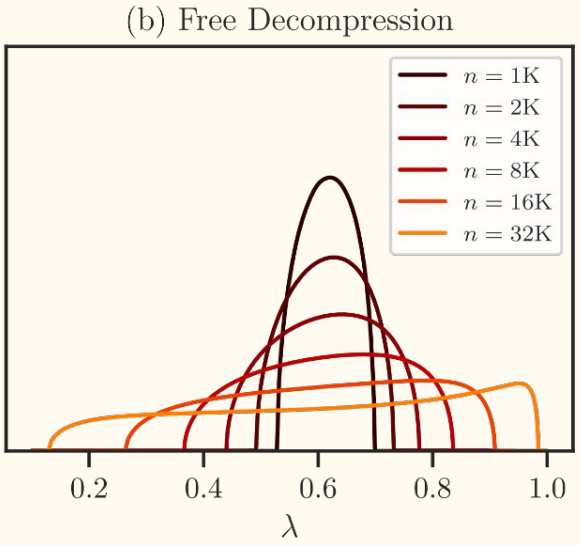
Densities under free decompression

Expected density & solution from free decompression

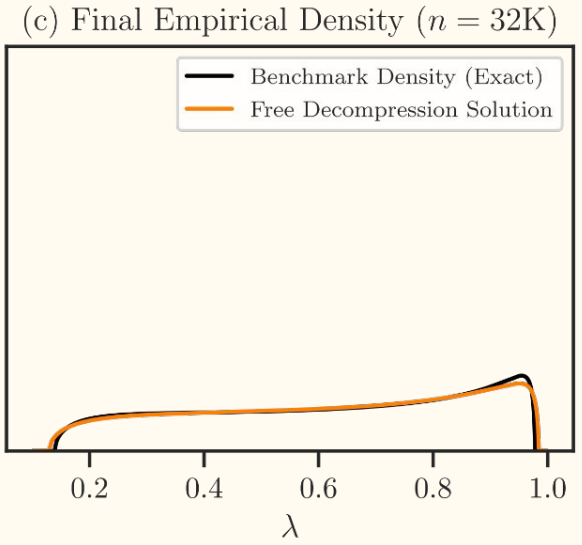
Generalized Eigenvalue Problems (Wachter Law)



Histogram of eigenvalues of small matrix & density estimate



Densities under free decompression

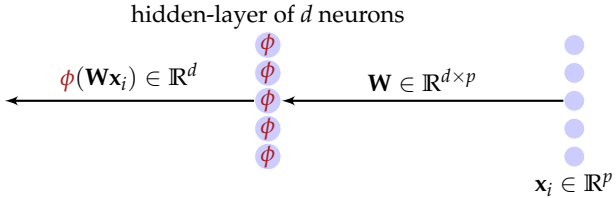


Expected density & solution from free decompression

Towards the future …

- "Spectral Estimation with Free Decompression," Ameli et al. arXiv:2506.11994, NeurIPS 2025.
- "Random Matrix Theory for Deep Learning: Beyond Eigenvalues of Linear Models," Liao et al. arXiv:2506.13139, IEEE SPM (2026).
- "PRISM: Distribution-free Adaptive Computation of Matrix Functions for Accelerating Neural Network Training," Yang et al. arXiv:2601.22137, ICML 2026.
- "Learning to Discover Iterative Spectral Algorithms," Liu et al. arXiv:2602.09530.

A deep neural network model



- **linear transformation** with first-layer weight matrix $\mathbf{W} \in \mathbb{R}^{d\times p}$
- **nonlinear transformation**: activation function $\phi\colon \mathbb{R} \to \mathbb{R}$ acting entry-wise on $\mathbf{W}\mathbf{x}_i$
- **data representation** at the output of first-layer $\boxed{\mathbf{x}_i \mapsto \phi(\mathbf{W}\mathbf{x}_i)}$
- do the same thing in a layer-by-layer fashion:

$$\frac{1}{\sqrt{d_L}}\mathbf{w}^{\mathsf{T}}\phi_L\left(\frac{1}{\sqrt{d_{L-1}}}\mathbf{W}_L\phi_{L-1}\left(\cdots\frac{1}{\sqrt{d_2}}\phi_2\left(\frac{1}{\sqrt{d_1}}\mathbf{W}_2\phi_1(\mathbf{W}_1\mathbf{x}_i)\right)\right)\right), \tag{1}$$

for a large number n of input data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$

Technical challenges and key ideas

Analyze and Optimize Large-scale ML model $\mathcal{M}_\phi(\mathbf{X};\mathbf{\Theta})$

Objective: Evaluation of $\mathcal{M}_\phi(\mathbf{X};\mathbf{\Theta})$ via Performance Metric $f(\cdot)$

Technical Challenge 1 High-dimensionality in $\mathbf{X}, \mathbf{\Theta}$	**Key Idea 1** Concentration of $f\left(\mathcal{M}_\phi(\mathbf{X};\mathbf{\Theta})\right) \simeq \mathbb{E}[f\left(\mathcal{M}_\phi(\mathbf{X};\mathbf{\Theta})\right)]$
Technical Challenge 2 Analysis of Eigen-functional	**Key Idea 2** Deterministic Equivalent for Resolvent
Technical Challenge 3 Non-linearity in ML model	**Key Idea 3** High-dimensional linearization of $\mathcal{M}_\phi(\mathbf{X};\mathbf{\Theta})$

High-dimensional Equivalent

Definition (High-dimensional Equivalent)

Let $\mathcal{M}_\phi(\mathbf{X}) \in \mathbb{R}^{p\times n}$ be a (nonlinear) random matrix model that depends on a random matrix $\mathbf{X} \in \mathbb{R}^{p\times n}$ and function $\phi\colon \mathbb{R} \to \mathbb{R}$ (typically applied entrywise). Let $f\left(\mathcal{M}_\phi(\mathbf{X})\right)$ be a scalar observation of $\mathcal{M}_\phi(\mathbf{X})$ for some $f\colon \mathbb{R}^{p\times n} \to \mathbb{R}$. We say that $\tilde{\mathcal{M}}_\phi(\mathbf{X})$ (random or deterministic) is a High-dimensional Equivalent of $\mathcal{M}_\phi(\mathbf{X})$ with respect to $f(\cdot)$ if

$$f(\mathcal{M}_\phi(\mathbf{X})) - f(\tilde{\mathcal{M}}_\phi(\mathbf{X})) \to 0, \tag{2}$$

in probability or almost surely as $n, p \to \infty$ with $p/n \to c \in (0, \infty)$. We denote this relation as

$$\mathcal{M}_\phi(\mathbf{X}) \overset{f}{\leftrightarrow} \tilde{\mathcal{M}}_\phi(\mathbf{X}) \text{ or simply } \mathcal{M}_\phi(\mathbf{X}) \leftrightarrow \tilde{\mathcal{M}}_\phi(\mathbf{X}), \tag{3}$$

when f is clear from context.

- without (entrywise) nonlinearities, $f(\mathbf{X})$ concentrates around expectation $f(\mathbf{X}) \simeq \mathbb{E}[f(\mathbf{X})]$, and can be assessed through **Deterministic Equivalent** $f(\bar{\mathbf{X}})$;
- for scalar eigenspectral functionals, Deterministic Equivalent for Resolvent framework provides a unified approach to eigenspectral functionals of random matrices;
- for nonlinear models in two different scaling regimes (LLN versus CLT), $\phi(\mathbf{X})$ can be linearized to yield a **Linear Equivalent**.

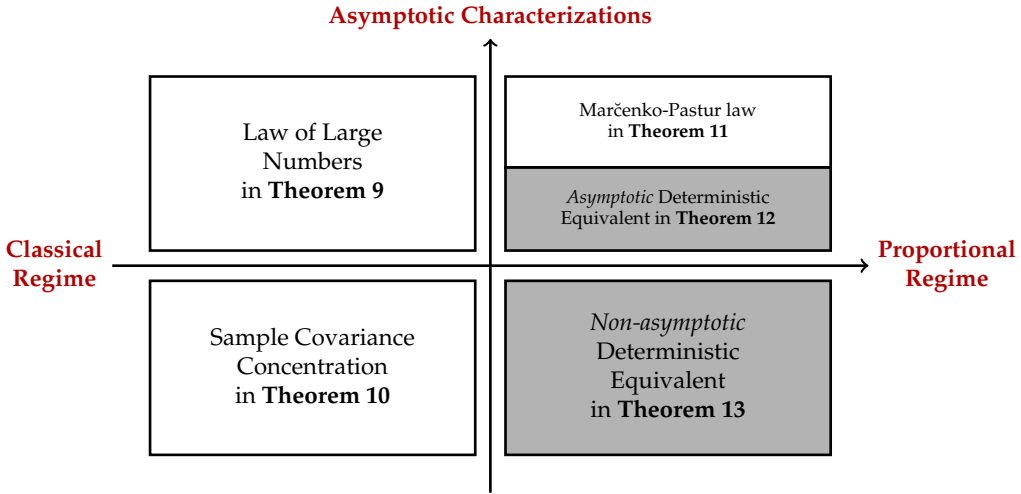


Figure: Taxonomy of four different ways to characterize the sample covariance matrix $\hat{\mathbf{C}} = \frac{1}{n}\mathbf{X}\mathbf{X}^\mathsf{T}$.

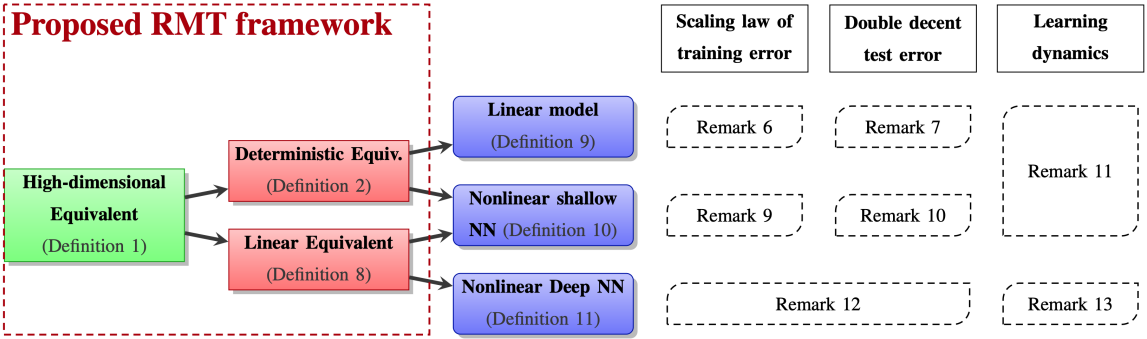


Figure: Overview of [LM25], summarizing major concepts and results and where to find them.

Towards the future …

- "Spectral Estimation with Free Decompression," Ameli et al. arXiv:2506.11994, NeurIPS 2025.
- "Random Matrix Theory for Deep Learning: Beyond Eigenvalues of Linear Models," Liao et al. arXiv:2506.13139, IEEE SPM (2026).
- "PRISM: Distribution-free Adaptive Computation of Matrix Functions for Accelerating Neural Network Training," Yang et al. arXiv:2601.22137, ICML 2026.
- "Learning to Discover Iterative Spectral Algorithms," Liu et al. arXiv:2602.09530.

Matrix functions are hidden primitives in deep learning

Many training algorithms repeatedly apply a scalar function f to a matrix spectrum:

$$A = U\Lambda U^\top \Rightarrow f(A) = Uf(\Lambda)U^\top, \qquad A = U\Sigma V^\top \Rightarrow f(A) = Uf(\Sigma)V^\top.$$

Algorithms	Matrix functions	Typical formula
Shampoo-type preconditioning (Vineet Gupta et al., 2018)	inverse matrix roots	$W_{t+1} = W_t - \eta\, L_t^{-\alpha} G_t R_t^{-\alpha}$
Natural-gradient / K-FAC (James Martens et al., 2015)	$(F + \lambda I)^{-1}$	$\Delta\theta = -(F + \lambda I)^{-1} g$
Muon / orthogonalization (Keller Jordan et al., 2024)	$\mathrm{polar}(M)$	$Q = M(M^\top M)^{-1/2} = UV^\top$
Second-order layers (Covariance pooling)	$C^{1/2}$, $\log(C)$	$z = \mathrm{vec}(C^{1/2})$ or $\mathrm{vec}(\log C)$
Whitening Decorrelated BatchNorm	$\Sigma^{-1/2}$	$\hat{x} = \Sigma^{-1/2}(x - \mu)$

Spectral gradient descent and Muon

Muon targets **matrix-shaped** parameters $W \in \mathbb{R}^{m \times n}$. Let $G_t = \nabla_W \mathcal{L}(W_t)$ and momentum

$$M_t = \beta M_{t-1} + (1-\beta) G_t.$$

Instead of stepping with M_t, Muon computes a **direction-only** update via the polar factor:

$$Q_t = \mathrm{polar}(M_t) = M_t (M_t^\top M_t)^{-1/2}.$$

If $M_t = U \Sigma V^\top$, then $Q_t = U V^\top$ (all singular values are 1), and $W_{t+1} = W_t - \eta\, Q_t$.

- Muon produces **direction-only** updates for weight matrices in NNs.
- Muon uses Newton–Schulz-like polynomial iterations (GPU-friendly) instead of SVD.
- Existing accelerations, e.g., PolarExpress (Amsel et al., 2025); CANS (Grishina et al., 2025), for Muon often require spectral bounds ($\sigma_{\min}$), which are hard to know in practice.

Efficient computation of matrix functions

Why matrix functions matter here

The expensive part is repeatedly computing $(\cdot)^{-1}$, $\mathrm{polar}(\cdot)$, or $(\cdot)^{-\alpha}$ for many blocks. SVD/eigendecomp is accurate but too slow inside large-scale training.

Our goals:

- **Scalability:** GPU-friendly, **GEMM-dominant** iterations (matrix–matrix multiplies) instead of SVD/eigendecomposition.
- **Spectral adaptivity:** automatic, spectrum-aware acceleration across datasets and training stages (no hand-tuned spectral bounds).
- **Generality:** one unified framework that supports many matrix functions (roots, inverse roots, sign, polar/orthogonalization, ...).

PRISM: **P**olynomial-fitting and **R**andomized
Iterative **S**ketching for **M**atrix-function computation

PRISM Meta-algorithm for Computing $T(A)$

PRISM is a *meta-algorithm*: once you express the base iteration as $X_{k+1} = X_k p(R_k)$, PRISM adapts coefficient of p to the current spectrum rather than deterministic values.

Part I: put your method in a polynomial form	Part II: accelerate by fitting + random sketching
1. Write target as $T(a) = x\, f(\xi)$ with residual $\xi = \xi(x, a)$.	1. Replace Taylor with a fitted polynomial $g_d(\xi; \alpha) = f_{d-1}(\xi) + \alpha \xi^d$.
2. Use d-degree Taylor approximation of f at $\xi = 0$ to get $x_{k+1} = x_k f_d(\xi(x_k, a))$.	2. Choose α_k to minimize the next residual (Frobenius norm).
3. Lift to matrices: $X_{k+1} = X_k f_d(R_k)$ with residual matrix $R_k = \xi(X_k, A)$.	3. Use randomized sketching so computing α_k is cheap.

Randomized sketching: keep the overhead negligible

Directly computing α_k^* can require traces of high powers of R_k (expensive).
PRISM replaces the objective with a sketched one:

$$\tilde{\alpha}_k = \arg\min_{\alpha \in [\ell, u]} \left\| S_k \left(I - X_k^2 \, g_d(R_k; \alpha)^2 \right) \right\|_F^2,$$

where $S_k \in \mathbb{R}^{p \times n}$ is a small random sketch (e.g., Gaussian).

- Cost to evaluate coefficients becomes $\mathcal{O}(n^2 p)$ instead of $\mathcal{O}(n^3)$.
- In practice, very small p can work; theory suggests $p = \mathcal{O}(\log n)$ for guarantees.

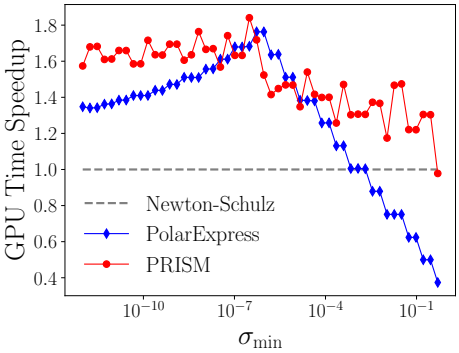
PRISM-accelerated iterations

Method	Target	Initialization	Iteration (PRISM form)	Residual
Newton-Schulz (3rd-order)	$A^{1/2}$ $A^{-1/2}$	$X_0 = A$ $Y_0 = I$	$X_{k+1} = X_k(I + \alpha_k R_k)$ $Y_{k+1} = (I + \alpha_k R_k)Y_k$	$R_k = I - X_k Y_k$
Newton-Schulz (5th-order)	$A^{1/2}$ $A^{-1/2}$	$X_0 = A$ $Y_0 = I$	$X_{k+1} = X_k(I + \frac{1}{2}R_k + \alpha_k R_k^2)$ $Y_{k+1} = (I + \frac{1}{2}R_k + \alpha_k R_k^2)Y_k$	$R_k = I - X_k Y_k$
Newton-Schulz (3rd-order)	UV^T	$X_0 = A$	$X_{k+1} = X_k(I + \alpha_k R_k)$	$R_k = I - X_k^T X_k$
Newton-Schulz (5th-order)	UV^T	$X_0 = A$	$X_{k+1} = X_k(I + \frac{1}{2}R_k + \alpha_k R_k^2)$	$R_k = I - X_k^T X_k$
Coupled Inverse Newton	$A^{-1/p}$ $(p \geq 1)$	$X_0 = I$ $M_0 = A$	$X_{k+1} = X_k(I + \alpha_k R_k)$ $M_{k+1} = (I + \alpha_k R_k)^p M_k$	$R_k = I - X_k^p A$
DB (Denman-Beavers) Newton	$A^{1/2}$ $A^{-1/2}$	$X_0 = A$ $Y_0 = I$	$X_{k+1} = (1-\alpha_k)X_k + \alpha_k Y_k^{-1}$ $Y_{k+1} = (1-\alpha_k)Y_k + \alpha_k X_k^{-1}$	$R_k = I - X_k Y_k$
Chebyshev	A^{-1}	$X_0 = A^T$	$X_{k+1} = X_k(I + R_k + \alpha_k R_k^2)$	$R_k = I - AX_k$

- α_k is chosen by a sketched least-squares fit that minimizes the next residual.

Why adaptivity matters: spectral mismatch can hurt

- PolarExpress (Amsel et al., 2025) optimizes polynomials for a *predefined* singular-value interval.
- If the true spectrum is narrower/wider, convergence can degrade (or even slow down).
- PRISM is **distribution-free**: it adapts to the evolving spectrum without explicit spectral bounds.



Speedup in GPU time over the classical Newton-Schulz for square root.

Towards the future …

- "Spectral Estimation with Free Decompression," Ameli et al. arXiv:2506.11994, NeurIPS 2025.
- "Random Matrix Theory for Deep Learning: Beyond Eigenvalues of Linear Models," Liao et al. arXiv:2506.13139, IEEE SPM (2026).
- "PRISM: Distribution-free Adaptive Computation of Matrix Functions for Accelerating Neural Network Training," Yang et al. arXiv:2601.22137, ICML 2026.
- "Learning to Discover Iterative Spectral Algorithms," Liu et al. arXiv:2602.09530.

Fundamental Data Primitives for Structural Exploitation

Every data modality has fundamental primitive for structure exploitation

- Image Data → Convolution (spatial filtering)
- Language → Attention (relational filtering)
- Linear Operators → Iterative matrix functions (matrix polynomial/rational filtering)

Frameworks for constructing iterative matrix functions
are of paramount importance in NLA and ML

Role of Iterative Matrix Functions in Numerical Linear Algebra

- Linear systems / preconditioning
  - Polynomial preconditioners reshape the spectrum for faster convergence
  - Neumann, Chebyshev, minimax, least-squares
- Eigenvalue problems
  - Polynomial filters amplify target eigenspaces
  - Used in filtered subspace iteration, Krylov, Davidson, and related eigensolvers
- Matrix functions
  - Matrix-free approximation of matrix function actions
  - Explicit approximation of a matrix function
  - Neumann, Chebyshev, Legendre, Newton-Schulz, Inverse Newton, DB Newton recurrences

Why iterative matrix functions are important in Numerical Linear Algebra?

- Hardware/communication-friendly
- Matrix-free implementations for large sparse problems
- Strong theoretical foundations
- Controllable tradeoffs: accuracy vs. compute vs. stability

Role of Iterative Matrix Functions in Machine Learning

- Whitening / decorrelation / normalization
  - Approximate inverse-roots of covariance and related matrices
  - Used for whitening, decorrelation, and second-order feature normalization
- Linear solves in ML
  - Many ML methods require solving linear systems
    Examples: ridge regression, Gauss-Newton, natural gradient, kernel methods, and implicit layers
  - Polynomial approximations can provide matrix-free approximate solves or preconditioning
- Second-order / preconditioned optimizers
  - Require matrix transforms such as inverse, inverse-root, and polar factors
  - Examples: Shampoo, Muon, PolarExpress optimizers

Why iterative matrix functions are important in ML?

- Replaces expensive matrix functions with cheap iterative kernels
- Makes advanced preconditioning / normalization feasible in large-scale training
- Implicitly regularizes matrix transforms

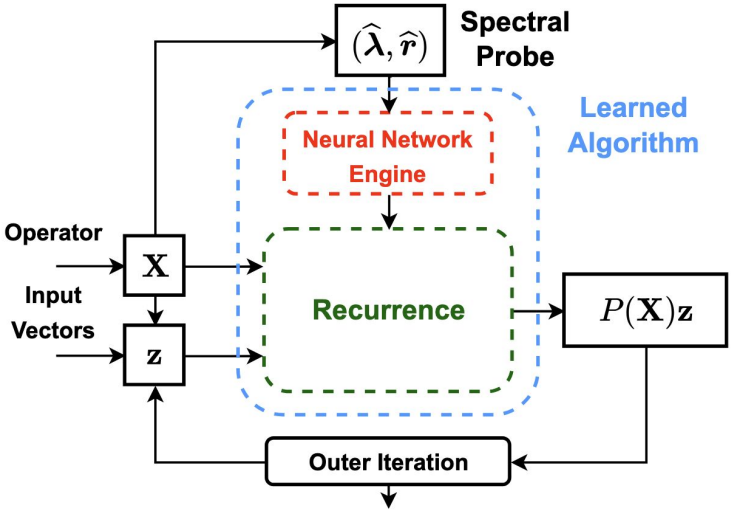
AutoSpec: End-to-end Automated Discovery of Iterative Matrix Functions

Train a **Neural Network Engine** that

- takes as input a **Spectral Probe** of operator $\mathbf{X}$
- outputs the **Recurrence Coefficients** representing action of $P(\mathbf{X})$ on vectors $\mathbf{z}$: $(\mathbf{X}, \mathbf{z}) \longrightarrow P(\mathbf{X})\mathbf{z}$

Key components:

- End-to-end differentiable architecture that resembles NLA recurrences
- Effective self-supervised training on small synthetic matrices
- Transfers to large-scale real-world operators with varying sizes and properties
- A task-defined objective that enforces desired spectral properties uniformly on training data



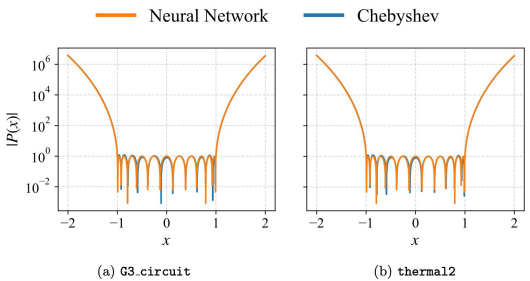
AutoSpec: Advancing State-of-the-art NLA

AutoSpec recurrences possess properties analogous to Chebyshev polynomials

- Minimax optimality

Matrix	Minimax Optimality Gap (↓)
thermal1	9.390×10^{-7}
thermal2	9.331×10^{-7}
Si02	8.656×10^{-7}
C0	8.670×10^{-7}
Random	0.954
Optimal	2.850×10^{-7}

- Equiripple behavior



(a) G3_circuit (b) thermal2

AutoSpec often outperforms Chebyshev accelerators

(a) Linear systems

Matrix X	numIters (AutoSpec, probe=200)	numIters (AutoSpec, probe=50)	numIters (Cheb, probe=200)	numIters (Cheb, probe=50)
thermal1	**173**	178	221	**168**
thermal2	686	705	**646**	**658**
nos5	**75**	77	100	79
wathen100	**36**	**35**	78	58
thermomech_TC	10	11	11	10
G2_circuit	**69**	**70**	129	79
gyro_m	**28**	**27**	141	98
Dubcova2	**22**	**22**	34	37
Flan_1565	**366**	**375**	496	374
G3_circuit	**100**	101	125	100

Conclusions

AutoSpec is a neural network framework for discovering spectrum-adaptive iterative algorithms for NLA tasks

- AutoSpec algorithms combine:
  - neural network engine
  - executable NLA recurrence
- AutoSpec provides an end-to-end differentiable architecture, enabling self-supervised training by backpropagating task-defined NLA losses.
- AutoSpec neural engine can be trained effectively on small synthetic matrices:
  - NLA losses enforce desired spectral behavior uniformly across training instances
  - Algorithms generalize across spectral shapes
- AutoSpec-generated recurrences consistently provide significant improvements on convergence and/or accuracy relative to the training baselines

Summary

Past: RandNLA has by now a long history of:

theory, applications, and implementations

Present: RandNLA is at a transition point:

new theory, applications, and hardware

Future: RandNLA's future:

is up to you!