TAMING ILL-CONDITIONED PROBLEMS IN MACHINE LEARNING: ALGORITHMS, ANALYSIS, AND SOFTWARE

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Pratik Rathore
May 2026

Abstract

Ill-conditioning is pervasive in machine learning and is the dominant obstacle to solving many problems at scale. Datasets, kernel matrices, and neural-network loss landscapes are routinely ill-conditioned, which causes first-order optimizers to converge slowly, while classical second-order methods that are robust to ill-conditioning become computationally infeasible at modern scales. This dissertation develops algorithms, theoretical analyses, and software that correct ill-conditioning in a variety of machine learning settings. First, we introduce `ASkotch`, a memory-efficient iterative method for full kernel ridge regression that combines sketch-and-project iterative solvers with the randomized Nyström approximation. `ASkotch` enjoys condition-number-free linear convergence and scales full KRR to datasets with hundreds of millions of samples on a single GPU, a regime where previous state of the art methods fail to complete a single iteration. Second, we analyze the optimization landscape of physics-informed neural networks (PINNs) and show how ill-conditioned differential operators in the residual term make the PINN loss hard to minimize. We establish that a pipeline applying Adam followed by L-BFGS is a strong default for training PINNs, and develop NysNewton-CG (NNCG), a second-order optimizer that rescues PINN training on problems where this pipeline stalls. Third, we develop `rlaopt`, an open-source PyTorch package that makes scalable preconditioned optimization accessible to practitioners, with GPU-enabled solvers for a broad class of machine learning problems, a CVXPY-style modeling language, and end-to-end differentiability. Together, these contributions show that ill-conditioning need not be an obstacle to scalable machine learning.

Acknowledgments

I would like to start by thanking my PhD advisor, Madeleine Udell. It has been a pleasure to work with her during my time at Stanford. When I started my PhD, I was completely new to academic research. Madeleine took me under her wing and showed me what it takes to see a research project through, from generating ideas, to executing a research plan, to writing and presenting my findings. If there is one thing I will always remember throughout my career, it's that figures should always be PDFs!

I would like to thank my committee members, Stephen Boyd, Michał Dereziński, Ching-Yao Lai, and Mert Pilanci. I had the pleasure of learning optimization from both Stephen and Mert during my first year at Stanford, which inspired me to pursue projects in optimization for much of my PhD. I am grateful to Michał for his sharp insights on our collaborations involving kernels and Gaussian processes and to Ching-Yao for her guidance on our project involving subsurface imaging.

My PhD would not have been the same without my good friends Joe Stitt and Zachary Frangella. I have collaborated closely with both these guys during my PhD, and have learned so much from working alongside them.

My friendship with Joe started on my first day at Stanford, when we were assigned the same apartment in grad housing. We navigated that first year at Stanford together, figuring out how to adjust to the PhD life. I'll miss my regular weekend outings with Joe, especially getting those beer flights at Fieldwork.

My friendship with Zach started when I joined Madeleine's group, where we quickly bonded over our common interests in research. Zach and I have spent countless hours together, thinking about research questions, working together as course assistants, and powering through to meet conference deadlines. I'll miss Zach's great sense of humor and wit, not to mention his excellent taste in restaurants.

I had the opportunity to intern at several companies during my PhD, and these experiences have been formative for me as a researcher. I would like to thank Steven Diamond, Amir Zargari, Stuart Farris, and Allen Zhao for these opportunities. At Gridmatic, Steven showed me the value of optimization in electricity markets. At Skyworks, Amir gave me a window into how optimization and AI are reshaping semiconductor design. At Hestia, Stuart and Allen have shown me what it

takes to contribute to an early-stage startup.

I would like to thank my colleagues, Rustam Akhmadiev, Thomas Cullison, Shaghayegh Fazliani, Zachary Frangella, Sachin Garg, Alexa Hu, Weimu Lei, Lu Lu, Parth Nobel, Joshua Rines, Joe Stitt, Jiaming Yang, and Shipu Zhao. I have really enjoyed working with and learning from all of you!

The Udell group has been a welcoming and collaborative environment for me during my PhD. I would like to thank Ya-Chi Chu, Shaghayegh Fazliani, Zachary Frangella, Wenzhi Gao, Connor Lawless, Weimu Lei, Ali Teshnizi, Mike van Ness, Yinjun Wang, and Wanyu Zhang for making the Udell group a great place to be. I would also like to thank Matt Gonzales for all his help on the administrative side of things over the years!

I am grateful for all the friends I have made during my PhD, especially Saksham Consul, Joshua Dong, Hemanth Kollipara, Evan Rees, Stuti Saria, and Anders Wikum. Saksham and Josh, I will remember our hiking trips and hangouts in San Francisco. Hemanth, I really enjoyed working with you at Skyworks and going to Palm Springs. Evan, it was a lot of fun living in that Airbnb in Costa Mesa, and thanks for getting me hip to Buldak ramen. Stuti, I will miss those banger Diwali parties. Anders, I will fondly remember hanging out with you in Vienna and our discussions in the office.

I also want to thank my friends from high school and college, particularly Suyaib Ahmed, Ethan Chen, Guang Cui, Dawson Do, and Daniel Kline. Suyaib and Daniel, our trip to Florida was a great time, and it was a lot of fun watching movies together on the weekends. Ethan, I really enjoyed having you as a roommate during the final year of my PhD. Guang, it's always great catching up with you, whether in San Francisco or back home in Maryland. Dawson, it's been great hanging out, whether it's been watching NBA playoff games, going hiking, or trying out Vietnamese food in the Bay.

Finally, I would like to thank my family. My sister, Khushboo, has always been supportive to me during my PhD. My parents, Pooja and Dharmendar, encouraged me to pursue my dreams, which led me to Stanford. My road to pursuing a PhD started when my mom started teaching me math at home as a kid, which led me to become passionate about math in middle and high school, pursuing math and electrical engineering degrees in college, and finally a PhD in electrical engineering at Stanford. It's safe to say that I would not be where I am today without my family by my side.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Ill-conditioning is pervasive in machine learning. Datasets used for regularized empirical risk minimization (ERM) are routinely ill-conditioned, with data-matrix condition numbers ranging from 10^4 to 10^8 or larger [Frangella et al., 2024b]. Kernel matrices arising in large-scale kernel ridge regression (KRR) exhibit approximate low-rank structure, due to rapid spectral decay [Caponnetto and DeVito, 2007, Bach, 2013, Tu et al., 2016, Ma and Belkin, 2017, Belkin, 2018]. The differential operators that appear in the losses of physics-informed neural networks (PINNs) induce such severe ill-conditioning that practitioners find that quasi-Newton methods are necessary for effective training [Rathore et al., 2024].

Ill-conditioning matters because it is the dominant obstacle to solving machine learning problems at scale. First-order methods such as stochastic gradient descent (SGD) and Adam [Kingma and Ba, 2014] enjoy cheap per-iteration costs and parallelize naturally on modern hardware, but their convergence is governed by the problem's condition number and so they are slow on ill-conditioned problems. Classical second-order methods such as Newton's method and BFGS [Broyden, 1970, Fletcher, 1970, Goldfarb, 1970, Shanno, 1970] are robust to ill-conditioning but require either expensive matrix factorizations or storage costs that are computationally infeasible at the scales considered in modern machine learning. Consequently, practitioners are caught between scalable-but-slow first-order methods and fast-but-infeasible second-order methods.

The way out of this dilemma is scalable preconditioning: cheaply transform the problem (or a subproblem) so that its condition number is small, (approximately) recovering the fast convergence of second-order methods without paying their full cost. Different settings call for different preconditioning tools. For convex ERM problems, randomized low-rank approximations (the Nyström approximation in particular) capture the dominant eigenvalues cheaply and yield preconditioners that can

be rapidly constructed and applied to gradients [Frangella et al., 2023, 2024b,a]. For KRR, randomized low-rank approximations can be combined with sketch-and-project iterative solvers [Gower and Richtárik, 2015] to yield memory-efficient methods with condition-number-free linear convergence [Rathore et al., 2026]. For PINN losses, quasi-Newton methods like L-BFGS [Liu and Nocedal, 1989] approximate curvature from gradient history without ever forming a Hessian and remain practical at the scales encountered in PINN training. For composite optimization, operator splitting reduces the problem to a sequence of simpler subproblems whose conditioning can be controlled through preconditioned inner solvers [Zhao et al., 2022, Diamandis et al., 2026]. In each case, the preconditioner sidesteps the prohibitive factorization and storage costs of classical second-order methods while still capturing enough curvature information to dramatically improve conditioning.

This dissertation identifies where ill-conditioning arises in three important machine learning settings, explains why it limits the state of the art, and develops algorithms, analyses, and software that correct it: kernel ridge regression (Chapter 2), the optimization of physics-informed neural networks (Chapter 3), and a unified software framework that makes scalable preconditioned optimization accessible to practitioners (Chapter 4).

1.2 Contributions and Outline

The core contributions of this dissertation are new algorithms, theoretical analyses, and software for ill-conditioned problems in machine learning. We summarize the content and contributions of each chapter below.

Chapter 2. We develop `ASkotch`, an iterative method for solving full kernel ridge regression (KRR). `ASkotch` combines sketch-and-project iterative solvers [Gower and Richtárik, 2015] with the randomized Nyström approximation [Frangella et al., 2023]. For an $n \times n$ kernel matrix, `ASkotch` enjoys per-iteration cost that is linear in n and storage cost that is independent of n. We establish fine-grained convergence guarantees for `ASkotch` via a first-moment analysis of the *Nyström projector*, which relies on a novel reduction from approximate ridge leverage score sampling to determinantal point processes. When the effective dimension of the kernel matrix is not too large, `ASkotch` enjoys condition-number-free linear convergence and achieves a near-optimal $\tilde{\mathcal{O}}(n^2 \log(1/\epsilon))$ total computational complexity. Empirically, `ASkotch` outperforms preconditioned conjugate gradient (PCG), EigenPro 2.0 [Ma and Belkin, 2019], EigenPro 3.0 [Abedsoltan et al., 2023], and Falkon [Rudi et al., 2017, Meanti et al., 2020] on predictive performance across 23 large-scale KRR problems, and scales full KRR to a dataset with $n = 10^8$ samples on a single 48 GB GPU—a regime in which PCG cannot complete a single iteration.

Chapter 3. We study the optimization landscape of physics-informed neural networks (PINNs) and show how ill-conditioning of the PINN loss restricts the application of these methods. We demonstrate empirically that the PINN loss is highly ill-conditioned due to the differential operators

in the residual term, and that quasi-Newton methods improve the conditioning of the loss by 1000× or more. We show that Adam+L-BFGS, a pipeline that applies Adam followed by L-BFGS, is a superior default compared to either optimizer alone, but can stall at a high loss when the L-BFGS line search fails to find a satisfactory step size. To address this failure mode, we introduce NysNewton-CG (NNCG), a second-order optimizer that uses a randomized Nyström approximation of the Hessian to precondition a Newton-CG solve, and show empirically that it rescues training when Adam+L-BFGS stalls. We also prove that an ill-conditioned linear differential operator induces an ill-conditioned PINN loss, giving a theoretical justification for our empirical observations.

Chapter 4. Despite the strong algorithmic foundations of RandNLA-based optimization methods, there is a gap between the algorithms developed in the literature and the software available to practitioners: existing implementations do not leverage modern parallel hardware and do not provide a user-friendly interface for modeling optimization problems. We close this gap with `rlaopt`, an open-source, PyTorch-based software package for large-scale optimization using RandNLA. `rlaopt` provides GPU-enabled implementations of NyströmPCG [Frangella et al., 2023] for large-scale linear systems, NysADMM [Zhao et al., 2022, Diamandis et al., 2026] for constrained convex optimization, and SAPPHIRE [Frangella et al., 2024a, Sun et al., 2025] for preconditioned stochastic variance-reduced empirical risk minimization. A flexible modeling language, inspired by CVXPY [Diamond and Boyd, 2016], allows users to specify problems using natural mathematical syntax, and the package supports differentiating through the solver for end-to-end learning. We demonstrate GPU speedups of up to 20× for NyströmPCG, 20–30× for NysADMM, and 5–10× for SAPPHIRE over CPU implementations on benchmark problems.

Chapter 5. We summarize the contributions of the dissertation and discuss promising directions for future research.

Taken together, these contributions identify where ill-conditioning arises in important machine learning problems, show why it limits the state of the art, and develop scalable algorithms and software that correct it.

1.3 Other Publications and Projects

In addition to the contributions covered in this dissertation, I have co-authored three more publications during my PhD: [Frangella et al., 2024b] (SIMODS '24), [Frangella et al., 2024a] (JMLR '24), and [Rathore et al., 2025] (NeurIPS '25).

Chapter 2

Have ASkotch: A Neat Solution for Large-scale Kernel Ridge Regression

We develop `ASkotch`, an iterative method for solving large-scale kernel ridge regression (KRR) that combines sketch-and-project with the randomized Nyström approximation. `ASkotch` enjoys per-iteration cost linear in the number of samples n, storage independent of n, and condition-number-free linear convergence when the effective dimension of the kernel matrix is not too large. Empirically, `ASkotch` outperforms state-of-the-art KRR solvers across 23 large-scale datasets and scales full KRR to a dataset with $n = 10^8$ samples on a single 48 GB GPU, a regime in which PCG cannot complete a single iteration.

This chapter is based on Rathore et al. [2026], currently under review at the Journal of Machine Learning Research.

2.1 Introduction

Kernel ridge regression (KRR) is one of the most popular methods in machine learning, with important applications in computational chemistry [Stuke et al., 2019, Blücher et al., 2023, Parkinson and Wang, 2023], healthcare [Cheng et al., 2020, Wu et al., 2021, Townes and Engelhardt, 2023], and, more recently, scientific machine learning [Raissi et al., 2017, Meanti et al., 2024, Batlle et al., 2024]. Given a kernel function $k(x, x')$ and training set $\{(x_i, y_i)\}_{i=1}^n$, *full KRR* seeks the function f

in a reproducing kernel Hilbert space $\mathcal{H}$ that satisfies

$$\underset{f\in\mathcal{H}}{\text{minimize}}\ \frac{1}{2}\sum_{i=1}^{n}\left(f(x_i)-y_i\right)^2+\frac{\lambda}{2}\|f\|_{\mathcal{H}}^2. \tag{2.1}$$

The celebrated representer theorem [Kimeldorf and Wahba, 1970, Schölkopf and Smola, 2002] says that the solution of (2.1) lies in the subspace

$$\mathcal{H}_n=\left\{f\in\mathcal{H}: f(x)=\sum_{j=1}^{n}w^{(j)}k(x,x_j),\ \text{where } w^{(j)}\in\mathbb{R}\right\}.$$

With this reduction, the infinite-dimensional optimization problem in (2.1) collapses to a finite-dimensional, convex, least-squares problem:

$$\underset{w\in\mathbb{R}^n}{\text{minimize}}\ \mathcal{L}_{\text{full}}(w)\coloneqq\frac{1}{2}\|Kw-y\|^2+\frac{\lambda}{2}\|w\|_K^2. \tag{2.2}$$

Here K is a $n\times n$ kernel matrix whose entries are given by $K_{ij}=k(x_i,x_j)$. The optimal weights $w_\star$ in (2.2) are the solution of the linear system

$$K_\lambda w_\star=y, \tag{2.3}$$

where $K_\lambda\coloneqq K+\lambda I$.

Despite its popularity, full KRR is challenging to scale to large datasets: the state-of-the-art methods for solving (2.3) have costs that are superlinear in n. Direct methods (e.g., Cholesky decomposition) have $\mathcal{O}(n^3)$ computational complexity and $\mathcal{O}(n^2)$ storage complexity. Therefore, when $n\gtrsim 10^4$, Cholesky decomposition becomes unsuitable for solving (2.3). Iterative methods such as preconditioned conjugate gradient (PCG) have per-iteration complexity $\mathcal{O}(n^2)$. In addition to expensive iterations, most PCG methods employ low-rank preconditioners, which require $\mathcal{O}(nr)$ storage, where r is a rank parameter which needs to be sufficiently large to ensure effective preconditioning. Therefore, when $n\gtrsim 10^6$, PCG is either (i) too slow or (ii) requires too much memory to solve (2.3).

The standard approach for addressing the scalability issues of full KRR is to select a set of m *inducing points* from the training set, and solve the *inducing points KRR* problem [Schölkopf and Smola, 2002, Rudi et al., 2015]. The inducing points KRR objective is

$$\underset{w\in\mathbb{R}^m}{\text{minimize}}\ \mathcal{L}_{\text{ind}}(w)\coloneqq\frac{1}{2}\|K_{nm}w-y\|^2+\frac{\lambda}{2}\|w\|_{K_{mm}}^2, \tag{2.4}$$

where K_{nm} is the $n\times m$ matrix of the columns of K identified by the inducing points, and K_{mm} is the corresponding principal submatrix of K. The optimal weights $w_\star$ in (2.4) are the solution of

the linear system

$$(K_{nm}^T K_{nm} + \lambda K_{mm}) w_\star = K_{nm}^T y. \tag{2.5}$$

The linear system (2.5) accesses K only through K_{nm} and K_{mm}, reducing the storage and per-iteration computation needed to solve the KRR problem. As a result, Cholesky decomposition has $\mathcal{O}(nm^2 + m^3)$ computational complexity and requires $\mathcal{O}(m^2)$ storage. Consequently, when $m \gtrsim 10^5$, Cholesky is unsuitable for solving (2.5). State-of-the-art PCG methods for inducing points KRR, like Falkon [Rudi et al., 2017, Meanti et al., 2020] and KRILL [Díaz et al., 2023], require $\mathcal{O}(m^3)$ time to construct the preconditioner and $\mathcal{O}(m^2)$ storage. Thus, when $m \gtrsim 10^5$, these PCG methods are also unsuitable for solving (2.5).

Alas, scale matters: previous work has established that increasing the number of inducing points leads to better predictive performance [Frangella et al., 2023, Díaz et al., 2023, Abedsoltan et al., 2023]. Moreover, full KRR often allows better predictive performance than inducing points KRR [Wang et al., 2019, Frangella et al., 2023, Díaz et al., 2023]. Therefore, a new, more scalable approach to KRR is needed for better prediction on large-scale tasks.

In this work, we turn conventional wisdom around, demonstrating that better, faster solutions can be achieved by targeting the full KRR problem rather than the inducing points KRR approximation. We do so by developing the iterative method `ASkotch` to solve full KRR. `ASkotch` has (i) per-iteration costs that are linear in n and (ii) storage costs that are linear in the preconditioner rank r but are independent of n. These properties allow `ASkotch` to scale to larger datasets than existing state-of-the-art methods for solving (2.3). `ASkotch` possesses strong theoretical guarantees and easy-to-set hyperparameters that deliver reliable performance. Furthermore, `ASkotch` exploits parallelism on modern hardware accelerators like GPUs, which is essential for scaling to large KRR problems.

Fig. 2.1 demonstrates the core messages of our work—full KRR can scale to massive datasets and outperform inducing points KRR with respect to predictive performance. On a dataset with $n = 10^8$ samples, `ASkotch` scales better than the standard approach to full KRR: PCG is unable to complete a single iteration within a 24-hour time limit. Moreover, `ASkotch` is more reliable than the KRR solvers EigenPro 2.0 and EigenPro 3.0 [Ma and Belkin, 2019, Abedsoltan et al., 2023] when all three methods are run with their default hyperparameters: EigenPro 2.0 and EigenPro 3.0 diverge, while `ASkotch` reaches a root mean square error (RMSE) of approximately 230. Finally, `ASkotch` outperforms Falkon on RMSE. While Meanti et al. [2020] scales an optimized version of Falkon to the full taxi dataset ($n \approx 10^9$) with $m = 10^5$ inducing points, `ASkotch` is solving a problem of size $10^8 \cdot 10^8 = 10^{16}$, which is two orders of magnitude larger than the problem solved by Falkon ($10^5 \cdot 10^9 = 10^{14}$). For additional discussion, please see Section 2.3.2.

Table 2.1 compares the capabilities of `ASkotch` to state-of-the-art methods for KRR. `ASkotch` is the only method that solves full KRR while having a moderate memory requirement, reliable default hyperparameters, and theoretical convergence guarantees. In the EigenPro GitHub repos, there is no option for the user to set hyperparameters such as the learning rate or gradient batch size. Since

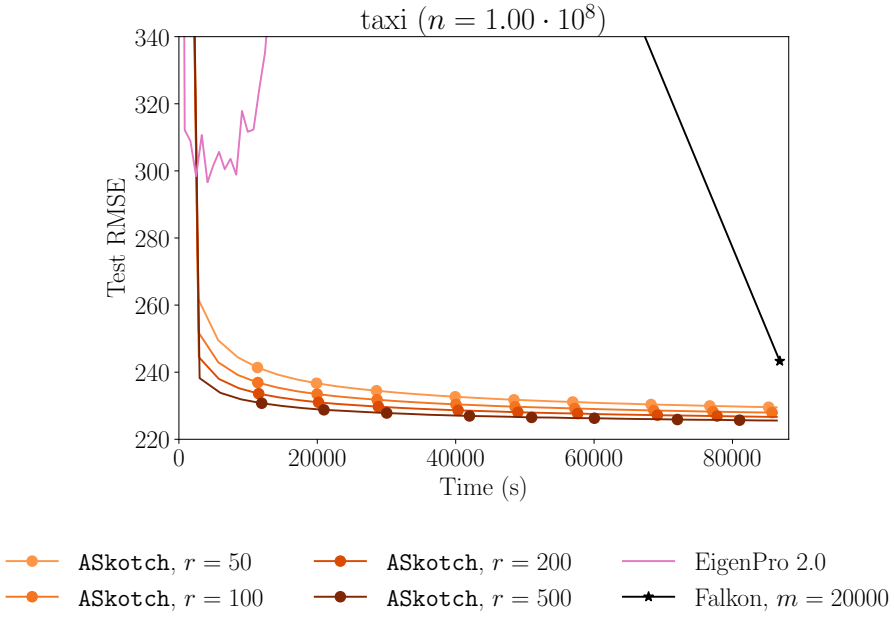


Figure 2.1: Full KRR is advantageous over inducing points KRR, even for large problems. Our method `ASkotch`, run with its default hyperparameters, outperforms the state-of-the-art for both full and inducing points KRR on a subsample of the taxi dataset. Falkon is limited to $m = 2 \cdot 10^4$ inducing points due to memory constraints. State-of-the-art Nyström PCG methods [Frangella et al., 2023] Gaussian Nyström [Frangella et al., 2023] and Randomly Pivoted Cholesky [Díaz et al., 2023, Epperly et al., 2025], each with a rank $r = 50$ preconditioner, fail to complete a single iteration. EigenPro 2.0 and EigenPro 3.0 (not shown) diverge on their default hyperparameters. All methods have a 24-hour time limit and are run on a single 48 GB NVIDIA RTX A6000 GPU.

the EigenPro methods diverge on taxi and several other datasets in this work, we conclude that the default hyperparameters for EigenPro can be unreliable.

Table 2.2: Iteration and storage complexities for state-of-the-art full KRR methods. Storage complexity refers to memory that is explicitly used by the algorithm (this excludes matrix-vector product oracles). Total computational complexity refers to the total cost required by the algorithm to compute an ϵ-approximate solution, omitting logarithmic dependencies upon ϵ. Here, $b_g \leq n$ is the stochastic gradient batch size used by EigenPro 2.0, $s \leq n$ is the sample size used to construct the rank-r preconditioner in EigenPro 2.0, $b \leq n$ is the blocksize in `ASkotch`, and $d^\lambda(K)$ is the λ-effective dimension of K (Definition 2.4). `ASkotch` has the same total cost as PCG but has lower storage and per-iteration costs.

Algorithm	Per-iteration complexity	Storage complexity	Total computational complexity
PCG	$\mathcal{O}(n^2)$	$\mathcal{O}(d^\lambda(K)n)$	$\tilde{\mathcal{O}}(n^2)$
EigenPro 2.0	$\mathcal{O}(nb_g + rs)$	$\mathcal{O}(rs)$	$\tilde{\mathcal{O}}\left(\frac{\lambda_r(K)}{\lambda_{\min}(K)}n^2\right)$
`ASkotch`	$\mathcal{O}(nb)$	$\mathcal{O}\left(d^\lambda(K)b\right)$	$\tilde{\mathcal{O}}\left(n^2\right)$

Table 2.1: A comparison of the capabilities of `ASkotch` and state-of-the-art methods for KRR. "Reliable defaults" indicates whether the method comes with default hyperparameters that work well in practice. "Converges" indicates whether the method has a rigorous linear convergence guarantee.

Algorithm	Full KRR?	Memory-efficient?	Reliable defaults?	Converges?
`ASkotch`	✓	✓	✓	✓
EigenPro 2.0	✓	✓	✗	✓
EigenPro 3.0	✗	✓	✗	✗
PCG	✓	✗	✓	✓
Falkon	✗	✗	✓	✓

2.1.1 Contributions

We make significant algorithmic and theoretical contributions to the literature on KRR solvers and sketch-and-project methods.

1. We develop the iterative method `ASkotch` for solving full KRR. The key to our approach lies in carefully combining two methodologies: sketch-and-project iterative solvers [Gower and Richtárik, 2015] and Nyström matrix approximations [Williams and Seeger, 2000, Tropp et al., 2017]. We incorporate the Nyström approximation into the sketch-and-project update in a way that preserves convergence, and combine this with Nesterov acceleration to solve (2.3) with reduced time and memory requirements.

2. We establish fine-grained convergence guarantees for `ASkotch` based on ridge leverage score sampling, determinantal point processes, and careful spectral approximation arguments. A detailed overview of our theoretical contributions is presented in Section 2.1.2.

3. We perform an extensive empirical evaluation comparing `ASkotch` to the existing state-of-the-art for full and inducing points KRR. We show that `ASkotch` outperforms EigenPro 2.0, EigenPro 3.0, PCG, and Falkon on predictive performance for 23 large-scale KRR problems (typically, $n \geq 10^5$). Moreover, we show that full KRR consistently obtains equivalent or better predictive performance than inducing points KRR, establishing the superiority of full KRR when combined with `ASkotch`. We provide open-source code in PyTorch to make it easy to reproduce our experiments and run `ASkotch` on new problems on CPU and GPU hardware.

2.1.2 Our Techniques

Our main theoretical contributions lie in the convergence analysis of `ASkotch`. Establishing convergence of `ASkotch` requires overcoming two fundamental challenges, discussed below.

Technical Contribution 1: *First-moment analysis of SAP projector via ARLS-to-DPP reduction.* Our first contribution overcomes a fundamental limitation in the existing theory of sketch-and-project (SAP) solvers. The convergence of SAP methods is controlled by a random projection

matrix that we call the *SAP projector*, with the convergence rate being determined by the smallest eigenvalue of the expected SAP projector. Existing approaches for analyzing the smallest eigenvalue of the expected SAP projector [e.g., Mutny et al., 2020, Dereziński and Yang, 2024, Dereziński et al., 2025a] require either (1) an expensive, impractical sub-sampling scheme known as a Determinantal Point Process (DPP), or (2) preprocessing the input matrix with the Randomized Hadamard Transform, which is prohibitive for large problems due to its $\mathcal{O}(n^2)$ storage cost.

To address this challenge, we adopt a different sampling scheme called approximate ridge leverage score (ARLS) sampling [Alaoui and Mahoney, 2015], which is less expensive (both in compute and storage) than either of these approaches. Our convergence analysis develops a reduction from ARLSs to DPPs (Lemma 2.12), by relating the former to the marginals of a DPP through a novel measure concentration argument. This reduction allows us to establish a non-trivial lower bound on the smallest eigenvalue of the expected SAP projector when using ARLS sampling (Lemma 2.10). We believe this result is of independent interest and could be used to improve the convergence analysis of other iterative solvers that use the sketch-and-project paradigm.

Technical Contribution 2: *First-moment analysis of Nyström projector.* `ASkotch` replaces the linear system solve in SAP with an approximate solution based on Nyström sketch-and-solve [Bach, 2013, Alaoui and Mahoney, 2015, Frangella et al., 2023]. In contrast, prior sketch-and-project solvers [Gower and Richtárik, 2015, Gower et al., 2018, Dereziński and Yang, 2024, Dereziński et al., 2025a] assume the SAP linear system is either solved exactly or to sufficient accuracy at each iteration using an iterative solver. The use of Nyström sketch-and-solve complicates the analysis in two ways: 1) the convergence rate is no longer controlled by an expected projection matrix; 2) the Nyström solution is generally not close to the exact solution [Frangella et al., 2023]. The first issue prevents us from directly applying the classical convergence analysis of SAP, which is deeply reliant on the SAP projector actually being a projection matrix. The second issue means we cannot appeal to existing inexact SAP theory, which requires the approximate solution to be close to the exact solution.

Remarkably, although the matrix that controls convergence of `ASkotch` is not the expectation of a projection matrix, it is the expectation of a matrix that is *nearly* a projection matrix. We refer to this matrix as the *Nyström projector*. Using careful spectral approximation arguments, we show that the expected Nyström projector is lower bounded in the Loewner ordering by the expected SAP projector multiplied by a factor that captures the price of replacing the SAP linear system solve with Nyström sketch-and-solve. Thus, we are able to transfer the first-moment projector analysis from Theoretical Contribution 1 (which is for exact SAP) to `ASkotch`, which yields a lower bound on the smallest eigenvalue of the expected Nyström projector (Theorem 2.13). This lower bound on the smallest eigenvalue of the expected Nyström projector lets us establish a fine-grained convergence rate for `ASkotch` (Theorem 2.17). Using this theory, we prove that when the effective dimension of the kernel matrix is not too large, `ASkotch` enjoys fast condition-number-free linear convergence, for a near-optimal $\tilde{\mathcal{O}}(n^2 \log\left(\frac{1}{\epsilon}\right))$ computational complexity to reach an ϵ-approximate full KRR solution

(Corollary 2.18). This convergence rate is comparable to that of PCG while improving over EigenPro 2.0 (Table 2.2).

2.1.3 Roadmap

Section 2.2 introduces `ASkotch` (and its non-accelerated variant, `Skotch`), and describes the techniques (sketch-and-project, Nyström approximation, automatic computation of stepsizes) that are used in this algorithm. Section 2.3 reviews existing methods to solve full and inducing points KRR and places `ASkotch` in the context of these works. Section 2.4 introduces several key quantities used in the convergence analysis of `ASkotch`. Section 2.5 develops lower bounds on the quantities introduced in Section 2.4 by exploiting connections between ARLS sampling and DPPs. Section 2.6 uses the lower bounds from Section 2.5 to establish a fine-grained convergence rate for `ASkotch` on full KRR. Section 2.7 demonstrates the superior performance of `ASkotch` over state-of-the-art full and inducing-points KRR solvers.

2.1.4 Notation

We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. Throughout the paper, $\mathcal{B}$ is a subset of $[n]$ that is sampled randomly according to some distribution. $I_{\mathcal{B}}$ concatenates the rows of I_n indexed by $\mathcal{B}$. Given a square matrix A, $A_{\mathcal{B}\mathcal{B}}$ denotes the principal submatrix of A indexed by $\mathcal{B}$.

$\mathrm{Tr}(\cdot)$ is the trace of a matrix, $\det(\cdot)$ is the determinant of a matrix, and $\mathrm{null}(\cdot)$ is the null space of a matrix. $\mathbb{S}^n_{++}$ ($\mathbb{S}^n_+$) denotes the convex cone of pd (psd) matrices in $\mathbb{R}^{n\times n}$. The symbol $\preceq$ denotes the Loewner order on the convex cone of psd matrices: $A \preceq B$ means $B - A$ is psd. For a square matrix A and constant λ, we define $A_\lambda \coloneqq A + \lambda I$. We use $\|\cdot\|$ to denote norms; $\|\cdot\|$ with no subscript is the Euclidean norm. For a matrix $A \in \mathbb{S}^n_+$, its eigenvalues in decreasing order are $\lambda_1(A) \geq \lambda_2(A) \geq \ldots \geq \lambda_n(A)$. A^+ is the Moore-Penrose pseudoinverse of a matrix A. If $A \in \mathbb{S}^n_+$, the smallest non-zero eigenvalue is denoted by $\lambda^+_{\min}(A)$. For a positive integer b and matrix A, $\lfloor A \rfloor_b$ is the best rank-b approximation of A with respect to the spectral norm.

The symbols $\Omega(\cdot)$, $\Theta(\cdot)$, and $\mathcal{O}(\cdot)$ are the standard quantities from asymptotic complexity analysis. $\tilde{\Omega}(\cdot)$, $\tilde{\Theta}(\cdot)$, and $\tilde{\mathcal{O}}(\cdot)$ are used to hide poly-log factors.

2.2 Algorithms

We propose `ASkotch` (**A**ccelerated **s**calable **k**ernel **o**ptimization and **t**raining with **c**oordinate sketch-and-project and approximate **H**essians, Algorithm 1) to solve full KRR. We introduce the `ASkotch` algorithm in Section 2.2.1, and we follow this by describing the key building blocks of the method, such as approximate sketch-and-project (Section 2.2.2), randomized Nyström approximation (Section 2.2.3), and automatic computation of the stepsize (Section 2.2.4). `ASkotch` comes with default

hyperparameter settings (Section 2.2.5), which reduces the effort required by practitioners to tune the method; we use these defaults in our experiments (Section 2.7).

2.2.1 ASkotch

Algorithm 1 ASkotch

Require: blocksize b, coordinate sampling distribution $\mathcal{P}$, acceleration parameters $\hat{\mu}, \hat{\nu}$, acceleration flag use_accel, rank r, damping ρ, kernel oracle K, targets y, ridge parameter λ, number of iterations N, initialization w_0

 # Compute acceleration parameters
$\beta \leftarrow 1 - \sqrt{\hat{\mu}/\hat{\nu}}$
$\gamma \leftarrow 1/\sqrt{\hat{\mu}\hat{\nu}}$
$\alpha \leftarrow 1/(1+\gamma\hat{\nu})$
$v_0 \leftarrow w_0, z_0 \leftarrow w_0$

for $i = 0, 1, \ldots, N-1$ **do**
  Sample a block of coordinates $\mathcal{B} \subseteq [n]$ according to $\mathcal{P}$ ▷ $|\mathcal{B}| = b$
  $\hat{K}_{\mathcal{B}\mathcal{B}} \leftarrow \texttt{Nyström}(K_{\mathcal{B}\mathcal{B}}, r)$ ▷ Low-rank approximation of $K_{\mathcal{B}\mathcal{B}}$
  $L_{P_\mathcal{B}} \leftarrow \texttt{get\_L}(K_{\mathcal{B}\mathcal{B}} + \lambda I, \hat{K}_{\mathcal{B}\mathcal{B}}, \rho)$ ▷ Via powering

  # Update iterates
  if use_accel **then**
    $d_i \leftarrow I_\mathcal{B}^T(\hat{K}_{\mathcal{B}\mathcal{B}} + \rho I)^{-1}((K_\lambda)_{\mathcal{B}:} z_i - y_\mathcal{B})$ ▷ Approximate projection; costs $\mathcal{O}(nb + rb)$
    $w_{i+1} \leftarrow z_i - (L_{P_\mathcal{B}})^{-1} d_i$ ▷ Costs $\mathcal{O}(b)$
    $v_{i+1} \leftarrow \beta v_i + (1-\beta) z_i - \gamma (L_{P_\mathcal{B}})^{-1} d_i$ ▷ Costs $\mathcal{O}(n)$
    $z_{i+1} \leftarrow \alpha v_{i+1} + (1-\alpha) w_{i+1}$ ▷ Costs $\mathcal{O}(n)$
  else
    # Non-accelerated version (Skotch)
    $d_i \leftarrow I_\mathcal{B}^T(\hat{K}_{\mathcal{B}\mathcal{B}} + \rho I)^{-1}((K_\lambda)_{\mathcal{B}:} w_i - y_\mathcal{B})$ ▷ Approximate projection; costs $\mathcal{O}(nb + rb)$
    $w_{i+1} \leftarrow w_i - (L_{P_\mathcal{B}})^{-1} d_i$ ▷ Costs $\mathcal{O}(b)$
  end if
end for
return w_N ▷ Approximate solution to $K_\lambda w = y$

At each iteration, ASkotch randomly samples a block $\mathcal{B}$ containing b distinct coordinates (we omit the dependence of $\mathcal{B}$ on i to avoid notational clutter). ASkotch then computes a rank-r Nyström approximation of $K_{\mathcal{B}\mathcal{B}}$ and computes a *preconditioned smoothness constant* for the block, $L_{P_\mathcal{B}}$, which is used to set the stepsize. Next, ASkotch computes the approximate projection step d_i. Finally, ASkotch combines the approximate projection step d_i with Nesterov acceleration [Nesterov, 2018] to update its estimate of the solution.

ASkotch is compatible with any coordinate sampling distribution $\mathcal{P}$. Our implementation uses two approaches inspired by popular sampling distributions in the kernel literature: uniform sampling and ARLS sampling. The uniform sampling distribution places an equal weight of $1/n$ on each coordinate, while ARLS sampling weights coordinates with probabilities proportional to their approximate RLSs (Definition 2.4). In our implementation, we approximate the RLSs using BLESS [Rudi et al., 2018, Gautier et al., 2019].

ASkotch has a flag indicating whether or not to use Nesterov acceleration: when this flag is False, acceleration is disabled. Throughout the rest of the paper, we will use the name Skotch to refer to the non-accelerated version of ASkotch.

2.2.2 Approximate Sketch-and-Project

Sketch-and-project (SAP) [Gower and Richtárik, 2015, Richtárik and Takáč, 2020] is a randomized, iterative framework to solve consistent linear systems. SAP for the full KRR linear system (2.3) takes in two parameters: (i) a fixed, pd matrix $Q \in \mathbb{S}^n_{++}$ and (ii) a distribution $\mathcal{D}$ over matrices in $\mathbb{R}^{b\times n}$, where b is a positive integer. At iteration i, SAP draws a sketching matrix $S_i \overset{\text{i.i.d.}}{\sim} \mathcal{D}$ and computes the next iterate by solving the optimization problem

$$\begin{aligned} w_{i+1} = \quad & \arg\min_{w} \quad \|w - w_i\|_Q^2 \\ & \text{subject to} \quad S_i K_\lambda w = S_i y. \end{aligned} \tag{2.6}$$

In other words, SAP progresses by projecting the current iterate w_i (with respect to the Q-norm) onto the solution space of a sketched version of (2.3). Importantly, when $Q = K_\lambda$ and $\mathcal{D}$ is the uniform distribution over row selection matrices of size $b \times n$, the SAP update has the following closed form:

$$w_{i+1} = w_i - Q^{-1}K_\lambda S_i^T (S_i K_\lambda Q^{-1} K_\lambda S_i^T)^+ (S_i K_\lambda w_i - S_i y) \tag{2.7}$$

$$= w_i - I_{\mathcal{B}}^T (K_{\mathcal{B}\mathcal{B}} + \lambda I)^{-1} ((K_\lambda)_{\mathcal{B}:} w_i - y_{\mathcal{B}}). \tag{2.8}$$

Nesterov acceleration has been applied to a wide range of iterative optimization methods to obtain algorithms with faster convergence rates [Allen-Zhu et al., 2016, Liu and Wright, 2016, Allen-Zhu, 2018, Ye et al., 2020]. SAP is no exception—Gower et al. [2018] combines SAP with Nesterov acceleration and shows that Nesterov-accelerated SAP (NSAP) converges at least as fast as its non-accelerated counterpart.

In theory, NSAP improves over PCG when $b = o(n^{2/3})$. However, a larger blocksize b is often beneficial for SAP methods—for example, the convergence rate for the randomized Newton method enjoys a superlinear speedup as b increases [Gower and Richtárik, 2015]. Using a direct method to solve the linear system in (2.7) requires $\mathcal{O}(b^3)$ computation, which becomes slow for $b \gtrsim 10^4$. Dereziński et al. [2025a], Dereziński and Yang [2024] propose approximating the solution of this linear system propose inexactly solving this linear system to a tolerance ϵ using PCG with a preconditioner P. Doing so requires $\mathcal{O}(b^2\sqrt{\kappa_P}\log(1/\epsilon))$ computation, where κ_P is the preconditioned condition number. However, for $b \gtrsim 10^4$, repeatedly applying PCG may be too slow to be practical.

`ASkotch` also approximates the SAP update, but it does so in a way that allows for a blocksize $b \gtrsim 10^4$. The matrix $K_{\mathcal{B}\mathcal{B}} + \lambda I$ in the SAP update (2.8) is replaced by a regularized rank-r Nyström approximation (Section 2.2.3), $\hat{K}_{\mathcal{B}\mathcal{B}} + \rho I$, where $\rho > 0$. This Nyström approximation can be computed rapidly, and the resulting linear system can be solved in $\mathcal{O}(br)$ time.

The theoretical analysis of SAP uses the fact that $Q^{-1}K_\lambda S_i^T (S_i K_\lambda Q^{-1} K_\lambda S_i^T)^+ S_i K_\lambda$ in (2.7) is a projection with respect to the Q-inner product [Gower and Richtárik, 2015]. However, by replacing

$S_i K_\lambda Q^{-1} K_\lambda S_i^T$ by a low-rank approximation, `ASkotch` loses this projection property. Consequently, `ASkotch` requires a (dynamic) stepsize to converge, unlike SAP, which is guaranteed to converge with a constant stepsize of 1. We provide a principled, automated method for computing this stepsize in Section 2.2.4.

2.2.3 Randomized Nyström Approximation

The Nyström approximation [Williams and Seeger, 2000, Bach, 2013, Alaoui and Mahoney, 2015, Tropp et al., 2017] is a well-known technique for producing low-rank approximations to psd matrices. Since kernel matrices have fast spectral decay (i.e., they have approximate low-rank structure) [Caponnetto and DeVito, 2007, Bach, 2013, Tu et al., 2016, Ma and Belkin, 2017, Belkin, 2018] we replace $K_{\mathcal{BB}}$ in (2.8) with a Nyström approximation, which is easier to invert.

Given a symmetric psd matrix $M \in \mathbb{S}_+^p$, the randomized Nyström approximation of M with respect to a random test matrix $\Omega \in \mathbb{R}^{p\times r}$ is

$$\hat{M} = (M\Omega)\left(\Omega^T M\Omega\right)^+ (M\Omega)^T.$$

Common choices for Ω include standard Gaussian random matrices, randomized Hadamard transforms, and sparse sign embeddings [Tropp et al., 2017]. The latter two test matrices reduce the computational cost of the sketch $M\Omega$. Our theoretical analysis uses sparse sign embeddings due to their computational efficiency, but our implementation uses Gaussian random matrices for simplicity. In the future, we will extend our implementation to use randomized Hadamard transforms and sparse sign embeddings.

Our practical implementation, `Nyström` (Algorithm 3), follows Tropp et al. [2017, Algorithm 3]. `Nyström` takes in a psd matrix $M \in \mathbb{S}_+^p$ and rank r and outputs a low-rank approximation $\hat{M} = \hat{U}\,\mathrm{diag}(\hat{\Lambda})\hat{U}^T$ to M in $\mathcal{O}(p^2r + pr^2)$ time[1], where $\hat{U} \in \mathbb{R}^{p\times r}$ is an orthogonal matrix containing approximate top-r eigenvectors of M and $\hat{\Lambda} \in \mathbb{R}^r$ is a vector containing approximate top-r eigenvalues of M. Note that `Nyström` never forms $\hat{M}$ as a matrix; rather, it returns the factors $\hat{U}$ and $\hat{\Lambda}$. For more details, please see Appendix A.1.1.

At each iteration, `ASkotch` computes a randomized Nyström approximation of the block kernel $K_{\mathcal{BB}}$ in the SAP update (2.8). This approximation is always used in conjunction with a damping $\rho > 0$ to ensure positive definiteness. For any vector $v \in \mathbb{R}^p$, $(\hat{M} + \rho I)^{-1}v$ can be computed in $\mathcal{O}(pr)$ time using the Woodbury formula, which lets `ASkotch` cheaply compute the approximate projection step d_i.

[1] For sparse sign embeddings, this complexity is reduced to $\tilde{\mathcal{O}}(pr + pr^2)$.

2.2.4 Automatic Computation of the Stepsize

The stepsize is often challenging to set in iterative optimization methods: if the stepsize is too large, `ASkotch` will diverge, while if the stepsize is too small, `ASkotch` will make little progress towards the solution. In practice, tuning the stepsize can require an expensive hyperparameter search, which is inconvenient for practitioners. We present a practical approach for automatically computing the stepsize in `ASkotch`.

Our idea is to use a preconditioned smoothness constant to set the stepsize in `ASkotch`. The preconditioned smoothness constant in `ASkotch` is defined as

$$L_{P_\mathcal{B}} \coloneqq \lambda_1 \left(\left(\hat{K}_{\mathcal{B}\mathcal{B}} + \rho I \right)^{-1/2} \left(K_{\mathcal{B}\mathcal{B}} + \lambda I \right) \left(\hat{K}_{\mathcal{B}\mathcal{B}} + \rho I \right)^{-1/2} \right).$$

`ASkotch` uses randomized powering [Kuczyński and Woźniakowski, 1992, Martinsson and Tropp, 2020] to compute $L_{P_\mathcal{B}}$. The total cost of the procedure is $\tilde{\mathcal{O}}(b^2)$. In practice, `ASkotch` uses just 10 iterations of powering. Hence, the cost of computing $L_{P_\mathcal{B}}$ is dominated by the $\mathcal{O}(nb)$ cost of computing the search direction d_i in `ASkotch`. `ASkotch` computes $L_{P_\mathcal{B}}$ using the subroutine `get_L` (Algorithm 4).

In Sections 2.4 and 2.6, we show that our approach for setting the stepsize guarantees that `Skotch` and `ASkotch` obtain linear convergence.

2.2.5 Default Hyperparameters

We provide default recommendations for setting hyperparameters in `ASkotch`. These recommendations are summarized in Table 2.3.

- Blocksize b and rank r: Choose as large as hardware allows. A good, general default is $b = n/100$ and $r = 100$.
- Damping ρ: Set adaptively for each block $\mathcal{B}$; we use $\rho = \lambda + \lambda_r(\hat{K}_{\mathcal{B}\mathcal{B}})$.
- Sampling distribution $\mathcal{P}$: Use the uniform sampling distribution. While we use ARLS sampling to prove fast convergence rates for `ASkotch`, it (i) performs similarly to uniform sampling in our ablations (Section 2.7) and (ii) computing approximate RLSs via BLESS costs $\tilde{\mathcal{O}}(nb^2)$.
- Acceleration parameters $\hat{\mu}$ and $\hat{\nu}$: Set $\hat{\mu} = \lambda$ and $\hat{\lambda} = n/b$. These choices are guided by the convergence analysis of `ASkotch`. However, the user must ensure that $\hat{\mu} \leq \hat{\nu}$ and $\hat{\mu}\hat{\nu} \leq 1$. We believe it is possible to make the acceleration in `ASkotch` parameter-free, as done in Dereziński et al. [2025b], but we leave this extension to future work.

Table 2.3: Default hyperparameters for `ASkotch`.

Hyperparameter	Default recommendation
b	$n/100$
$\mathcal{P}$	Uniform
$\hat{\mu}$	λ
$\hat{\nu}$	n/b
r	100
ρ	$\lambda + \lambda_r(\hat{K}_{\mathcal{B}\mathcal{B}})$

2.3 Related Work

We review existing solvers for full and inducing points KRR and discuss how our methods compare to the literature.

2.3.1 Full KRR

Many prior works have developed methods to solve full KRR that avoid the $\mathcal{O}(n^3)$ cost of direct methods such as Cholesky decomposition. The various approaches can be roughly divided into 3 categories: (i) PCG methods, (ii) gradient-based methods, and (iii) SAP methods.

For PCG, much work has been done on developing efficient preconditioners [Cutajar et al., 2016, Avron et al., 2017, Frangella et al., 2023, Díaz et al., 2023]. Among the numerous proposals, the most popular preconditioners are based on the Nyström approximation [Nyström, 1930]. Nyström approximations can be constructed from the kernel matrix via uniform sampling of columns [Williams and Seeger, 2000], greedy pivoting [Harbrecht et al., 2012], leverage-score sampling [Musco and Musco, 2017], random projection [Tropp et al., 2017], and randomized pivoting [Chen et al., 2025a, Epperly et al., 2025]. Preconditioners have also been constructed using random-features approximations [Avron et al., 2017], but these do not perform as well in practice as Nyström preconditioners [Díaz et al., 2023, Frangella et al., 2023]. Typically, if a Nyström preconditioner is constructed with a rank $r = \mathcal{O}(d^{\lambda}(K))$, then PCG converges at a condition-number-free rate with high probability [Zhao et al., 2022, Frangella et al., 2023]. The downside of these PCG approaches is that they exhibit an $\mathcal{O}(n^2)$ per-iteration cost and require $\mathcal{O}(nr)$ storage, making it difficult to scale these methods beyond $n \gtrsim 10^5$.

EigenPro and EigenPro 2.0 [Ma and Belkin, 2017, 2019] are the most well-known stochastic gradient methods for solving full KRR. Both EigenPro and EigenPro 2.0 set the regularization $\lambda = 0$ and solve the resulting KRR problem via preconditioned stochastic gradient descent. EigenPro 2.0 reduces the per-iteration cost to $\mathcal{O}(nb_g + rs)$, a significant improvement over the $\mathcal{O}(n^2)$ cost of PCG. Iterative sketching methods [Pilanci and Wainwright, 2016, Lacotte et al., 2019, Lacotte and Pilanci, 2020, 2022] could also be applied to solve full KRR, but doing so would require the square root of

the kernel matrix, which costs $\mathcal{O}(n^3)$ time to compute.

The third approach to solve full KRR problems is based on SAP. Tu et al. [2016] proposes randomized block Gauss-Seidel for full KRR. In follow-up work, Tu et al. [2017] develops NSAP for solving the full KRR problem. Gazagnadou et al. [2022] applies SAP for solving full KRR but uses count sketches instead of coordinate sampling to form $SK_\lambda S^T$ in the SAP update. Unfortunately, these methods have $\mathcal{O}(b^3)$ computational cost per iteration, which is too expensive for our largest example, taxi, where $b = 5 \cdot 10^4$. More recently, Lin et al. [2024] proposes a variant of NSAP where $K_{\mathcal{BB}} + \lambda I$ is replaced by the identity matrix. While this replacement removes the $\mathcal{O}(b^3)$ iteration cost associated with matrix factorization, its convergence becomes much slower.

Among these (approximate) SAP methods, `Skotch` and `ASkotch` are the only methods that guarantee condition-number-free convergence under reasonable assumptions on the kernel matrix, while also avoiding $\mathcal{O}(b^3)$ per-iteration costs.

2.3.2 Inducing Points KRR

PCG is also popular for solving inducing points KRR, with Falkon [Rudi et al., 2017, Meanti et al., 2020] and KRILL [Díaz et al., 2023] being the current state-of-the-art. KRILL is similar to Falkon, but solves (2.5) via PCG with a preconditioner constructed using a sparse sketch. KRILL has robust theoretical guarantees, and numerical tests in Díaz et al. [2023] show it yields comparable or better performance than Falkon. Nevertheless, KRILL still has a $\tilde{\mathcal{O}}(nm+m^3)$ runtime and requires $\mathcal{O}(m^2)$ storage. Consequently, it encounters the same runtime and storage barriers as Falkon for large m.

Meanti et al. [2020] develops a software package for Falkon that relies on extensive code and hardware optimizations. These optimizations include stabilizing Falkon in single precision, out-of-core matrix operations, parallelized memory transfers, and distribution over multiple GPUs. Consequently, Meanti et al. [2020] scales Falkon to $n = 10^9$ with $m = 10^5$. Despite these optimizations, Falkon still encounters fundamental memory barriers: Abedsoltan et al. [2023] shows that the optimized version of Falkon cannot scale beyond 2.56×10^5 inducing points when given 340 GB of RAM, which is larger than the memory budget available to most practitioners.

Yang et al. [2017] generalizes the inducing points estimator to a broader class of sketching-based estimators. However, these sketching-based estimators require matrix-matrix products with the $n \times n$ kernel matrix, incurring $\tilde{\Omega}(n^2)$ runtime, and therefore are not scalable to the same extent as Falkon and KRILL.

Abedsoltan et al. [2023] develops the stochastic gradient method EigenPro 3.0 for inducing points KRR. EigenPro 3.0 has a lower iteration and storage complexity than Falkon, which allows it to use more inducing points. Unfortunately, this method has no convergence guarantees.

2.4 Key Quantities in the Analysis of ASkotch

Establishing convergence of ASkotch involves quantities whose relevance is not obvious a priori. Therefore, we show how the key quantities in the convergence analysis arise. For simplicity, our analysis focuses on ASkotch without acceleration (i.e., Skotch). Throughout the analysis, we omit the dependence of $\mathcal{B}$ on i to avoid notational clutter.

2.4.1 SAP and Nyström Projectors

We begin by introducing two matrices that will play a central role in the convergence analysis of ASkotch.

Definition 2.1 (SAP projector)**.** For a block $\mathcal{B}$, kernel matrix K, and regularization λ, the SAP projector is

$$\Pi_{\mathcal{B}} \coloneqq K_\lambda^{1/2} I_{\mathcal{B}}^T (K_{\mathcal{B}\mathcal{B}} + \lambda I)^{-1} I_{\mathcal{B}} K_\lambda^{1/2}.$$

The SAP projector $\Pi_{\mathcal{B}}$ is an *exact* projection matrix, which controls the convergence rate of SAP methods.

Definition 2.2 (Nyström projector)**.** For a block $\mathcal{B}$, kernel matrix K, regularization λ, damping $\rho > 0$, and stepsize $\eta_{\mathcal{B}} > 0$, the Nyström projector is

$$\hat{\Pi}_{\mathcal{B},\rho} \coloneqq \eta_{\mathcal{B}} K_\lambda^{1/2} I_{\mathcal{B}}^T (\hat{K}_{\mathcal{B}\mathcal{B}} + \rho I)^{-1} I_{\mathcal{B}} K_\lambda^{1/2}.$$

The Nyström projector controls the convergence rate of ASkotch. Note that the name Nyström projector is a slight abuse of terminology, as $\hat{\Pi}_{\mathcal{B},\rho}$ is not idempotent. However, for appropriately chosen $\eta_{\mathcal{B}}$, we will show that $\hat{\Pi}_{\mathcal{B},\rho}$ is closely related to the SAP projector.

2.4.2 One-Step Analysis of Skotch and SAP

We now show how the SAP and Nyström projectors arise by performing a one-step analysis of SAP and Skotch.

We begin by writing the updates for SAP and Skotch:

$$w_i = w_{i-1} - I_{\mathcal{B}}^T \left(K_{\mathcal{B}\mathcal{B}} + \lambda I\right)^{-1} I_{\mathcal{B}} (K_\lambda w_i - y), \tag{SAP}$$

$$w_i = w_{i-1} - \eta_{\mathcal{B}} I_{\mathcal{B}}^T \left(\hat{K}_{\mathcal{B}\mathcal{B}} + \rho I\right)^{-1} I_{\mathcal{B}} (K_\lambda w_i - y). \tag{Skotch}$$

Some elementary manipulations show that

$$\|w_i - w_\star\|_{K_\lambda}^2 \leq (w_{i-1} - w_\star)^T K_\lambda^{1/2} \left(I - \Pi_{\mathcal{B}}\right)^2 K_\lambda^{1/2} (w_{i-1} - w_\star), \tag{SAP}$$

$$\|w_i - w_\star\|_{K_\lambda}^2 \leq (w_{i-1} - w_\star)^T K_\lambda^{1/2} \left(I - \hat{\Pi}_{\mathcal{B},\rho}\right)^2 K_\lambda^{1/2} (w_{i-1} - w_\star). \tag{Skotch}$$

As $\Pi_{\mathcal{B}}$ is a projection, we can further deduce for SAP that

$$\|w_i - w_\star\|_{K_\lambda}^2 \leq (w_{i-1} - w_\star)^T K_\lambda^{1/2} \left(I - \Pi_{\mathcal{B}}\right) K_\lambda^{1/2} (w_{i-1} - w_\star). \tag{SAP}$$

Define[2]

$$\mu \coloneqq \lambda_{\min}(\mathbb{E}\left[\Pi_{\mathcal{B}}\right]).$$

Then, taking the expectation conditioned on w_{i-1} yields

$$\mathbb{E}[\|w_i - w_\star\|_{K_\lambda}^2 \mid w_{i-1}] \leq (1 - \mu) \|w_{i-1} - w_\star\|_{K_\lambda}^2. \tag{SAP}$$

Thus, the convergence of SAP is controlled by the smallest positive eigenvalue of the expected projection matrix $\mathbb{E}\left[\Pi_{\mathcal{B}}\right]$. As `Skotch` is an approximate version of SAP, we would like to deduce a similar result. Indeed, if we can ensure

$$\hat{\Pi}_{\mathcal{B},\rho} \preceq I,$$

then

$$\|w_i - w_\star\|_{K_\lambda}^2 \leq (w_{i-1} - w_\star)^T K_\lambda^{1/2} \left(I - \hat{\Pi}_{\mathcal{B},\rho}\right) K_\lambda^{1/2} (w_{i-1} - w_\star). \tag{Skotch}$$

Defining

$$\hat{\mu} \coloneqq \lambda_{\min}(\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]),$$

it immediately follows from the preceding display that

$$\mathbb{E}[\|w_i - w_\star\|_{K_\lambda}^2 \mid w_{i-1}] \leq (1 - \hat{\mu}) \|w_{i-1} - w_\star\|_{K_\lambda}^2. \tag{Skotch}$$

Setting the Stepsize η_B

The convergence of `Skotch` is controlled by $\hat{\mu}$, the smallest positive eigenvalue of $\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]$, similar to how μ controls the convergence of SAP. To conclude this relation, we require

$$\hat{\Pi}_{\mathcal{B},\rho} \preceq I.$$

This condition tells us that the stepsize $\eta_{\mathcal{B}}$ cannot be arbitrarily large. However, we also want to characterize the behavior of $\hat{\mu}$: when $\hat{K}_{\mathcal{B}\mathcal{B}}$ is a good approximation to $K_{\mathcal{B}\mathcal{B}}$, $\hat{\mu}$ should be closely related to μ. The following lemma formalizes this intuition.

[2] If K_λ was only psd rather than pd, then this quantity must be replaced by $\lambda_{\min}^+(\mathbb{E}[\Pi_{\mathcal{B}}])$, as $\mathbb{E}[\Pi_{\mathcal{B}}]$ would be singular. For our setting, this does not matter as K_λ is pd.

Lemma 2.3. *Let $\rho > 0$ and $\mathcal{B} \subseteq [n]$ be fixed. Let $\hat{K}_{\mathcal{B}\mathcal{B}}$ be a Nyström approximation of $K_{\mathcal{B}\mathcal{B}}$. Define*

$$
\begin{aligned}
L_{P_\mathcal{B}} &\coloneqq \lambda_{\max}\left((K_{\mathcal{B}\mathcal{B}} + \lambda I)^{1/2}\left(\hat{K}_{\mathcal{B}\mathcal{B}} + \rho I\right)^{-1}(K_{\mathcal{B}\mathcal{B}} + \lambda I)^{1/2}\right), \\
\hat{L}_{P_\mathcal{B}} &\coloneqq \max\{1, L_{P_\mathcal{B}}\}, \\
\sigma_{P_\mathcal{B}} &\coloneqq \lambda_{\min}\left((K_{\mathcal{B}\mathcal{B}} + \lambda I)^{1/2}\left(\hat{K}_{\mathcal{B}\mathcal{B}} + \rho I\right)^{-1}(K_{\mathcal{B}\mathcal{B}} + \lambda I)^{1/2}\right).
\end{aligned}
$$

If we set $\eta_B = 1/\hat{L}_{P_\mathcal{B}}$ in Definition 2.2, then the Nyström projector satisfies

$$
\frac{\sigma_{P_\mathcal{B}}}{\hat{L}_{P_\mathcal{B}}}\Pi_\mathcal{B} \preceq \hat{\Pi}_{\mathcal{B},\rho} \preceq \Pi_\mathcal{B}. \tag{2.9}
$$

Setting the `ASkotch` stepsize to $\eta_\mathcal{B} = 1/\hat{L}_{P_\mathcal{B}}$ ensures convergence and allows for a lower bound on $\lambda_{\min}(\hat{\Pi}_{\mathcal{B},\rho})$ in terms of $\lambda_{\min}(\Pi_\mathcal{B})$. Recall that we set $\eta_\mathcal{B} = 1/L_{P_\mathcal{B}}$ as the stepsize for `ASkotch` in Section 2.2. The conclusions of Lemma 2.3 still hold for this simpler stepsize. We only require $1/\hat{L}_{P_\mathcal{B}}$ for our theoretical analysis to show `ASkotch` achieves a better upper bound on the number of iterations to reach an ϵ-approximate solution of (2.3) than `Skotch`.

2.5 First-Moment Analysis of the Expected Nyström Projector

The argument in Section 2.4.2 shows that the convergence of `ASkotch` is controlled by $\hat{\mu}$, which is the smallest eigenvalue of the first moment of the Nyström projector, $\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]$. Analyzing $\hat{\mu}$ and $\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]$ directly seems impenetrable. Instead, we would like to derive a lower bound on $\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]$ in terms of the first moment of the SAP projector, $\mathbb{E}[\Pi_\mathcal{B}]$:

$$
\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}] \succeq g(\rho,\lambda)\mathbb{E}[\Pi_\mathcal{B}], \tag{2.10}
$$

where $g(\rho,\lambda)$ is a scalar function that captures the price of replacing $K_{\mathcal{B}\mathcal{B}}$ with $\hat{K}_{\mathcal{B}\mathcal{B}}$. Once (2.10) has been established, we immediately obtain

$$
\hat{\mu} \geq g(\rho,\lambda)\mu. \tag{2.11}
$$

Provided that we have a non-trivial lower bound on μ, we can combine (2.11) with the one-step analysis in Section 2.4.2 to obtain an explicit convergence rate for `Skotch`. We can obtain an explicit convergence rate for `ASkotch` using a similar approach, with some additional work to account for Nesterov acceleration.

There are two challenges in establishing the lower bound on $\hat{\mu}$ in (2.11):

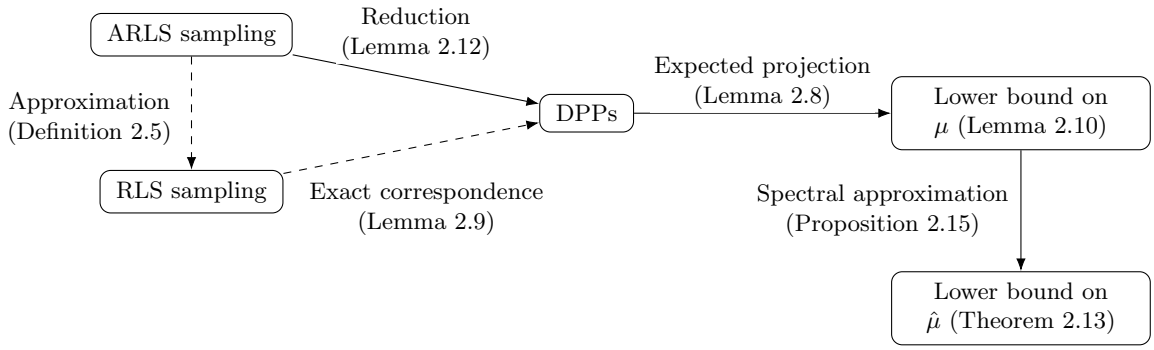


Figure 2.2: Summary of reductions and correspondences between ARLS/RLS sampling, DPPs, and the resulting lower bounds on μ and $\hat{\mu}$. The dashed arrows are for intuition only: ARLS is closely related to RLS and RLS is related to DPPs, so we expect ARLS to be related to DPPs. The solid arrows correspond to our approach for obtaining a lower bound on $\hat{\mu}$.

1. **Deriving a meaningful lower bound on μ is non-trivial**. Existing works that establish such a lower bound rely on either expensive DPP sampling techniques [Mutny et al., 2020] or preprocessing the kernel matrix using a Hadamard transform [Dereziński and Yang, 2024, Dereziński et al., 2025a], which incurs a $\mathcal{O}(n^2)$ cost and requires materializing a $n \times n$ matrix in memory. Therefore, we must develop a different approach for establishing a lower bound on μ.

2. **Identifying the form of $g(\rho, \lambda)$ requires care**. The analysis to determine $g(\rho, \lambda)$ requires careful spectral approximation arguments based on the theory of the Nyström approximation.

In the following subsections, we address these challenges. In Section 2.5.1, we provide background on (approximate) ridge leverage scores and DPPs. In Section 2.5.2, we develop a novel ARLS-to-DPP reduction (Lemma 2.12) to establish a non-trivial lower bound on μ (Lemma 2.10). This reduction and its associated results can be applied to any psd matrix, making them of independent interest outside of this manuscript. In Section 2.5.3, we relate $\hat{\Pi}_{\mathcal{B},\rho}$ to $\Pi_{\mathcal{B}}$ using spectral approximation arguments based on the Nyström approximation. Combining these ideas, we show the following lower bound on $\hat{\mu}$:

$$\hat{\mu} \geq \frac{\lambda}{16\bar{\kappa}_{k:n}\rho}\frac{k}{n},$$

where $\bar{\kappa}_{k:n} \coloneqq \frac{1}{n-k}\sum_{i>k}\lambda_i(K_\lambda)/\lambda_{\min}(K_\lambda)$. The exact statement is presented in Theorem 2.13.

For clarity, we illustrate our argument to lower bound $\hat{\mu}$ in Fig. 2.2.

2.5.1 Approximate Ridge Leverage Score Sampling and DPPs

In this subsection, we formally introduce approximate ridge leverage scores and ARLS sampling and discuss their efficient computation. As our argument relies on a reduction from ARLS sampling to DPP sampling, we also define determinantal point processes and introduce some fundamental facts about them.

Approximate Ridge Leverage Score Sampling

We begin by introducing ridge leverage scores.

Definition 2.4 (Ridge leverage scores, effective dimension, and degrees of freedom)**.** Given $A \in \mathbb{S}^n_+$ and $\lambda > 0$, its i-th λ-ridge leverage score $\ell_i^\lambda(A)$ is the i-th diagonal entry of matrix $A(A+\lambda I)^{-1}$:

$$\ell_i^\lambda(A) = e_i^T A(A+\lambda I)^{-1} e_i.$$

The sum of all its λ-ridge leverage scores is called its λ-effective dimension:

$$d^\lambda(A) \coloneqq \operatorname{Tr}\left(A(A+\lambda I)^{-1}\right) = \sum_{i=1}^n \ell_i^\lambda(A).$$

The λ-maximal degrees of freedom of A is given by:

$$d^\lambda_{\max}(A) \coloneqq n \max_{i\in[n]} \ell_i^\lambda(A).$$

The effective dimension measures the degrees of freedom of A, taking into account the regularization λ. For matrices with decaying eigenvalues (e.g., kernel matrices), $d^\lambda(A)$ is much smaller than the ambient dimension n [Alaoui and Mahoney, 2015]. The i-th RLS captures how much a row i of A contributes to the effective dimension. The maximal degrees of freedom is always larger than the effective dimension, and can be significantly larger when the leverage score distribution is highly non-uniform.

The RLS scores naturally define a distribution on the rows of A: sample row i with probability $p_i = \ell_i^\lambda(A)/d^\lambda(A)$. This sampling distribution admits strong theoretical guarantees [Alaoui and Mahoney, 2015, Rudi et al., 2018], but is impractical as computing the leverage scores requires an eigendecomposition at a cost of $\mathcal{O}(n^3)$, which is more expensive than the problem we wish to solve. Fortunately, good guarantees can still be obtained by using approximate ridge leverage score sampling (ARLS sampling), which we now define.

Definition 2.5 (Approximate RLSs and ARLS^λ_c-sampling)**.** Given $A \in \mathbb{S}^n_+$ and $\lambda > 0$, we say that

$\{\tilde{\ell}_i\}_{i=1}^n$ are a c-approximation of λ-ridge leverage scores $\{\ell_i^\lambda(A)\}_{i=1}^n$ for some $c \geq 1$, if they satisfy

$$\tilde{\ell}_i \geq \ell_i^\lambda(A) \quad \forall i \qquad \text{and} \qquad \tilde{\ell} \coloneqq \sum_{i=1}^n \tilde{\ell}_i \leq c \cdot d^\lambda(A).$$

Define $p_i := \frac{\tilde{\ell}}{n}\left\lceil \frac{n}{\tilde{\ell}}\tilde{\ell}_i \right\rceil$ for all $i \in [n]$, and sample indices $\{i_j\}_{j=1}^b$ i.i.d. according to the probabilities $\{p_i/\|p\|_1\}_{i=1}^n$. Then $\mathcal{B} = \bigcup_{j=1}^b \{i_j\}$ is an ARLS_c^λ-sampling.[3]

Approximate RLSs were first introduced by Alaoui and Mahoney [2015] for constructing randomized Nyström approximations of kernel matrices. The key advantage of ARLSs over RLSs is that they can be computed efficiently. In particular, `ASkotch` uses the BLESS algorithm proposed by Rudi et al. [2018] to compute the ARLSs. BLESS admits the following complexity guarantee:

Lemma 2.6 (Rudi et al. [2018], Theorem 1)**.** *Given $A \in \mathbb{S}_+^n$, a positive integer k, $c \geq 1$ and $\lambda > 0$ such that the λ-effective dimension of A satisfies $d^\lambda(A) \leq k$, there is an algorithm (BLESS) that with high probability returns c-approximations for all n λ-ridge leverage scores of A in $\tilde{\mathcal{O}}(nk^2)$ time.*

Lemma 2.6 shows we can obtain accurate λ-ARLSs to construct the sampling distribution in $\tilde{O}(nk^2)$ time. This stands in contrast to SAP solvers in Dereziński and Yang [2024], Dereziński et al. [2025a], which require $\mathcal{O}(n^2)$ time to preprocess K by a Hadamard transform H, and $\mathcal{O}(n^2)$ storage to store HKH^T, from which rows are subsampled. Thus, ARLSs yield a significant improvement in computational and storage complexities over Hadamard preprocessing.

Determinantal Point Processes and their Connection to RLSs

The heart of our analysis is a novel ARLS-to-DPP reduction argument that exploits a fundamental connection between RLS and determinantal point processes (DPPs). DPPs are a family of non-i.i.d. probability measures used in machine learning to sample diverse subsets of data points [Kulesza and Taskar, 2012].

Definition 2.7 (Determinantal point process)**.** Given $A \in \mathbb{S}_+^n$, a (random-size) determinantal point process $\mathrm{DPP}(A)$ is a distribution over all 2^n index subsets $\mathcal{B} \subseteq [n]$ such that $\Pr(\mathcal{B}) = \frac{\det(A_{\mathcal{B}\mathcal{B}})}{\det(A+I)}$. Moreover, a (fixed-size) k-$\mathrm{DPP}(A)$ is a distribution over subsets $\mathcal{B} \subseteq \binom{[n]}{k}$ such that $\Pr(\mathcal{B}) \propto \det(A_{\mathcal{B}\mathcal{B}})$.

DPP-based row selection matrices yield strong convergence guarantees for SAP methods thanks to the following formula connecting random projectors with DPP sampling.

Lemma 2.8 (Dereziński et al. [2020], Lemma 5)**.** *Given $A \in \mathbb{S}_+^n$, let $\mathcal{B} \sim \mathrm{DPP}(A)$, and define the random projection matrix $\Pi_\mathcal{B} := A^{1/2}S^T(SAS^T)^+SA^{1/2}$, where S is a row selection matrix for set $\mathcal{B}$. Then*

$$\mathbb{E}[\Pi_\mathcal{B}] = A(A+I)^{-1}. \tag{2.12}$$

[3]The indices in $\mathcal{B}$ are actually distinct since our analysis allows us to discard duplicates.

Mutny et al. [2020] gives a sharp convergence analysis for DPP-based SAP methods using (2.12). Unfortunately, directly sampling from a DPP is impractical due to its prohibitive computational cost [Dereziński et al., 2019].

The goal of our ARLS-to-DPP reduction is to show that when using ARLS sampling, a relation similar to (2.12) holds:

$$\mathbb{E}[\Pi_{\mathcal{B}}] \succeq A(A+I)^{-1} - \delta A^{+}A.$$

That is, we want to obtain a perturbed version of the DPP expectation formula (2.12) for a small δ. Our arguments show that this is indeed the case. The key idea is to exploit the following connection between the RLSs of A and DPP(A):

Lemma 2.9 (Dereziński and Mahoney [2021], Theorem 10)**.** *For $\mathcal{B} \sim \mathrm{DPP}(A)$ and index $i \in [n]$, the marginal probability of $i \in \mathcal{B}$ is the i-th ridge leverage score of A, i.e.,*

$$\Pr(i \in \mathcal{B}) = \left[A(A+I)^{-1}\right]_{i,i} = \ell_i^1(A).$$

2.5.2 Analysis of μ with ARLS Sampling

The main goal of this subsection is establish the following result, which provides tight control on $\mathbb{E}[\Pi_{\mathcal{B}}]$ and μ when using ARLS sampling.

Lemma 2.10 (Projection analysis for ARLS sampling)**.** *Given $A \in \mathbb{S}_+^n$ with rank r, $k = \Omega(\log n)$, let $\bar{\lambda}, \tilde{\lambda} > 0$ be such that $d^{\tilde{\lambda}}(A) \geq 2d^{\bar{\lambda}}(A) = 4k$. If $\mathcal{B}$ is $\mathrm{ARLS}_c^{\tilde{\lambda}}$-sampled with blocksize $b = \Omega(ck\log^3 n)$, then the projection $\Pi_{\mathcal{B}} \coloneqq A^{1/2}I_{\mathcal{B}}^T(I_{\mathcal{B}}AI_{\mathcal{B}}^T)^{+}I_{\mathcal{B}}A^{1/2}$ satisfies*

$$\mathbb{E}[\Pi_{\mathcal{B}}] \;\succeq\; \frac{1}{2}A\left(A+\bar{\lambda}I\right)^{-1}. \tag{2.13}$$

Furthermore, this implies the following lower bound for the SAP convergence parameter μ:

$$\mu = \lambda_{\min}^{+}(\mathbb{E}[\Pi_{\mathcal{B}}]) \geq \frac{k}{2r\bar{\kappa}_{k:r}}, \quad \textit{where} \quad \bar{\kappa}_{k:r} \coloneqq \frac{1}{r-k}\sum_{i>k}\lambda_i(A)/\lambda_{\min}^{+}(A).$$

Lemma 2.10 gives a result similar to Lemma 2.8, only instead of exact equality as in (2.12), we obtain a lower bound. Nevertheless, this is sufficient for our needs, and allows us to obtain a lower bound on μ comparable to what is obtained in Dereziński and Yang [2024], Dereziński et al. [2025a] using Hadamard preprocessing. Thus, using ARLS sampling to select rows of A yields the same control over $\mathbb{E}[\Pi_{\mathcal{B}}]$, but at much lower computational and storage costs compared to SAP methods that use Hadamard preprocessing.

Lemma 2.10 immediately yields a projection analysis for uniform sampling, since uniform sampling corresponds to ARLS sampling with a specific choice of scores.

Corollary 2.11 (Projection analysis for uniform sampling)**.** *Suppose A, $\bar{\lambda}$, $\tilde{\lambda}$, and k are set as in Lemma 2.10. If $\mathcal{B}$ is sampled according to the uniform distribution with blocksize $b = \Omega(d_{\max}^{\tilde{\lambda}}(A)\log^3 n)$, then $\Pi_{\mathcal{B}}$ satisfies*

$$\mathbb{E}[\Pi_{\mathcal{B}}] \;\succeq\; \frac{1}{2}A\left(A+\bar{\lambda}I\right)^{-1}. \tag{2.14}$$

Under uniform sampling, the blocksize has to be proportional to the maximal degrees of freedom: $b = \mathcal{O}(d_{\max}^{\tilde{\lambda}}(A)\log^3 n)$. When the ridge leverage scores are relatively uniform, i.e., $\max_{i\in[n]} \ell_i^{\tilde{\lambda}}(A) \approx \frac{d^{\tilde{\lambda}}(A)}{n}$, then $d_{\max}^{\tilde{\lambda}}(A) = \Theta(d^{\tilde{\lambda}}(A))$, so uniform sampling performs very well without requiring a large blocksize. As uniform sampling works well empirically across a broad range of learning tasks (Section 2.7), we believe this setting is most relevant for practice.

ARLS-to-DPP Reduction

To prove Lemma 2.10, we rely on connections between RLSs and DPPs. Recall from Lemma 2.8 that for any $A \in \mathbb{S}_+^n$ and $\bar{\lambda} > 0$, the sample $\mathcal{B}_{\text{DPP}} \sim \text{DPP}(A/\bar{\lambda})$ satisfies $\mathbb{E}[\Pi_{\mathcal{B}_{\text{DPP}}}] = A(A+\bar{\lambda}I)^{-1}$. Moreover, from Lemma 2.9, the marginal probability of sampling element i into $\mathcal{B}$ is the i-th $\bar{\lambda}$-ridge leverage score of A, i.e., $\Pr(i \in \mathcal{B}_{\text{DPP}}) = \ell_i^{\bar{\lambda}}(A)$. We would like to exploit this connection to transfer the projection analysis from DPPs to ARLSs. Indeed, the following lemma shows that a sufficiently large ARLS sample contains a DPP sample, which is crucial for the proof of Lemma 2.10:

Lemma 2.12 (ARLS-to-DPP reduction)**.** *Let $k, \tilde{\lambda}$, and $\mathcal{B}$ be defined as in Lemma 2.10. Given $\delta = n^{-\mathcal{O}(1)}$, for any $j \in \left[2k - \sqrt{6k\log\left(\frac{2}{\delta}\right)}, 2k + \sqrt{6k\log\left(\frac{2}{\delta}\right)}\right]$, we can couple a DPP sample $\mathcal{B}_{j\text{-DPP}} \sim j\text{-DPP}(A/\tilde{\lambda})$ with $\mathcal{B}$ such that with probability at least $1-\delta$, $\mathcal{B}_{j\text{-DPP}} \subseteq \mathcal{B}$.*

The proof of Lemma 2.12 is given in Appendix A.2.2. Our analysis is inspired by Dereziński and Yang [2024], who use a Markov Chain Monte Carlo (MCMC) argument [Anari et al., 2024] to show that after conditioning $\mathcal{B}_{\text{DPP}}$ on a fixed sample size j, the i.i.d. sampling distribution over the marginals $p_{i|j} = \Pr(i \in \mathcal{B}_{\text{DPP}} \mid |\mathcal{B}_{\text{DPP}}| = j)$ is an effective proxy for $j\text{-DPP}(A/\bar{\lambda})$. We modify the approach in Dereziński and Yang [2024] by showing that the approximate ridge leverage scores $\tilde{\ell}_i^{\bar{\lambda}}(A)$ are close to the marginals $p_{i|j}$ as long as the size j is close to the expected size of $\mathcal{B}_{\text{DPP}}$, $\mathbb{E}[|\mathcal{B}_{\text{DPP}}|]$. Fortunately, this occurs with high probability (Lemma A.2). We then use the law of total expectation to decompose the distribution of $\text{DPP}(A/\bar{\lambda})$ into a mixture of fixed-size DPP distributions $j\text{-DPP}(A)$, and apply an MCMC argument to each fixed-size DPP. Combining all of these ideas yields the result.

Proof of Lemma 2.10

To complete the proof of Lemma 2.10, we leverage the ARLS-to-DPP reduction from Lemma 2.12 and apply the DPP projection formula from Lemma 2.8.

Proof. Define the interval $\mathcal{I} = \left[2k - \sqrt{6k\log\left(\frac{2}{\delta}\right)}, 2k + \sqrt{6k\log\left(\frac{2}{\delta}\right)}\right]$ where $k = d^{\bar{\lambda}}(A)/2$, and denote event $\mathcal{E}_j = \{\mathcal{B}_{j\text{-DPP}} \subseteq \mathcal{B}\}$. According to Lemma 2.12, for any $j \in \mathcal{I}$ we have $\Pr(\mathcal{E}_j) \geq 1 - \delta$. By using the fact that $\Pi_{\mathcal{B}_1} \preceq \Pi_{\mathcal{B}_2}$ whenever $\mathcal{B}_1 \subseteq \mathcal{B}_2$ [Dereziński and Yang, 2024, Lemma 6.3], we have

$$\begin{aligned}
\mathbb{E}[\Pi_{\mathcal{B}}] &\succeq \mathbb{E}[\Pi_{\mathcal{B}} \mid \mathcal{E}_j] \Pr(\mathcal{E}_j) \\
&\succeq \mathbb{E}[\Pi_{\mathcal{B}_{j\text{-DPP}}} \mid \mathcal{E}_j] \Pr(\mathcal{E}_j) \\
&= \mathbb{E}[\Pi_{\mathcal{B}_{j\text{-DPP}}}] - \mathbb{E}[\Pi_{\mathcal{B}_{j\text{-DPP}}} \mid \mathcal{E}_j^c] \Pr(\mathcal{E}_j^c) \\
&\succeq \mathbb{E}[\Pi_{\mathcal{B}_{j\text{-DPP}}}] - \delta \cdot A^+A, \qquad (2.15)
\end{aligned}$$

where the last step follows since $\Pi_{\mathcal{B}_{j\text{-DPP}}}$ is an orthogonal projector onto a subspace of range(A^T), thus $\mathbb{E}[\Pi_{\mathcal{B}_{j\text{-DPP}}} \mid \mathcal{E}_j^c] \preceq \Pi_{[n]} = A^+A$. Let $\mathcal{B}_{\text{DPP}} \sim \text{DPP}(A/\bar{\lambda})$ and recall that $\mathbb{E}[|\mathcal{B}_{\text{DPP}}|] = d^{\bar{\lambda}}(A) = 2k$. Also, define $w_{\bar{\lambda},j} \coloneqq \Pr(|\mathcal{B}_{\text{DPP}}| = j)$. Using the law of total expectation, we can express the expected projection matrix for a DPP in terms of fixed-size j-DPPs:

$$\begin{aligned}
\mathbb{E}[\Pi_{\mathcal{B}_{\text{DPP}}}] &= \sum_{j\in\mathcal{I}} w_{\bar{\lambda},j}\mathbb{E}[\Pi_{\mathcal{B}_{\text{DPP}}} \mid |\mathcal{B}_{\text{DPP}}| = j] + \sum_{j\notin\mathcal{I}} w_{\bar{\lambda},j}\mathbb{E}[\Pi_{\mathcal{B}_{\text{DPP}}} \mid |\mathcal{B}_{\text{DPP}}| = j] \\
&= \sum_{j\in\mathcal{I}} w_{\bar{\lambda},j}\mathbb{E}[\Pi_{\mathcal{B}_{j\text{-DPP}}}] + \sum_{j\notin\mathcal{I}} w_{\bar{\lambda},j}\mathbb{E}[\Pi_{\mathcal{B}_{j\text{-DPP}}}],
\end{aligned}$$

where the second equality uses that $\mathbb{E}[\Pi_{\mathcal{B}_{\text{DPP}}} \mid |\mathcal{B}_{\text{DPP}}| = j] = \mathbb{E}[\Pi_{\mathcal{B}_{j\text{-DPP}}}]$ by definition. Recalling (2.15) and using $\Pi_{\mathcal{B}_{j\text{-DPP}}} \preceq \Pi_{[n]} = A^+A$, we can bound the preceding display as

$$\begin{aligned}
\mathbb{E}[\Pi_{\mathcal{B}_{\text{DPP}}}] &\preceq \left(\sum_{j\in\mathcal{I}} w_{\bar{\lambda},j}\right)\left(\mathbb{E}[\Pi_{\mathcal{B}}] + \delta \cdot A^+A\right) + \left(\sum_{j\notin\mathcal{I}} w_{\bar{\lambda},j}\right) A^+A \\
&\preceq \mathbb{E}[\Pi_{\mathcal{B}}] + \delta \cdot A^+A + \left(\sum_{j\notin\mathcal{I}} w_{\bar{\lambda},j}\right) A^+A,
\end{aligned}$$

where the last relation follows as $\sum_{j\in\mathcal{I}} w_{\bar{\lambda},j} \leq 1$. Now, as $\Pr(j \notin \mathcal{I}) = \sum_{j\notin\mathcal{I}} w_{\bar{\lambda},j}$, Lemma A.2 yields $\sum_{j\notin\mathcal{I}} w_{\bar{\lambda},j} \leq \delta$. Combining this with the preceding display, we deduce

$$\mathbb{E}[\Pi_{\mathcal{B}_{\text{DPP}}}] \preceq \mathbb{E}[\Pi_{\mathcal{B}}] + 2\delta A^+A.$$

Rearranging the preceding display yields

$$\mathbb{E}[\Pi_{\mathcal{B}}] \succeq \mathbb{E}[\Pi_{\mathcal{B}_{\text{DPP}}}] - 2\delta A^+A.$$

Applying Lemma 2.8 and setting $\delta \leq \min\{\lambda^+_{\min}(A)/(4(\lambda^+_{\min}(A) + \bar{\lambda})), n^{-c}\}$ for some constant $c > 0$,

so that $\delta = n^{-\mathcal{O}(1)}$, we conclude

$$\mathbb{E}[\Pi_{\mathcal{B}}] \succeq \frac{1}{2} A \left(A + \bar{\lambda} I\right)^{-1}, \tag{$*$}$$

which is precisely (2.13).

We now establish the lower bound on μ. To this end, we first show that $\bar{\lambda} \leq \frac{1}{k}\sum_{i>k}\lambda_i(A)$. Indeed, if we suppose $\bar{\lambda} > \frac{1}{k}\sum_{i>k}\lambda_i(A)$, then

$$2k \leq d^{\bar{\lambda}}(A) = \sum_{i=1}^{k} \frac{\lambda_i}{\lambda_i + \bar{\lambda}} + \sum_{i>k} \frac{\lambda_i}{\lambda_i + \bar{\lambda}} < k + \frac{\sum_{i>k}\lambda_i}{\bar{\lambda}} < k + k = 2k,$$

resulting in a contradiction. Hence $\bar{\lambda} \leq \frac{1}{k}\sum_{i>k}\lambda_i(A)$.

Finally, it follows from $(*)$ that

$$\mu = \lambda_{\min}^{+}\left(\mathbb{E}[\Pi_{\mathcal{B}}]\right) \geq \frac{1}{2}\lambda_{\min}^{+}\left(A\left(A + \bar{\lambda} I\right)^{-1}\right) = \frac{1}{2}\frac{\lambda_{\min}^{+}(A)}{\lambda_{\min}^{+}(A) + \bar{\lambda}} \geq \frac{1}{2\left(1 + \frac{1}{k}\sum_{i>k}\frac{\lambda_i(A)}{\lambda_{\min}^{+}(A)}\right)}$$

$$= \frac{1}{2\left(1 + \frac{r-k}{k}\frac{1}{r-k}\sum_{i>k}\frac{\lambda_i(A)}{\lambda_{\min}^{+}(A)}\right)} \geq \frac{k}{2\left(\frac{1}{r-k}\sum_{i>k}\frac{\lambda_i(A)}{\lambda_{\min}^{+}(A)}\right) r} = \frac{k}{2\bar{\kappa}_{k:r}(A) r}.$$

□

2.5.3 Lower Bound on $\hat{\mu}$

Having established a lower bound on μ, we now turn to lower bounding $\hat{\mu}$, which will allow us to establish a fine-grained convergence rate for `ASkotch`.

The main goal of this subsection is to establish the following theorem:

Theorem 2.13 (Lower bound on $\hat{\mu}$)**.** *Suppose the blocksize satisfies $b = \Omega(k \log^3 n)$ for some $k \in [n]$. Then using ARLS sampling as in Lemma 2.10 with $A = K_\lambda$ and $c = \mathcal{O}(1)$, the Nyström approximation as in Proposition 2.15, and defining $\bar{\lambda}$ as in Lemma 2.10,*

$$\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}] \succeq \frac{\lambda}{8\rho} K_\lambda (K_\lambda + \bar{\lambda} I)^{-1}.$$

Consequently, recalling $\bar{\kappa}_{k:n} = \frac{1}{n-k}\sum_{i>k}\lambda_i(K_\lambda)/\lambda_{\min}(K_\lambda)$, we have

$$\hat{\mu} \geq \frac{\lambda}{16\bar{\kappa}_{k:n}\rho}\frac{k}{n}.$$

Theorem 2.13 provides us with the desired lower bound on $\hat{\mu}$. When b is sufficiently large, the following corollary shows that $\hat{\mu}$ is lower bounded by a quantity independent of the conditioning of K_λ.

Corollary 2.14. *Fix a failure probability $\delta = n^{-\mathcal{O}(1)}$. Suppose the blocksize satisfies $b = \Omega\left(k \log^3 n\right)$ where $k \geq 2d^{\lambda}(K)$. If the rank of the Nyström approximation satisfies $r = \mathcal{O}\left(d^{\rho}(\lfloor K \rfloor_b) \log\left(\frac{d^{\rho}(\lfloor K \rfloor_b)}{\delta}\right)\right)$ and $\rho \geq \lambda$, then*

$$\hat{\mu} \geq \frac{\lambda}{32\rho}\frac{k}{n}.$$

When the effective blocksize $k = \Omega(d^{\lambda}(K))$, Corollary 2.14 shows the lower bound on $\hat{\mu}$ no longer depends on the conditioning of K_λ. Instead, $\hat{\mu}$ is controlled by k and ρ. Using a smaller ρ (larger r) leads to a larger $\hat{\mu}$. Conversely, larger ρ corresponds to using a smaller rank, and the lower bound on $\hat{\mu}$ decreases. We shall see shortly that Corollary 2.14 implies `Skotch` and `ASkotch` enjoy condition-number-free convergence rates.

The Nyström Projector is Close to the SAP Projector

Lemma 2.10 guarantees a tight lower bound on μ when $\mathcal{B}$ is an $\mathrm{ARLS}_c^{\tilde{\lambda}}$-sampling for any $c = \mathcal{O}(1)$ with respect to K_λ. Thus, if we can determine the form of $g(\rho, \lambda)$, we can obtain a lower bound on $\hat{\mu}$.

We begin with Proposition 2.15, which explicitly characterizes the *shrinkage factor* $\sigma_{P_\mathcal{B}}/\hat{L}_{P_\mathcal{B}}$ in Lemma 2.3 with high probability. The proof of Proposition 2.15 appears in Appendix A.2.4.

Proposition 2.15 (Controlling the shrinkage factor)**.** *Let $\mathcal{B} \subseteq [n]$ satisfy $|\mathcal{B}| = b < n$, and set $\rho \geq \lambda$. If the Nyström approximation $\hat{K}_{\mathcal{B}\mathcal{B}}$ of $K_{\mathcal{B}\mathcal{B}}$ is constructed from a sparse sign embedding Ω with $r = \mathcal{O}\left(d^{\rho}(\lfloor K \rfloor_b) \log\left(\frac{d^{\rho}(\lfloor K \rfloor_b)}{\delta}\right)\right)$ columns and $\zeta = \mathcal{O}\left(\log\left(\frac{d^{\rho}(\lfloor K \rfloor_b)}{\delta}\right)\right)$ non-zeros per column, then*

$$\frac{\lambda}{2\rho}\Pi_\mathcal{B} \preceq \hat{\Pi}_{\mathcal{B},\rho} \preceq \Pi_\mathcal{B}, \quad \textit{with probability at least } 1 - \delta.$$

The shrinkage factor obtained in Proposition 2.15 depends upon λ/ρ. If $\rho = \mathcal{O}(\lambda)$, then the shrinkage factor is *constant* with high probability, in which case little is lost by replacing $K_{\mathcal{B}\mathcal{B}}$ with the Nyström approximation $\hat{K}_{\mathcal{B}\mathcal{B}}$. Selecting a larger ρ allows smaller Nyström rank r, but incurs a trade-off in the shrinkage factor. We note that $d^{\rho}(\lfloor K \rfloor_b)$ is always upper-bounded by $d^{\rho}(K)$, and in some cases, may be substantially smaller.

Remark 2.16. While the guarantee in Proposition 2.15 is for when $\hat{K}_{\mathcal{B}\mathcal{B}}$ is constructed with a sparse sign embedding, the same guarantee also holds when Ω is a Gaussian embedding (which we use in our experiments) or a randomized Hadamard transform.

Proof of Theorem 2.13

Here we prove Theorem 2.13, which provides a lower bound on $\hat{\mu}$.

Proof. Define the event

$$\mathcal{E} = \left\{ \frac{\lambda}{2\rho}\Pi_\mathcal{B} \preceq \hat{\Pi}_{\mathcal{B},\rho} \preceq \Pi_\mathcal{B} \right\}.$$

Then Proposition 2.15 tells us that by setting δ appropriately, we can guarantee $\Pr(\mathcal{E}) \geq 1 - \frac{\mu}{2}$. Thus, the law of total expectation yields

$$\begin{aligned}\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}] &= \mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho} \mid \mathcal{E}] \Pr(\mathcal{E}) + \mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho} \mid, \mathcal{E}^C] \Pr(\mathcal{E}^C) \\ &\succeq \mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho} \mid \mathcal{E}] \Pr(\mathcal{E}) \\ &\succeq \frac{\lambda}{2\rho} \mathbb{E}\left[\Pi_{\mathcal{B}} \mid \mathcal{E}\right] \Pr(\mathcal{E}). \qquad (*)\end{aligned}$$

We now lower bound $\mathbb{E}\left[\Pi_{\mathcal{B}} \mid \mathcal{E}\right]$ in the Loewner ordering. To this end, the law of total expectation and a simple rearrangement imply

$$\mathbb{E}\left[\Pi_{\mathcal{B}} \mid \mathcal{E}\right] = \frac{1}{\Pr(\mathcal{E})} \left(\mathbb{E}[\Pi_{\mathcal{B}}] - \mathbb{E}[\Pi_{\mathcal{B}} \mid \mathcal{E}^C] \Pr(\mathcal{E}^C)\right).$$

The preceding display, along with the facts that $\Pi_{\mathcal{B}} \preceq I$ and $\Pr(\mathcal{E}^C) \leq \mu/2$, yields

$$\mathbb{E}\left[\Pi_{\mathcal{B}} \mid \mathcal{E}\right] \succeq \frac{1}{\Pr(\mathcal{E})} \left(\mathbb{E}[\Pi_{\mathcal{B}}] - \Pr(\mathcal{E}^C) I\right) \succeq \frac{1}{\Pr(\mathcal{E})} \left(\mathbb{E}[\Pi_{\mathcal{B}}] - \frac{\mu}{2} I\right).$$

As $\frac{\mu}{2} I \preceq \frac{1}{2}\mathbb{E}[\Pi_{\mathcal{B}}]$, we obtain

$$\mathbb{E}\left[\Pi_{\mathcal{B}} \mid \mathcal{E}\right] \succeq \frac{1}{2\Pr(\mathcal{E})} \mathbb{E}[\Pi_{\mathcal{B}}].$$

Thus, combining this last display with $(*)$ and applying Lemma 2.10 with $A = K_\lambda$, we conclude

$$\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}] \succeq \frac{\lambda}{4\rho} \mathbb{E}[\Pi_{\mathcal{B}}] \succeq \frac{\lambda}{8\rho} K_\lambda \left(K_\lambda + \bar{\lambda} I\right)^{-1}.$$

The claimed lower bound on $\hat{\mu}$ follows from this last display and Lemma 2.10. □

2.6 Convergence of `ASkotch`

We have established all the preliminary results for the convergence analysis of `ASkotch`. We begin with the following general convergence result (Theorem 2.17). In Section 2.6.1, we discuss when `Skotch` and `ASkotch` achieve near-optimal log-linear runtimes and how they can use large blocksizes to leverage the benefits offered by modern computing hardware.

Theorem 2.17 (Convergence of `ASkotch`)**.** *Consider `Skotch` and `ASkotch` (Algorithm 1) under the following hyperparameter settings: the blocksize satisfies $b = \Omega(k \log^3 n)$, where $k \geq 2d^\lambda(K)$, $\rho \geq \lambda$, $\mathcal{P}$ is a $\mathrm{ARLS}^{\tilde{\lambda}}_{\mathcal{O}(1)}$-sampling distribution for K_λ with $\tilde{\lambda} > 0$ such that $d^{\tilde{\lambda}}(K_\lambda) \geq 4k$, $\hat{K}_{\mathcal{BB}}$ is constructed from a sparse sign embedding with $r = \mathcal{O}\left(d^\rho(\lfloor K \rfloor_b) \log\left(\frac{d^\rho(\lfloor K \rfloor_b)}{\delta}\right)\right)$ columns and $\zeta = \mathcal{O}\left(\log\left(\frac{d^\rho(\lfloor K \rfloor_b)}{\delta}\right)\right)$ non-zeros per column, and $\eta_{\mathcal{B}} = 1/\hat{L}_{P_{\mathcal{B}}}$. Then the following statements hold:*

1. *After t iterations, the output of* `Skotch` *satisfies*

$$\mathbb{E}[\|w_t - w_\star\|_{K_\lambda}^2] \leq \left(1 - \frac{\lambda}{32\rho}\frac{k}{n}\right)^t \|w_0 - w_\star\|_{K_\lambda}^2.$$

Thus, the total number of iterations required to achieve an ϵ-approximate solution is bounded by $\mathcal{O}\left(\frac{\rho}{\lambda}\frac{n}{k}\log\left(\frac{1}{\epsilon}\right)\right)$.

2. *After t iterations, the output of* `ASkotch` *satisfies*

$$\mathbb{E}[\|w_t - w_\star\|_{K_\lambda}^2] \leq 2\left(1 - \frac{1}{16\sqrt{2}}\min\left\{\sqrt{\frac{\lambda}{\rho}}\frac{k}{n}, \sqrt{\frac{k}{n}}\frac{\lambda}{\rho}\right\}\right)^t \|w_0 - w_\star\|_{K_\lambda}^2.$$

Thus, the total number of iterations required to achieve an ϵ-approximate solution is bounded by $\mathcal{O}\left(\max\left\{\sqrt{\frac{\rho}{\lambda}}\frac{n}{k}, \frac{\rho}{\lambda}\sqrt{\frac{n}{k}}\right\}\log\left(\frac{1}{\epsilon}\right)\right)$.

The proof of Theorem 2.17 is provided in Section 2.6.2.

Theorem 2.17 shows `Skotch` and `ASkotch` converge linearly to the solution of (2.3). `Skotch` requires $\mathcal{O}(\frac{\rho}{\lambda}\frac{n}{k}\log\left(\frac{1}{\epsilon}\right))$ iterations to produce an ϵ-approximate solution, while `ASkotch` requires $\mathcal{O}\left(\max\left\{\sqrt{\frac{\rho}{\lambda}}\frac{n}{k}, \frac{\rho}{\lambda}\sqrt{\frac{n}{k}}\right\}\log\left(\frac{1}{\epsilon}\right)\right)$ iterations. As $\rho/\lambda, n/k \geq 1$, the upper bound for `ASkotch` is always an improvement over the upper bound for `Skotch`, demonstrating the benefit of acceleration.

2.6.1 `ASkotch`: Two Convergence Regimes

Theorem 2.17 shows that `ASkotch`'s convergence exhibits two different regimes: (i) $n/k > \rho/\lambda$, (ii) $\rho/\lambda > n/k$. In regime (i), the "low-rank approximation condition number" ρ/λ is dominated by the "dimensional condition number" n/k, and `ASkotch` converges in $\tilde{\mathcal{O}}\left(\sqrt{\frac{\rho}{\lambda}}\frac{n}{k}\right)$ iterations. In this regime, `ASkotch` can use a larger ρ (and hence, a smaller Nyström rank r) than `Skotch`, as the convergence rate depends only upon $\sqrt{\frac{\rho}{\lambda}}$. However, if ρ is set too large, so that $\rho > (n/k)\lambda$, `ASkotch` enters regime (ii), where the low-rank approximation condition number ρ/λ dominates the dimensional condition number n/k, leading to an iteration complexity of $\tilde{\mathcal{O}}\left(\frac{\rho}{\lambda}\sqrt{\frac{n}{k}}\right)$.

Which convergence regime of `ASkotch` is preferable? In the worst case, a n/k dependence in the iteration complexity is necessary, since `ASkotch` must make a full pass through K_λ to solve (2.3). Thus, regime (i) is preferable as it allows `ASkotch` to reduce the price ρ/λ of using a low-rank approximation. `ASkotch` is guaranteed to be in regime (i) provided it does not over-regularize, since the phase transition to regime (ii) occurs when $\rho > (n/k)\lambda$.

When the effective blocksize k is small, i.e., $k = o(n)$, n/k is large, so `ASkotch` can still use large ρ and small rank r while avoiding over-regularization. When the effective blocksize is large, i.e., $k = \Omega(n)$, $n/k = \mathcal{O}(1)$, so `ASkotch` must set $\rho = \mathcal{O}(\lambda)$ to stay in regime (i). Consequently, if K exhibits heavy-tailed spectral decay, the rank r might have to increase significantly for `ASkotch` to remain in regime (i). Fortunately, most kernel matrices exhibit fast spectral decay—indeed, under

standard regularity assumptions on the kernel function, $d^\lambda(K) = \mathcal{O}(\sqrt{n})$ [Rudi et al., 2017, 2018]. Thus, even if $k = \Omega(n)$, `ASkotch` remains in regime (i) when it uses rank $r = \mathcal{O}(\sqrt{n})$. Overall, convergence regime (i) for `ASkotch` is both preferable and also more likely to occur, compared to regime (ii).

When Does `ASkotch` Converge in Log-Linear Time?

A natural question to ask is if there is a reasonable setting where `Skotch` and `ASkotch` enjoy a near-optimal runtime of $\tilde{\mathcal{O}}(n^2)$. A priori, the answer is not obvious, as we use a low-rank approximation to $K_{\mathcal{BB}}$. Consequently, the price of using this approximation may be too steep to ensure an $\tilde{\mathcal{O}}(n^2)$ runtime. Fortunately, the following corollary shows that for matrices whose effective dimension is not too large, the low-rank approximation has a minimal effect on the overall runtime. As the effective dimension of kernel matrices is typically $\mathcal{O}(\sqrt{n})$ [Rudi et al., 2017, 2018], this corollary shows that `Skotch` and `ASkotch` can achieve a near-optimal runtime for most KRR tasks, despite approximating $K_{\mathcal{BB}}$.

Corollary 2.18 (Log-linear convergence of `ASkotch`)**.** *Let $\mathcal{P}$ in `ASkotch` be the uniform distribution. Let k and $\tilde{\lambda}$ be as in Lemma 2.10 and suppose that $\max_{i\in[n]} \ell_i^{\tilde{\lambda}}(K_\lambda) = \Theta(d^{\tilde{\lambda}}(K_\lambda)/n)$, $\rho = c_\rho\lambda$ for $c_\rho \in [1, n/k]$, and $d^\lambda(K) = \mathcal{O}(\sqrt{n})$. Then under the assumptions of Theorem 2.17, with blocksize $b = \Theta(k\log^3 n)$, the total runtime of `ASkotch` to produce an ϵ-approximate solution is bounded by*

$$\tilde{\mathcal{O}}\left(\sqrt{c_\rho}n^2\log\left(\frac{1}{\epsilon}\right)\right).$$

The proof of Corollary 2.18 appears in Appendix A.2.6. Corollary 2.18 shows that when (i) the effective dimension grows like $\mathcal{O}(\sqrt{n})$, (ii) the ridge leverage scores are relatively uniform, and (iii) `ASkotch` uses uniform sampling with $\rho = \mathcal{O}(\lambda)$, `ASkotch` obtains a near-optimal runtime.

Another powerful consequence of Corollary 2.18 is that the total runtime is independent of the blocksize. `ASkotch` can use effective blocksize $k = \mathcal{O}(n)$ (which corresponds to $b = \tilde{\mathcal{O}}(n)$) and still have runtime $\tilde{\mathcal{O}}\left(n^2\right)$. Thus, when the leverage scores are relatively uniform, `ASkotch` can use large blocksizes and obtain a log-linear runtime. From a practical perspective, this is invaluable, as large blocksizes can exploit the massive parallelism offered by modern computing architectures. Indeed, in our experiments, we use $b = n/100$ and observe excellent performance. The ability of `ASkotch` to use a large blocksize separates it from SAP, which incurs an $\mathcal{O}(n^3)$ cost per iteration when $b = \mathcal{O}(n)$.

Remark 2.19. The same runtime bounds hold for ARLS sampling, which allows us to avoid the assumption that the ridge leverage scores satisfy $\max_{i\in[n]} \ell_i^{\tilde{\lambda}}(K_\lambda) = \Theta\left(\frac{d^{\tilde{\lambda}}(K_\lambda)}{n}\right)$. Instead, we require that $k = \Theta(d^\lambda(K))$, since this ensures that the cost of running BLESS is bounded by $\tilde{\mathcal{O}}(n^2)$. Thus, ARLS cannot use blocksizes as large as uniform sampling while maintaining near-optimal runtime. We focus on uniform sampling, as this is what works best in practice.

2.6.2 Proof of Theorem 2.17

Here, we provide the convergence proof of `Skotch`, which corresponds to the first claim in Theorem 2.17. The convergence proof for `ASkotch` is more involved and requires additional background, so it is deferred to Appendix A.2.5.

Proof. We have seen in Section 2.4.2 that at iteration i,

$$\mathbb{E}\left[\|w_i - w_\star\|_{K_\lambda}^2 \mid w_{i-1}\right] \leq (1-\hat{\mu})\|w_{i-1} - w_\star\|_{K_\lambda}^2.$$

Invoking the lower bound on $\hat{\mu}$ in Corollary 2.14, this becomes

$$\mathbb{E}\left[\|w_i - w_\star\|_{K_\lambda}^2 \mid w_{i-1}\right] \leq \left(1 - \frac{\lambda k}{32\rho n}\right)\|w_{i-1} - w_\star\|_{K_\lambda}^2.$$

Applying the law of total expectation in this last display, we deduce

$$\mathbb{E}\left[\|w_t - w_\star\|_{K_\lambda}^2\right] \leq \left(1 - \frac{\lambda k}{32\rho n}\right)^t \|w_0 - w_\star\|_{K_\lambda}^2.$$

The claimed iteration complexity bound follows immediately. □

2.7 Experiments

We perform an empirical evaluation of `ASkotch`, EigenPro 2.0, EigenPro 3.0, PCG, and Falkon on KRR problems drawn from diverse application domains, including computer vision, particle physics, ecological modeling, online advertising, computational chemistry, music, socioeconomics, and transportation. We use three different kernels in our experiments: Laplacian, Matérn-5/2, and radial basis function (RBF), which are among the most popular in the literature [Rasmussen and Williams, 2005, Gardner et al., 2018]. Across this broad spectrum of application domains and kernels, `ASkotch` obtains better predictive performance than the competing methods. These results show that (i) full KRR can effectively scale to large problems and obtain better predictive performance than inducing points KRR and (ii) `ASkotch` is a new state-of-the-art solver for large-scale full KRR.

We present the following results:

- Performance comparisons (Section 2.7.1): We compare `ASkotch` to the state-of-the-art methods for full KRR and inducing points KRR: EigenPro 2.0 [Ma and Belkin, 2019] and PCG for full KRR, and Falkon [Rudi et al., 2017, Meanti et al., 2020] and EigenPro 3.0 [Abedsoltan et al., 2023] for inducing points KRR. For PCG, we consider two of the most effective types of preconditioners: Gaussian Nyström (Nyström) [Frangella et al., 2023] and randomly pivoted Cholesky (RPC) [Díaz et al., 2023, Epperly et al., 2025]. `ASkotch` consistently obtains better predictive performance than the competitors.

- Showcase on huge-scale transportation data analysis (Section 2.7.2): We run `ASkotch` on KRR for a subsample of the New York City taxi dataset ($n = 10^8$). To the best of our knowledge, no previous work has scaled full KRR to a dataset of this size. `ASkotch` once again outperforms the competition.

- Verification of linear convergence (Section 2.7.3): We show that `ASkotch` achieves global linear convergence on three KRR problems with $n \geq 5 \cdot 10^5$ samples, which verifies the linear convergence guarantee in Section 2.6. `ASkotch` reaches machine precision in at most 100 passes through each KRR linear system, demonstrating its potential as a high-precision solver.

Each experiment is run on a single 48 GB NVIDIA RTX A6000 GPU with PyTorch 2.5.1 [Paszke et al., 2019], CUDA 12.5, PyKeOps 2.2.3 [Charlier et al., 2021], and Python 3.10.12. The code for our experiments is available at https://github.com/pratikrathore8/fast_krr.

Our arXiv report (https://arxiv.org/abs/2407.10070v3) includes expanded experimental results and an ablation study that are omitted from this version of the paper. We also provide additional information about datasets, kernel hyperparameters, regularizations, inducing points, preprocessing, and time limits in Appendix C of our arXiv report.

Optimizer hyperparameters. Throughout the experiments, we use the default hyperparameters that we recommend for `ASkotch` (Section 2.2.5), unless stated otherwise.

We compute the damping for PCG with the Gaussian Nyström preconditioner using the same procedure as `ASkotch`, ensuring a fair comparison. We also run PCG, EigenPro 2.0, and EigenPro 3.0 with the same rank as `ASkotch` to ensure a fair comparison. We set b_g, s, and the stepsize η in EigenPro 2.0 and EigenPro 3.0 using the defaults in the EigenPro 2.0 and EigenPro 3.0 GitHub repositories. We also run EigenPro 2.0 and 3.0 with regularization $\lambda = 0$, as recommended in Ma and Belkin [2019], Abedsoltan et al. [2023].

Numerical precision. `ASkotch`, EigenPro 2.0, and EigenPro 3.0 are stable in single precision, while Falkon and PCG often require double precision for satisfactory performance. Consequently, the figures in the main paper show results when `ASkotch`, EigenPro 2.0, and EigenPro 3.0 run in single precision and Falkon and PCG run in double precision.

We show in Appendix C of our arXiv report that `ASkotch` still outperforms Falkon and PCG when all methods are run in single precision.

Time limits. Each optimizer is run until it reaches a specified time limit, which depends on the size of the dataset. These time limits range between 0.5 to 24 hours. We implement all of the competing methods ourselves to ensure fair runtime comparisons with `ASkotch`. The most expensive computations in all methods are performed using KeOps [Charlier et al., 2021], which ensures our runtime comparisons are fair.

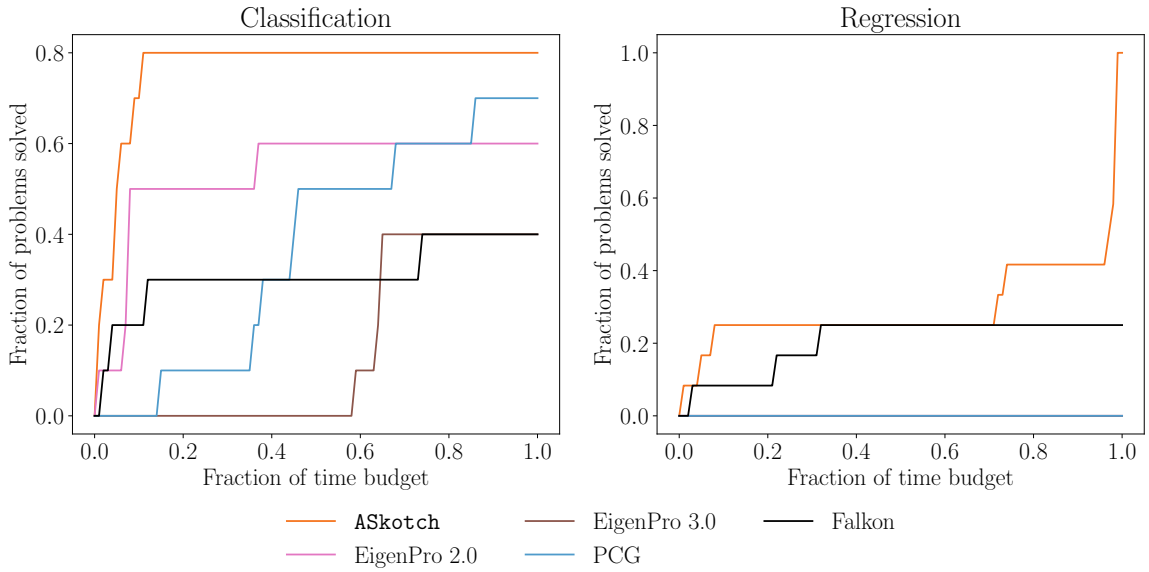


Figure 2.3: Performance comparison between `ASkotch` and competitors on 10 classification and 13 regression tasks. We designate a classification problem as "solved" when the method reaches within 0.001 of the highest classification accuracy found across all the optimizer + hyperparameter combinations. We designate a regression problem as "solved" when the method reaches within 1% of the lowest MAE (in a relative sense) found across all the optimizer + hyperparameter combinations. PCG and Falkon are run in double precision. EigenPro 2.0, EigenPro 3.0, and PCG do not solve any of the regression problems within the tolerance. `ASkotch` outperforms the competition on both classification and regression.

2.7.1 Performance Comparisons

We evaluate predictive performance on 10 classification and 13 regression tasks from established benchmarks. The classification tasks are from computer vision, particle physics, ecological modeling, and online advertising, while the regression tasks are from computational chemistry, music analysis, and socioeconomics. We assess predictive performance on classification and regression tasks by computing classification accuracy and mean-absolute error (MAE) between the predictions and targets, respectively, on the test set.

We summarize the results of our performance comparisons in Fig. 2.3. `ASkotch` outperforms the competition on both classification and regression. Notably, `ASkotch` outperforms the state-of-the-art inducing point methods EigenPro 3.0 and Falkon, demonstrating the value of using full KRR over inducing points KRR.

2.7.2 Showcase: Huge-Scale Transportation Data Analysis

We apply KRR to a subsample of the taxi dataset to predict taxi ride durations in New York City. Following Meanti et al. [2020], we use an RBF kernel. Due to hardware limitations, we set the blocksize b in `ASkotch` at $n/2{,}000 = 5 \cdot 10^4$ and vary the rank $r \in \{50, 100, 200, 500\}$. We set the rank for PCG as low as $r = 50$, but none of the PCG methods complete a single iteration in the time limit.

Following Meanti et al. [2020], we use the root mean square error (RMSE) between the predictions and targets.

The results are shown in Fig. 2.1. `ASkotch` outperforms Falkon for each value of r. Both EigenPro 2.0 and EigenPro 3.0 (not shown) diverge. Once again, our findings demonstrate the value of using full KRR over inducing points KRR for large-scale regression tasks.

2.7.3 `ASkotch` Converges Linearly to the Optimum

We demonstrate that `ASkotch` obtains linear convergence to the optimum for large-scale full KRR. To do so, we plot the relative residual

$$\|K_\lambda w - y\|/\|y\|$$

obtained by `ASkotch`. We run `ASkotch` in double precision.

Fig. 2.4 shows the relative residual obtained by `ASkotch` on three large-scale KRR problems. `ASkotch` converges linearly for all selections of the rank r. Excitingly, `ASkotch` reaches machine precision on all three problems, showing its potential as a high-precision linear system solver.

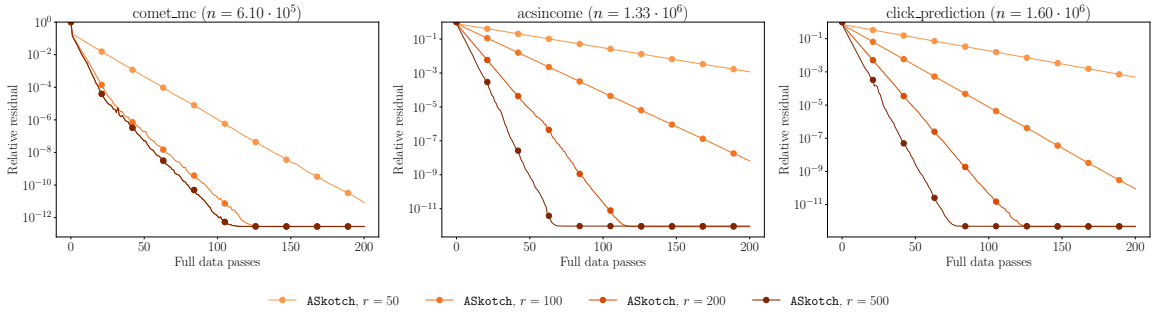


Figure 2.4: `ASkotch` converges linearly on large-scale full KRR. "Full data passes" indicates the number of passes through K_λ: since $b = n/100$, one full data pass is equivalent to 100 iterations of `ASkotch`. `ASkotch` tends to converge faster as the rank r increases, which matches our theoretical results. Interestingly, $r = 200$ and $r = 500$ yield similar results on comet_mc.

2.8 Conclusion

We introduce `ASkotch`, an approximate sketch-and-project method for large-scale full KRR. Our theoretical analysis and experiments demonstrate that `ASkotch` is a promising replacement for existing KRR solvers. Looking ahead, we aim to develop methods for automatically selecting the acceleration parameters $\hat{\mu}$ and $\hat{\nu}$ in `ASkotch`, create a distributed implementation of `ASkotch` that scales to datasets with $n \gtrsim 10^9$ training points (such as the full taxi dataset), and develop a mixed-precision version of `ASkotch` that delivers high-quality results with less memory and compute.

Our work, along with previous studies [Wang et al., 2019, Frangella et al., 2023, Díaz et al., 2023], shows that full KRR allows better predictive performance than inducing points KRR. `ASkotch` scales full KRR to datasets orders of magnitude larger than previously possible. In future work, we look forward to seeing how a fast solver for full KRR enables new applications. Moreover, following our methodology, it should be possible to solve general pd linear systems within the (approximate) sketch-and-project framework.

Chapter 3

Challenges in Training PINNs: A Loss Landscape Perspective

We study the loss landscape of physics-informed neural networks (PINNs) and diagnose why these models are difficult to train. Empirically, we show that the PINN loss is severely ill-conditioned because of the differential operators in its residual term, and that the standard Adam+L-BFGS pipeline (Adam followed by L-BFGS) can stall at a high loss when L-BFGS line search fails. To address this failure mode, we introduce NysNewton-CG (NNCG), a matrix-free second-order optimizer that preconditions a Newton-CG solve with a Nyström approximation of the Hessian. We also show that ill-conditioned differential operators induce ill-conditioned PINN losses, giving a theoretical justification for our empirical observations.

This chapter is based on Rathore et al. [2024], which appeared at the International Conference on Machine Learning (ICML) in 2024 and was selected for an oral presentation.

3.1 Introduction

The study of Partial Differential Equations (PDEs) grounds a wide variety of scientific and engineering fields, yet these fundamental physical equations are often difficult to solve numerically. Recently, neural network-based approaches including physics-informed neural networks (PINNs) have shown promise to solve both forward and inverse problems involving PDEs [Raissi et al., 2019, E and Yu, 2018, Lu et al., 2021a,b, Karniadakis et al., 2021, Cuomo et al., 2022]. PINNs parameterize the solution to a PDE with a neural network, and are often fit by minimizing a least-squares loss involving the PDE residual, boundary condition(s), and initial condition(s). The promise of PINNs is the potential to obtain solutions to PDEs without discretizing or meshing the space, enabling scalable solutions to high-dimensional problems that currently require weeks on advanced supercomputers.

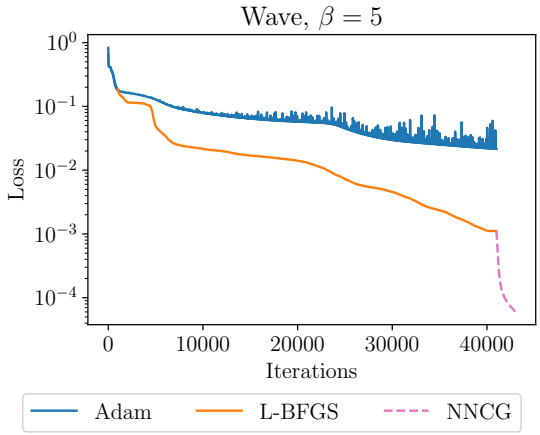


Figure 3.1: On the wave PDE, Adam converges slowly due to ill-conditioning and the combined Adam+L-BFGS optimizer stalls after about 40000 steps. Running NNCG (our method) after Adam+L-BFGS provides further improvement.

This loss is typically minimized with gradient-based optimizers such as Adam [Kingma and Ba, 2014], L-BFGS [Liu and Nocedal, 1989], or a combination of both.

However, the challenge of optimizing PINNs restricts the application and development of these methods. Previous work has shown that the PINN loss is difficult to minimize [Krishnapriyan et al., 2021, Wang et al., 2021a, 2022b, De Ryck et al., 2023] even in simple settings. As a result, the PINN often fails to learn the solution. Furthermore, optimization challenges can obscure the effectiveness of new neural network architectures for PINNs, as an apparently inferior performance may stem from insufficient loss function optimization rather than inherent limitations of an architecture. A simple, reliable training paradigm is critical to enable wider adoption of PINNs.

This work explores the loss landscape of PINNs and the challenges this landscape poses for gradient-based optimization methods. We provide insights from optimization theory that explain slow convergence of first-order methods such as Adam and show how ill-conditioned differential operators make optimization difficult. We also use our theoretical insights to improve the PINN training pipeline by combining existing and new optimization methods.

The most closely related works to ours are Krishnapriyan et al. [2021], De Ryck et al. [2023], which both identify ill-conditioning in the PINN loss. Unlike Krishnapriyan et al. [2021], we empirically confirm the ill-conditioning of the loss by visualizing the spectrum of the Hessian and demonstrating how quasi-Newton methods improve the conditioning. Our theoretical results directly show how an ill-conditioned linear operator induces an ill-conditioned objective, in contrast to the approach in De Ryck et al. [2023] which relies on a linearization.

Contributions. We highlight contributions of this paper:

- We demonstrate that the loss landscape of PINNs is ill-conditioned due to differential operators

in the residual term and show that quasi-Newton methods improve the conditioning by $1000\times$ or more (Section 3.5).

- We compare three optimizers frequently used for training PINNs: (i) Adam, (ii) L-BFGS, and (iii) Adam followed by L-BFGS (referred to as Adam+L-BFGS). We show that Adam+L-BFGS is superior across a variety of network sizes (Section 3.6).
- We show the PINN solution resembles the true PDE solution only for extremely small loss values (Section 3.4). However, we find that the loss returned by Adam+L-BFGS can be improved further, which also improves the PINN solution (Section 3.7).
- Motivated by the ill-conditioned loss landscape, we introduce a novel second-order optimizer, NysNewton-CG (NNCG). We show NNCG can significantly improve the solution returned by Adam+L-BFGS (Fig. 3.1 and Section 3.7).
- We prove that ill-conditioned differential operators lead to an ill-conditioned PINN loss (Section 3.8). We also prove that combining first- and second-order methods (e.g., Adam+L-BFGS) leads to fast convergence, providing justification for the importance of the combined method (Section 3.8).

Notation. We denote the Euclidean norm by $\|\cdot\|_2$ and use $\|M\|$ to denote the operator norm of $M \in \mathbb{R}^{m\times n}$. For a smooth function $f : \mathbb{R}^p \to \mathbb{R}$, we denote its gradient at $w \in \mathbb{R}^p$ by $\nabla f(w)$ and its Hessian by $H_f(w)$. We write $\partial_{w_i} f$ for $\partial f/\partial w_i$. For $\Omega \subset \mathbb{R}^d$, we denote its boundary by $\partial\Omega$. For any $m \in \mathbb{N}$, we use I_m to denote the $m \times m$ identity matrix. Finally, we use $\preceq$ to denote the Loewner ordering on the convex cone of positive semidefinite matrices.

3.2 Problem Setup

This section introduces physics-informed neural networks as optimization problems and our experimental methodology.

3.2.1 Physics-Informed Neural Networks

The goal of physics-informed neural networks is to solve partial differential equations. Similar to prior work [Lu et al., 2021b, Hao et al., 2023], we consider the following system of partial differential equations:

$$\mathcal{D}[u(x), x] = 0, \quad x \in \Omega, \tag{3.1a}$$

$$\mathcal{B}[u(x), x] = 0, \quad x \in \partial\Omega, \tag{3.1b}$$

where $\mathcal{D}$ is a differential operator defining the PDE, $\mathcal{B}$ is an operator associated with the boundary and/or initial conditions, and $\Omega \subseteq \mathbb{R}^d$. To solve (3.1), PINNs model u as a neural network $u(x; w)$

(often a multi-layer perceptron (MLP)) and approximate the true solution by the network whose weights solve the following non-linear least-squares problem:

$$\underset{w\in\mathbb{R}^p}{\text{minimize}}\ L(w) \coloneqq \frac{1}{2n_{\text{res}}}\sum_{i=1}^{n_{\text{res}}}\left(\mathcal{D}[u(x_r^i;w),x_r^i]\right)^2 + \frac{1}{2n_{\text{bc}}}\sum_{i=1}^{n_{\text{bc}}}\left(\mathcal{B}[u(x_b^j;w),x_b^j]\right)^2. \tag{3.2}$$

Here $\{x_r^i\}_{i=1}^{n_{\text{res}}}$ are the residual points and $\{x_b^j\}_{j=1}^{n_{\text{bc}}}$ are the boundary/initial points. The first loss term measures how much $u(x;w)$ fails to satisfy the PDE, while the second term measures how much $u(x;w)$ fails to satisfy the boundary/initial conditions.

For this loss, $L(w) = 0$ means that $u(x;w)$ exactly satisfies the PDE and boundary/initial conditions at the training points. In deep learning, this condition is called *interpolation* [Zhang et al., 2021, Belkin, 2021]. There is no noise in (3.1), so the true solution of the PDE would make (3.2) equal to zero. Hence a PINN approach should choose an architecture and an optimizer to achieve interpolation. Moreover, smaller training error corresponds to better generalization for PINNs [Mishra and Molinaro, 2023]. Common optimizers for (3.2) include Adam, L-BFGS, and Adam+L-BFGS [Raissi et al., 2019, Krishnapriyan et al., 2021, Hao et al., 2023].

3.2.2 Experimental Methodology

We conduct experiments on optimizing PINNs for convection, wave PDEs, and a reaction ODE. These equations have been studied in previous works investigating difficulties in training PINNs; we use the formulations in Krishnapriyan et al. [2021], Wang et al. [2022b] for our experiments. The coefficient settings we use for these equations are considered challenging in the literature [Krishnapriyan et al., 2021, Wang et al., 2022b]. Appendix B.1 contains additional details.

We compare the performance of Adam, L-BFGS, and Adam+L-BFGS on training PINNs for all three classes of PDEs. For Adam, we tune the learning rate by a grid search on $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$. For L-BFGS, we use the default learning rate 1.0, memory size 100, and strong Wolfe line search. For Adam+L-BFGS, we tune the learning rate for Adam as before, and also vary the switch from Adam to L-BFGS (after 1000, 11000, 31000 iterations). These correspond to Adam+L-BFGS (1k), Adam+L-BFGS (11k), and Adam+L-BFGS (31k) in our figures. All three methods are run for a total of 41000 iterations.

We use multilayer perceptrons (MLPs) with tanh activations and three hidden layers. These MLPs have widths 50, 100, 200, or 400. We initialize these networks with the Xavier normal initialization [Glorot and Bengio, 2010] and all biases equal to zero. Each combination of PDE, optimizer, and MLP architecture is run with 5 random seeds.

We use 10000 residual points randomly sampled from a 255×100 grid on the interior of the

problem domain. We use 257 equally spaced points for the initial conditions and 101 equally spaced points for each boundary condition.

We assess the discrepancy between the PINN solution and the ground truth using ℓ_2 relative error (L2RE), a standard metric in the PINN literature. Let $y = (y_i)_{i=1}^n$ be the PINN prediction and $y' = (y'_i)_{i=1}^n$ the ground truth. Define

$$\text{L2RE} = \sqrt{\frac{\sum_{i=1}^n (y_i - y'_i)^2}{\sum_{i=1}^n y_i'^2}} = \sqrt{\frac{\|y - y'\|_2^2}{\|y'\|_2^2}}.$$

We compute the L2RE using all points in the 255×100 grid on the interior of the problem domain, along with the 257 and 101 points used for the initial and boundary conditions.

We develop our experiments in PyTorch 2.0.0 [Paszke et al., 2019] with Python 3.10.12. Each experiment is run on a single NVIDIA Titan V GPU using CUDA 11.8. The code for our experiments is available at https://github.com/pratikrathore8/opt_for_pinns.

3.3 Related Work

Here we review common approaches for solving PDEs with physics-informed machine learning and PINN training strategies proposed in the literature.

3.3.1 Physics-Informed ML for Solving PDEs

A variety of ML-based methods for solving PDEs have been proposed, including PINNs [Raissi et al., 2019], the Fourier Neural Operator (FNO) [Li et al., 2021], and DeepONet [Lu et al., 2021a]. The PINN approach solves the PDE by using the loss function to penalize deviations from the PDE residual, boundary, and initial conditions. Notably, PINNs do not require knowledge of the solution to solve the forward PDE problem. On the other hand, the FNO and DeepONet sample and learn from known solutions to a parameterized class of PDEs to solve PDEs with another fixed value of the parameter. However, these operator learning approaches may not produce predictions consistent with the underlying physical laws that produced the data, which has led to the development of hybrid approaches such as physics-informed DeepONet [Wang et al., 2021c]. Our theory shows that the ill-conditioning issues we study in PINNs are unavoidable for any ML-based approach that penalizes deviations from the known physical laws.

3.3.2 Challenges in Training PINNs

The vanilla PINN [Raissi et al., 2019] can perform poorly when trying to solve high-dimensional, non-linear, and/or multi-scale PDEs. Researchers have proposed a variety of modifications to the vanilla PINN to address these issues, many of which attempt to make the optimization problem

easier to solve. Wang et al. [2021a, 2022a,b], Nabian et al. [2021], Wu et al. [2023a,b] propose loss reweighting/resampling to balance different components of the loss, Yao et al. [2023], Müller and Zeinhofer [2023] propose scale-invariant and natural gradient-based optimizers for PINN training, Jagtap et al. [2020a,b], Wang et al. [2023] propose adaptive activation functions which can accelerate convergence of the optimizer, and Liu et al. [2024] propose an approach to precondition the PINN loss itself. Other approaches include innovative loss functions and regularizations [E and Yu, 2018, Lu et al., 2021c, Kharazmi et al., 2021, Khodayi-Mehr and Zavlanos, 2020, Yu et al., 2022] and new architectures [Jagtap et al., 2020c, Jagtap and Karniadakis, 2020, Li et al., 2020, Moseley et al., 2023]. These strategies work with varying degrees of success, and no single strategy improves performance across all PDEs.

Our work attempts to understand and tame the ill-conditioning in the (vanilla) PINN loss directly. We expect our ideas to work well with many of the above training strategies for PINNs; none of these training strategies rid the objective of the differential operator that generates the ill-conditioning in the PINN loss (with the possible exception of Liu et al. [2024]). However, Liu et al. [2024] preconditions the PINN loss directly, which is equivalent to left preconditioning, while our work studies the effects of preconditioned optimization methods on the PINN loss, which is equivalent to right preconditioning (Appendix B.3.1). There is potential in combining the approach of Liu et al. [2024] and our approach to obtain a more reliable framework for training PINNs.

Our work analyzes the spectrum (eigenvalues) of the Hessian H_L of the loss. Previous work [Wang et al., 2022b] studies the conditioning of the loss using the neural tangent kernel (NTK), which requires an infinite-width assumption on the neural network; our work studies the conditioning of the loss through the lens of the Hessian and yields useful results for finite-width PINN architectures. Several works have also studied the *spectral bias* of PINNs [Wang et al., 2021b, 2022b, Moseley et al., 2023], which refers to the inability of neural networks to learn high-frequency functions. Note that our paper uses the word spectrum to refer to the Hessian eigenvalues, not the spectrum of the PDE solution.

3.4 Good Solutions Require Near-Zero Loss

First, we show that PINNs must be trained to near-zero loss to obtain a reasonably low L2RE. This phenomenon can be observed in Fig. 3.2, demonstrating that a lower loss generally corresponds to a lower L2RE. For example, on the convection PDE, a loss of 10^{-3} yields an L2RE around 10^{-1}, but decreasing the loss by a factor of 100 to 10^{-5} yields an L2RE around 10^{-2}, a $10\times$ improvement. This relationship between loss and L2RE in Fig. 3.2 is typical of many PDEs [Lu et al., 2022].

The relationship in Fig. 3.2 underscores that high-accuracy optimization is required for a useful PINN. There are instances (especially on the reaction ODE), where the PINN solution has a L2RE around 1, despite a near-zero loss; we provide insight into why this is occurring in Appendix B.2. In

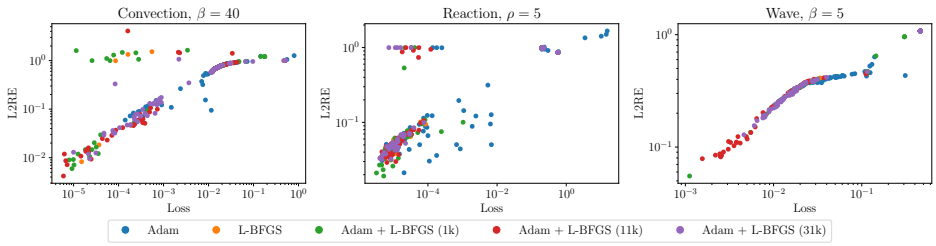


Figure 3.2: We plot the final L2RE against the final loss for each combination of network width, optimization strategy, and random seed. Across all three PDEs, a lower loss generally corresponds to a lower L2RE.

Sections 3.5 and 3.7, we show that ill-conditioning and under-optimization make reaching a solution with sufficient accuracy difficult.

3.5 The Loss Landscape is Ill-Conditioned

We show empirically that the ill-conditioning of the PINN loss is mainly due to the residual loss, which contains the differential operator. We also show that quasi-Newton methods like L-BFGS improve the conditioning of the problem.

3.5.1 The PINN Loss is Ill-Conditioned

The conditioning of the loss L plays a key role in the performance of first-order optimization methods [Nesterov, 2018]. We can understand the conditioning of an optimization problem through the eigenvalues of the Hessian of the loss, H_L. Intuitively, the eigenvalues of H_L provide information about the local curvature of the loss function at a given point along different directions. The condition number is defined as the ratio of the eigenvalue of largest magnitude to the eigenvalue of smallest magnitude. A large condition number implies the loss is very steep in some directions and flat in others, making it difficult for first-order methods to make sufficient progress toward the minimum. When $H_L(w)$ has a large condition number (particularly, for w near the optimum), the loss L is called *ill-conditioned*. For example, the convergence rate of gradient descent (GD) depends on the condition number [Nesterov, 2018], which results in GD converging slowly on ill-conditioned problems.

To investigate the conditioning of the PINN loss L, we would like to examine the eigenvalues of the Hessian. For large matrices, it is convenient to visualize the set of eigenvalues via *spectral density*, which approximates the distribution of the eigenvalues. Fast approximation methods for the spectral density of the Hessian are available for deep neural networks [Ghorbani et al., 2019, Yao et al., 2020]. Fig. 3.3 shows the estimated Hessian spectral density (solid lines) of the PINN loss for the convection, reaction, and wave problems after training with Adam+L-BFGS. For all three

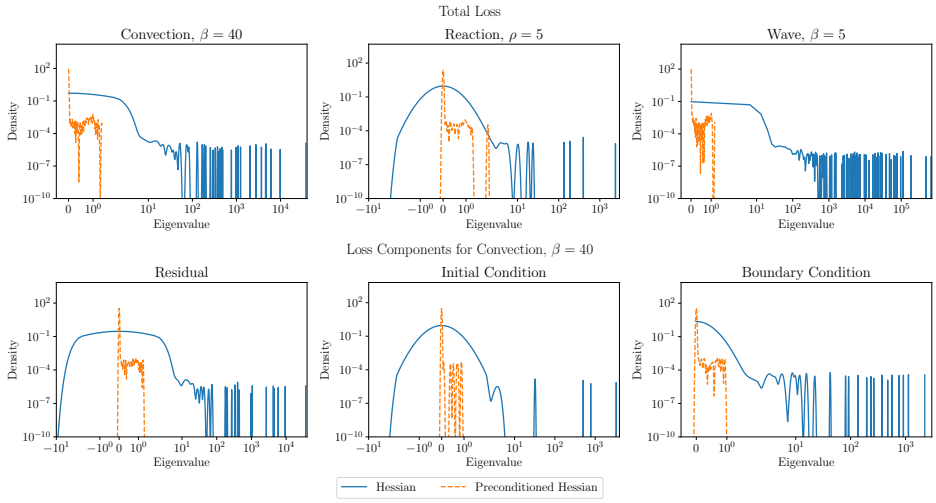


Figure 3.3: (Top) Spectral density of the Hessian and the preconditioned Hessian after 41000 iterations of Adam+L-BFGS. The plots show that the PINN loss is ill-conditioned and that L-BFGS improves the conditioning, reducing the top eigenvalue by 10^3 or more.
(Bottom) Spectral density of the Hessian and the preconditioned Hessian of each loss component after 41000 iterations of Adam+L-BFGS for convection. The plots show that each component loss is ill-conditioned and that the conditioning is improved by L-BFGS.

problems, we observe large outlier eigenvalues ($> 10^4$ for convection, $> 10^3$ for reaction, and $> 10^5$ for wave) in the spectrum, and a significant spectral density near 0, implying that the loss L is ill-conditioned. The plots also show how the spectrum is improved by preconditioning (Section 3.5.3).

3.5.2 The Ill-Conditioning is Due to the Residual Loss

We use the same method to study the conditioning of each component of the PINN loss. Figs. 3.3 and B.2 show the estimated spectral density of the Hessian of the residual, initial condition, and boundary condition components of the PINN loss for each problem after training with Adam+L-BFGS. We see residual loss, which contains the differential operator $\mathcal{D}$, is the most ill-conditioned among all components. Our theory (Section 3.8) shows this ill-conditioning is likely due to the ill-conditioning of $\mathcal{D}$.

3.5.3 L-BFGS Improves Problem Conditioning

Preconditioning is a popular technique for improving conditioning in optimization. A classic example is Newton's method, which uses second-order information (i.e., the Hessian) to (locally) transform an ill-conditioned loss landscape into a well-conditioned one. L-BFGS is a quasi-Newton method that improves conditioning without explicit access to the problem Hessian. To examine the effectiveness of quasi-Newton methods for optimizing L, we compute the spectral density of the Hessian after

Table 3.1: Lowest loss for Adam, L-BFGS, and Adam+L-BFGS across all network widths after hyperparameter tuning. Adam+L-BFGS attains both smaller loss and L2RE vs. Adam or L-BFGS.

Optimizer	Convection		Reaction		Wave	
	Loss	L2RE	Loss	L2RE	Loss	L2RE
Adam	1.40e-4	5.96e-2	4.73e-6	2.12e-2	2.03e-2	3.49e-1
L-BFGS	1.51e-5	8.26e-3	8.93e-6	3.83e-2	1.84e-2	3.35e-1
Adam+L-BFGS	**5.95e-6**	**4.19e-3**	**3.26e-6**	**1.92e-2**	**1.12e-3**	**5.52e-2**

L-BFGS preconditioning. (For details of this computation and how L-BFGS preconditions, see Appendix B.3.) Fig. 3.3 shows this preconditioned Hessian spectral density (dashed lines). For all three problems, the magnitude of eigenvalues and the condition number has been reduced by at least 10^3. In addition, the preconditioner improves the conditioning of each individual loss component of L (Figs. 3.3 and B.2). These observations offer clear evidence that quasi-Newton methods improve the conditioning of the loss, and show the importance of quasi-Newton methods in training PINNs, which we demonstrate in Section 3.6.

3.6 Adam+L-BFGS Optimizes the Loss Better Than Other Methods

We demonstrate that the combined optimization method Adam+L-BFGS consistently provides a smaller loss and L2RE than using Adam or L-BFGS alone. We justify this finding using intuition from optimization theory.

3.6.1 Adam+L-BFGS vs Adam or L-BFGS

Fig. B.3 in Appendix B.4 compares Adam+L-BFGS, Adam, and L-BFGS on the convection, reaction, and wave problems at difficult coefficient settings noted in the literature [Krishnapriyan et al., 2021, Wang et al., 2022b]. Across each network width, the lowest loss and L2RE is always delivered by Adam+L-BFGS. Similarly, the lowest median loss and L2RE are almost always delivered by Adam+L-BFGS (Fig. B.3). The only exception is the reaction problem, where Adam outperforms Adam+L-BFGS on loss at width = 100 and L2RE at width = 200 (Fig. B.3).

Table 3.1 summarizes the best performance of each optimizer. Again, Adam+L-BFGS is better than running either Adam or L-BFGS alone. Notably, Adam+L-BFGS attains 14.2× smaller L2RE than Adam on the convection problem and 6.07× smaller L2RE than L-BFGS on the wave problem.

3.6.2 Intuition From Optimization Theory

The success of Adam+L-BFGS over Adam and L-BFGS can be explained by existing results in optimization theory. In neural networks, saddle points typically outnumber local minima [Dauphin

et al., 2014, Lee et al., 2019]. A saddle point can never be a global minimum. We want to reach a global minimum when training PINNs.

Newton's method (which L-BFGS attempts to approximate) is attracted to saddle points [Dauphin et al., 2014], and quasi-Newton methods such as L-BFGS also converge to saddle points since they ignore negative curvature [Dauphin et al., 2014]. On the other hand, first-order methods such as gradient descent and AdaGrad [Duchi et al., 2011] avoid saddle points [Lee et al., 2019, Antonakopoulos et al., 2022]. We expect that (full-gradient) Adam also avoids saddles for similar reasons, although we are not aware of such a result.

Alas, first-order methods converge slowly when the problem is ill-conditioned. This result generalizes the well-known slow convergence of conjugate gradient (CG) for ill-conditioned linear systems: $\mathcal{O}(\sqrt{\kappa}\log(\frac{1}{\epsilon}))$ iterations to converge to an ϵ-approximate solution of a system with condition number κ. In optimization, an analogous notion of a condition number in a set $\mathcal{S}$ near a global minimum is given by $\kappa_f(\mathcal{S}) \coloneqq \sup_{w\in\mathcal{S}} \|H_f(w)\|/\mu$, where μ is the PŁ*-constant (see Section 3.8). Then gradient descent requires $\mathcal{O}(\kappa_f(\mathcal{S})\log(\frac{1}{\epsilon}))$ iterations to converge to an ϵ-suboptimal point. For PINNs, the condition number near a solution is often $> 10^4$ (Fig. 3.3), which leads to slow convergence of first-order methods. However, Newton's method and L-BFGS can significantly reduce the condition number (Fig. 3.3), which yields faster convergence.

Adam+L-BFGS combines the best of both first- and second-order/quasi-Newton methods. By running Adam first, we avoid saddle points that would attract L-BFGS. By running L-BFGS after Adam, we can reduce the condition number of the problem, which leads to faster local convergence. Fig. 3.1 exemplifies this, showing faster convergence of Adam+L-BFGS over Adam on the wave equation.

This intuition also explains why Adam sometimes performs as well as Adam+L-BFGS on the reaction problem. Fig. 3.3 shows the largest eigenvalue of the reaction problem is around 10^3, while the largest eigenvalues of the convection and wave problems are around 10^4 and 10^5, suggesting the reaction problem is less ill-conditioned.

3.7 The Loss is Often Under-Optimized

In Section 3.6, we show that Adam+L-BFGS improves on running Adam or L-BFGS alone. However, even Adam+L-BFGS does not reach a critical point of the loss: the loss is still under-optimized. We show that the loss and L2RE can be further improved by running a damped version of Newton's method.

3.7.1 Why is the Loss Under-Optimized?

Fig. 3.4 shows the run of Adam+L-BFGS with smallest L2RE for each PDE. For each run, L-BFGS stops making progress before reaching the maximum number of iterations. L-BFGS uses *strong*

Table 3.2: Loss and L2RE after fine-tuning by NNCG and GD. NNCG outperforms both GD and the original Adam+L-BFGS results.

Optimizer	Convection		Reaction		Wave	
	Loss	L2RE	Loss	L2RE	Loss	L2RE
Adam+L-BFGS	5.95e-6	4.19e-3	5.26e-6	1.92e-2	1.12e-3	5.52e-2
Adam+L-BFGS+NNCG	**3.63e-7**	**1.94e-3**	**2.89e-7**	**9.92e-3**	**6.13e-5**	**1.27e-2**
Adam+L-BFGS+GD	5.95e-6	4.19e-3	5.26e-6	1.92e-2	1.12e-3	5.52e-2

Wolfe line search, as it is needed to maintain the stability of L-BFGS [Nocedal and Wright, 2006]. L-BFGS often terminates because it cannot find a positive step size satisfying these conditions—we have observed several instances where L-BFGS picks a step size of zero (Fig. B.4 in Appendix B.5), leading to early stopping. Perversely, L-BFGS stops in these cases without reaching a critical point: the gradient norm is around 10^{-2} or 10^{-3} (see the bottom row of Fig. 3.4). The gradient still contains useful information for improving the loss.

3.7.2 NysNewton-CG (NNCG)

We can avoid premature termination by using a damped version of Newton's method with *Armijo line search*. The Armijo conditions use only a subset of the strong Wolfe conditions. Under only Armijo conditions, L-BFGS is unstable; we require a different approximation to the Hessian ($p \times p$ for a neural net with p parameters) that does not require storing ($\mathcal{O}(p^2)$) or inverting ($\mathcal{O}(p^3)$) the Hessian. Instead, we run a Newton-CG algorithm that solves for the Newton step using preconditioned conjugate gradient (PCG). This algorithm can be implemented efficiently with Hessian-vector products. These can be computed $\mathcal{O}\left((n_{\text{res}} + n_{\text{bc}})p\right)$ time [Pearlmutter, 1994]. Section 3.5 shows that the Hessian is ill-conditioned with fast spectral decay, so CG without preconditioning will converge slowly. Hence we use NyströmPCG, a PCG method that is designed to solve linear systems with fast spectral decay [Frangella et al., 2023]. The resulting algorithm is called NysNewton-CG (abbreviated NNCG); a full description of the algorithm appears in Appendix B.5.

3.7.3 Performance of NNCG

Fig. 3.4 shows that NNCG significantly improves both the loss and gradient norm of the solution when applied after Adam+L-BFGS, while Fig. 3.5 visualizes how NNCG improves the absolute error (pointwise) of the PINN solution when applied after Adam+L-BFGS. Furthermore, Table 3.2 shows that NNCG also improves the L2RE of the PINN solution. In contrast, applying gradient descent (GD) after Adam+L-BFGS improves neither the loss nor the L2RE. This result is unsurprising, as our theory predicts that NNCG will work better than GD for an ill-conditioned loss (Section 3.8).

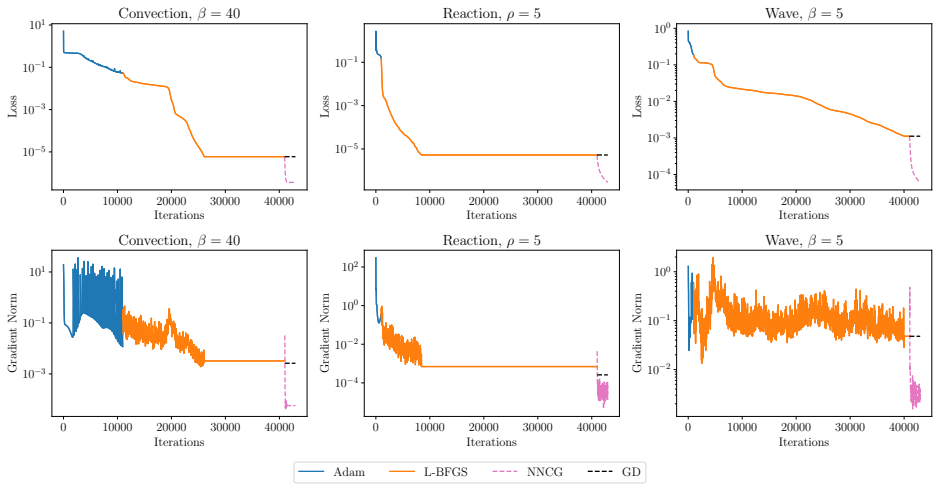


Figure 3.4: Performance of NNCG and GD after Adam+L-BFGS. (Top) NNCG reduces the loss by a factor greater than 10 in all instances, while GD fails to make progress. (Bottom) Furthermore, NNCG significantly reduces the gradient norm on the convection and wave problems, while GD fails to do so.

3.7.4 Why Not Use NNCG Directly After Adam?

Since NNCG improves the PINN solution and uses simpler line search conditions than L-BFGS, it is tempting to replace L-BFGS with NNCG entirely. However, NNCG is slower than L-BFGS: the L-BFGS update can be computed in $\mathcal{O}(mp)$ time, where m is the memory parameter, while just a single Hessian-vector product for computing the NNCG update requires $\mathcal{O}\left((n_{\text{res}} + n_{\text{bc}})p\right)$ time. Table B.1 shows NNCG takes 5, 20, and 322 more times per-iteration as L-BFGS on convection, reaction, and wave respectively. Consequently, we should run Adam+L-BFGS to make as much progress as possible before switching to NNCG.

3.8 Theory

We relate the conditioning of the differential operator to the conditioning of the PINN loss function (3.2) in Theorem 3.4. When the differential operator is ill-conditioned, gradient descent takes many iterations to reach a high-precision solution. As a result, first-order methods alone may not deliver sufficient accuracy.

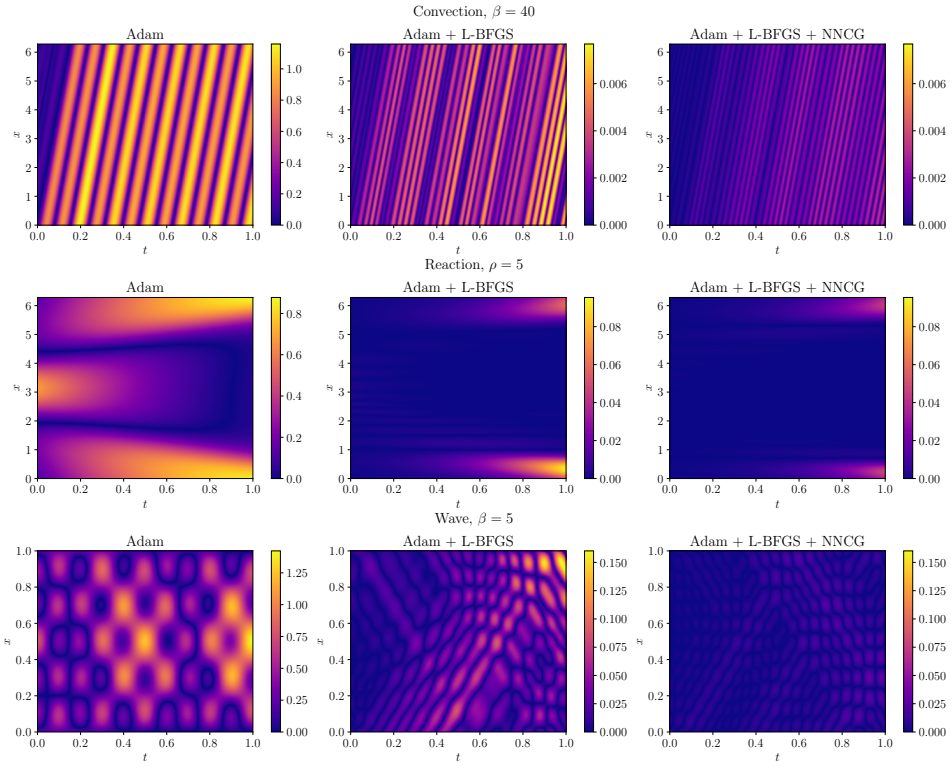


Figure 3.5: Absolute errors of the PINN solution at optimizer switch points. The first column shows errors after Adam, the second column shows errors after running L-BFGS following Adam, and the third column shows the errors after running NNCG following Adam+L-BFGS. L-BFGS improves the solution obtained from first running Adam, and NNCG further improves the solution even after Adam+L-BFGS stops making progress. Note that Adam solution errors (left-most column) are presented at separate scales as these solutions are far off from the exact solutions.

Algorithm 2 Gradient-Damped Newton Descent (GDND)

Require: # of gradient descent iterations K_{GD}, gradient descent learning rate η_{GD}, # of damped Newton iterations K_{DN}, damped Newton learning rate η_{DN}, damping parameter γ

Phase I: **Gradient descent**

for $k = 0, \ldots, K_{\mathrm{GD}} - 1$ **do**

$w_{k+1} = w_k - \eta_{\mathrm{GD}} \nabla L(w_k)$

end for

Phase II: **Damped Newton**

Set $\tilde{w}_0 = w_{K_{\mathrm{GD}}}$

for $k = 0, \ldots, K_{\mathrm{DN}} - 1$ **do**

$\tilde{w}_{k+1} = \tilde{w}_k - \eta_{\mathrm{DN}} \left(H_L(\tilde{w}_k) + \gamma I\right)^{-1} \nabla L(\tilde{w}_k)$

end for

Ensure: approximate solution $\tilde{w}_{K_{\mathrm{DN}}}$

To address this issue, we develop and analyze a hybrid algorithm, Gradient Damped Newton Descent (GDND, Algorithm 2), that switches from gradient descent to damped Newton's method after a fixed number of iterations. We show that GDND gives fast linear convergence independent of the condition number. This theory supports our empirical results, which show that the best performance is obtained by running Adam and switching to L-BFGS. Moreover, it provides a theoretical basis for using Adam+L-BFGS+NNCG to achieve the best performance.

GDND differs from Adam+L-BFGS+NNCG, the algorithm we recommend in practice. We analyze GD instead of Adam because existing analyses of Adam [Défossez et al., 2022, Zhang et al., 2022] do not mirror its empirical performance. The reason we run both L-BFGS and damped Newton is to maximize computational efficiency (Section 3.7.4).

3.8.1 Preliminaries

We begin with the main assumption for our analysis.

Assumption 3.1 (Interpolation)**.** Let $\mathcal{W}_\star$ denote the set of minimizers of (3.2). We assume that

$$L(w_\star) = 0, \quad \text{for all } w_\star \in \mathcal{W}_\star,$$

i.e., the model perfectly fits the training data.

From a theoretical standpoint, Theorem 3.1 is natural in light of various universal approximation theorems [Cybenko, 1989, Hornik et al., 1990, De Ryck et al., 2021], which show neural networks are capable of approximating any continuous function to arbitrary accuracy. Moreover, interpolation in neural networks is common in practice [Zhang et al., 2021, Belkin, 2021].

PŁ*-condition. In modern neural network optimization, the PŁ*-condition [Liu et al., 2022, 2023] is key to showing convergence of gradient-based optimizers. It is a local version of the celebrated Polyak-Łojasiewicz condition [Polyak, 1963, Karimi et al., 2016], specialized to interpolation.

Definition 3.2 (PŁ*-condition)**.** Suppose L satisfies Theorem 3.1. Let $\mathcal{S} \subset \mathbb{R}^p$. Then L is μ-PŁ*in $\mathcal{S}$ if

$$\frac{\|\nabla L(w)\|^2}{2\mu} \geq L(w), \qquad \forall w \in \mathcal{S}.$$

The PŁ*-condition relates the gradient norm to the loss and implies that any minimizer in $\mathcal{S}$ is a global minimizer. Importantly, the PŁ*-condition can hold for non-convex losses and is known to hold, with high probability, for sufficiently wide neural nets with the least-squares loss [Liu et al., 2022].

Definition 3.3 (Condition number for PŁ* loss functions)**.** Let $\mathcal{S}$ be a set for which L is μ-PŁ*. Then the condition number of L over $\mathcal{S}$ is given by

$$\kappa_L(\mathcal{S}) = \frac{\sup_{w\in\mathcal{S}} \|H_L(w)\|}{\mu},$$

where $H_L(w)$ is the Hessian matrix of the loss function.

Gradient descent over $\mathcal{S}$ converges to ϵ-suboptimality in $\mathcal{O}\left(\kappa_L(\mathcal{S}) \log\left(\frac{1}{\epsilon}\right)\right)$ iterations [Liu et al., 2022].

3.8.2 Ill-Conditioned Differential Operators Lead to Challenging Optimization

Here, we show that when the differential operator defining the PDE is *linear* and ill-conditioned, the condition number of the PINN objective (in the sense of Theorem 3.3) is large. Our analysis in this regard is inspired by the recent work of De Ryck et al. [2023], who prove a similar result for the population PINN residual loss. However, De Ryck et al. [2023]'s analysis is based on the *lazy training regime*, which assumes the NTK is approximately constant. This regime does not accurately capture the behavior of practical neural networks [Allen-Zhu and Li, 2019, Chizat et al., 2019, Ghorbani et al., 2020, 2021]. Moreover, gradient descent can converge even with a non-constant NTK [Liu et al., 2020]. Our theoretical result is more closely aligned with deep learning practice as it does not assume lazy training and pertains to the empirical loss rather than the population loss.

Theorem 3.4 provides an informal version of our result in Appendix B.6 that shows that ill-conditioned differential operators induce ill-conditioning in the loss (3.2). The theorem statement involves a kernel integral operator, $\mathcal{K}_\infty$ (defined in (B.4) in Appendix B.6), evaluated at the optimum $w_\star$.

Theorem 3.4 (Informal)**.** *Suppose Theorem 3.1 holds and $p \geq n_{\text{res}} + n_{\text{bc}}$. Fix $w_\star \in \mathcal{W}_\star$ and set $\mathcal{A} = \mathcal{D}^*\mathcal{D}$. For some $\alpha > 1/2$, suppose the eigenvalues of $\mathcal{A} \circ \mathcal{K}_\infty(w_\star)$ satisfy $\lambda_j(\mathcal{A} \circ \mathcal{K}_\infty(w_\star)) = \mathcal{O}\left(j^{-2\alpha}\right)$. If $\sqrt{n_{\text{res}}} = \Omega\left(\log\left(\frac{1}{\delta}\right)\right)$, then for any set $\mathcal{S}$ that contains $w_\star$ and for which L is μ-PŁ*,*

$$\kappa_L(\mathcal{S}) = \Omega\left(n_{\text{res}}^{\alpha}\right), \qquad \textit{with probability} \geq 1 - \delta.$$

Theorem 3.4 relates the conditioning of the PINN optimization problem to the conditioning of the operator $\mathcal{A} \circ \mathcal{K}_\infty(w_\star)$, where $\mathcal{A}$ is the Hermitian square of $\mathcal{D}$. If the spectrum of $\mathcal{A} \circ \mathcal{K}_\infty(w_\star)$ decays polynomially, then, with high probability, the condition number grows with n_{res}. As n_{res} typically ranges from 10^3 to 10^4, Theorem 3.4 shows the condition number of the PINN problem is generally large, and so first-order methods will be slow to converge to the optimum. Fig. B.5 in Appendix B.6.5 empirically verifies the claim of Theorem 3.4 for the convection equation.

3.8.3 Efficient High-Precision Solutions via GDND

We now analyze the convergence behavior of Algorithm 2. Theorem 3.5 provides an informal version of our result in Appendix B.7.

Theorem 3.5 (Informal)**.** *Suppose $L(w)$ satisfies the μ-PŁ*-condition in a certain ball about w_0. Then there exists $\eta_{\text{GD}} > 0$ and $K_{\text{GD}} < \infty$ such that Phase I of Algorithm 2 outputs a point $w_{K_{\text{GD}}}$, for which Phase II of Algorithm 2 with $\eta_{\text{DN}} = 5/6$ and appropriate damping $\gamma > 0$, satisfies*

$$L(\tilde{w}_k) \leq \left(\frac{2}{3}\right)^k L(w_{K_{\text{GD}}}).$$

Hence after $K_{\text{DN}} \geq 3 \log\left(\frac{L(w_{K_{\text{GD}}})}{\epsilon}\right)$ iterations, Phase II of Algorithm 2 outputs a point satisfying $L(\tilde{w}_{K_{\text{DN}}}) \leq \epsilon$.

Theorem 3.5 shows only a fixed number of gradient descent iterations are needed before Algorithm 2 can switch to damped Newton's method and enjoy linear convergence independent of the condition number. As the convergence rate of Phase II with damped Newton is independent of the condition number, Algorithm 2 produces a highly accurate solution to (3.2).

Note that Theorem 3.5 is local; Algorithm 2 must find a point sufficiently close to a minimizer with gradient descent before switching to damped Newton's method and achieving rapid convergence. It is not possible to develop a second-order method with a fast rate that does not require a good initialization, as in the worst-case, global convergence of second-order methods may fail to improve over first-order methods [Cartis et al., 2010, Arjevani et al., 2019]. Moreover, Theorem 3.5 is consistent with our experiments, which show L-BFGS is inferior to Adam+L-BFGS.

3.9 Conclusion

In this work, we explore the challenges posed by the loss landscape of PINNs for gradient-based optimizers. We demonstrate ill-conditioning in the PINN loss and show it hinders effective training of PINNs. By comparing Adam, L-BFGS, and Adam+L-BFGS, and introducing NNCG, we have demonstrated several approaches to improve the training process. Our theory supports our experimental findings: we connect ill-conditioned differential operators to ill-conditioning in the PINN loss

and prove the benefits of second-order methods over first-order methods for PINNs.

Chapter 4

GPU-Enabled Large-Scale Optimization Using Randomized Linear Algebra

This chapter caps a line of work on randomized numerical linear algebra (RandNLA) for large-scale optimization, including NyströmPCG [Frangella et al., 2023], NysADMM [Zhao et al., 2022, Diamandis et al., 2026], SketchySGD [Frangella et al., 2024b], PROMISE [Frangella et al., 2024a], and SAPPHIRE [Sun et al., 2025]. We develop `rlaopt`, an open-source, PyTorch-based package that makes RandNLA-based optimization accessible to practitioners. `rlaopt` includes GPU-enabled implementations of NyströmPCG, NysADMM, and SAPPHIRE, a CVXPY-inspired modeling language for specifying optimization problems, and support for differentiating through the solver for end-to-end learning; it delivers GPU speedups of $5-30\times$ over CPU implementations on benchmark problems.

This chapter is based on a manuscript in preparation for ACM Transactions on Mathematical Software.

4.1 Introduction

Large-scale optimization lies at the heart of modern machine learning and scientific computing. Datasets routinely contain millions of samples and features, giving rise to optimization problems whose sheer scale demands efficient algorithms and hardware-aware implementations. Over the past two decades, randomized numerical linear algebra (RandNLA) has emerged as a powerful toolkit for addressing these challenges, producing algorithms that exploit low-rank structure and stochastic approximations to dramatically reduce computational costs [Halko et al., 2011, Mahoney, 2011,

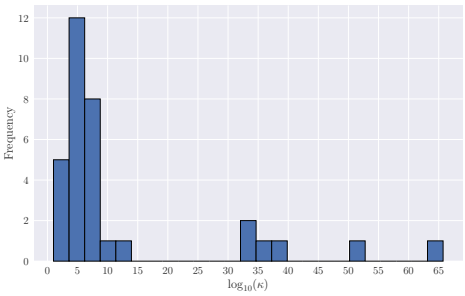
Woodruff, 2014, Martinsson and Tropp, 2020]. However, a significant gap remains between the strong algorithmic foundations of RandNLA and the software available to practitioners: existing implementations of RandNLA-based methods lack the ability to leverage modern parallel hardware such as GPUs, and do not provide a simple, user-friendly syntax for modeling optimization problems. Consequently, practitioners who wish to apply RandNLA-based algorithms to large-scale problems must piece together ad hoc implementations, limiting both accessibility and performance.

We now describe two important problem classes where RandNLA-based algorithms offer significant advantages, and where the lack of high-quality implementations is particularly acute.

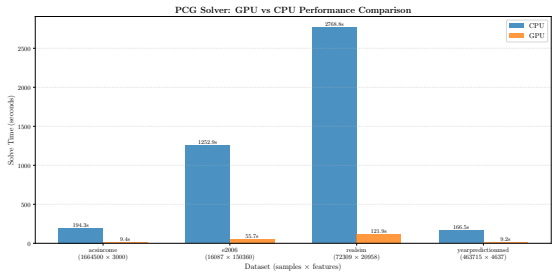
Large-scale linear systems. Dense linear systems of the form $(A + \lambda I)x = b$ arise in kernel ridge regression [Schölkopf and Smola, 2002], Gaussian process inference [Rasmussen and Williams, 2005, Gardner et al., 2018], and other settings throughout machine learning and scientific computing. Direct methods such as Cholesky decomposition require $\mathcal{O}(n^3)$ computation and $\mathcal{O}(n^2)$ storage, limiting them to problems with $n \sim 10^4$. Iterative methods such as conjugate gradient scale more favorably, with per-iteration complexity $\mathcal{O}(n^2)$, but converge slowly when the problem is ill-conditioned—a common occurrence in practice. RandNLA-based preconditioning, in particular the randomized Nyström preconditioner [Frangella et al., 2023], dramatically accelerates convergence by constructing high-quality low-rank approximations of the kernel matrix at modest cost. The resulting preconditioned conjugate gradient (NyströmPCG) method combines the scalability of iterative methods with robustness to ill-conditioning.

Large-scale optimization. Classical optimization methods, such as interior-point methods, produce high-accuracy solutions but rely on expensive matrix factorizations that limit their applicability to large-scale problems [O'Donoghue et al., 2016, Stellato et al., 2020, Applegate et al., 2021]. On the other end of the spectrum, first-order methods such as stochastic gradient descent (SGD) and its variants [Robbins and Monro, 1951, Johnson and Zhang, 2013, Defazio et al., 2014, Allen-Zhu, 2018] scale to massive datasets but suffer from slow convergence on ill-conditioned problems and sensitivity to hyperparameters such as the learning rate [Nemirovski et al., 2009]. Operator splitting frameworks such as the alternating direction method of multipliers (ADMM) [Boyd et al., 2011] naturally handle constraints and nonsmooth regularizers, but also require the solution of subproblems that can be ill-conditioned. RandNLA techniques integrate naturally with both stochastic gradient methods and operator splitting: Nyström-based preconditioning accelerates ADMM by improving the conditioning of linear system subproblems [Zhao et al., 2022, Diamandis et al., 2026], and randomized curvature estimates yield preconditioned stochastic gradient methods with reliable default hyperparameters and fast convergence [Frangella et al., 2024b,a, Sun et al., 2025]. Crucially, the dominant operations in these RandNLA-enhanced methods are matrix-matrix and matrix-vector products, which are highly amenable to GPU acceleration.

Fig. 4.1 illustrates the opportunity: machine learning datasets are commonly ill-conditioned



(a) Machine learning datasets are often ill-conditioned, with condition numbers ranging from 10^4 to 10^8 or larger.



(b) NyströmPCG on GPU obtains a 20× speedup over CPU.

Figure 4.1: Ill-conditioning is prevalent in machine learning datasets (left), and GPU acceleration yields dramatic speedups for RandNLA-based solvers (right).

(Fig. 4.1a), and GPU acceleration yields dramatic speedups for RandNLA-based solvers such as NyströmPCG (Fig. 4.1b).

To address the gap between RandNLA algorithms and practical software, we develop `rlaopt`, a PyTorch-based package for large-scale optimization and scientific computing. `rlaopt` includes GPU-enabled implementations of NyströmPCG for large-scale linear systems, NysADMM [Zhao et al., 2022] for constrained convex optimization, and SAPPHIRE [Frangella et al., 2024a, Sun et al., 2025] for empirical risk minimization. To make these solvers accessible, we create a flexible modeling language inspired by disciplined convex programming [Grant et al., 2006] and CVXPY [Diamond and Boyd, 2016] that allows users to specify optimization problems using natural mathematical syntax. `rlaopt` also supports differentiating through the solver, making it suitable for modern machine learning pipelines that require end-to-end gradient computation. Our implementation is open-sourced under an Apache license and is available at https://github.com/udellgroup/rlaopt. Documentation can be found at https://rlaopt.readthedocs.io/en/latest/, and a PyPI release is forthcoming.

4.1.1 Contributions

Our contributions are as follows:

1. We develop `rlaopt`, an open-source, PyTorch-based software package for large-scale optimization using randomized linear algebra. `rlaopt` supports both CPU and GPU execution, enabling practitioners to leverage hardware accelerators with minimal code changes.

2. We create a flexible modeling language, inspired by disciplined convex programming [Grant et al., 2006] and CVXPY [Diamond and Boyd, 2016], that allows users to specify optimization

problems via natural mathematical syntax. This modeling language supports a wide range of objectives, constraints, and regularizers.

3. We implement a suite of RandNLA-based algorithms within `rlaopt`, including NyströmPCG for large-scale linear systems, NysADMM for constrained convex optimization via operator splitting, and SAPPHIRE for preconditioned, stochastic variance-reduced optimization.

4. We demonstrate significant GPU speedups over CPU implementations across all algorithm families: up to 20× for NyströmPCG, 20–30× for NysADMM, and 5–10× for SAPPHIRE on benchmark problems.

5. We show that `rlaopt` supports differentiating through the solver, enabling applications such as hyperparameter tuning and end-to-end learning within modern machine learning pipelines.

4.1.2 Roadmap

Section 4.2 formally defines the problem classes addressed by `rlaopt`. Section 4.3 surveys related work organized by problem class. Section 4.4 describes the modeling language and demonstrates its flexibility through examples. Section 4.5 explains how `rlaopt` automatically detects problem structure to apply appropriate solvers. Section 4.6 presents performance benchmarks comparing CPU and GPU implementations across problem classes. Section 4.7 concludes the paper.

4.2 Problem Classes

In this section, we formally define the problem classes addressed by `rlaopt`. We consider two broad classes: positive definite linear systems (Section 4.2.1) and empirical risk minimization with constraints and regularizers (Section 4.2.2).

4.2.1 Positive-Definite Linear Systems

The first problem class consists of symmetric positive-definite (pd) linear systems of the form

$$Ax = b, \tag{4.1}$$

where $A \in \mathbb{R}^{n\times n}$ is symmetric and pd, $b \in \mathbb{R}^n$, and $x \in \mathbb{R}^n$ is the unknown. A common special case is the regularized linear system

$$(K + \lambda I)x = b, \tag{4.2}$$

where $K \in \mathbb{R}^{n\times n}$ is symmetric positive-semidefinite (psd) and $\lambda > 0$ is a regularization parameter.

Linear systems of this form arise frequently in machine learning and scientific computing. A prominent example is kernel ridge regression (KRR), in which one solves

$$\underset{w\in\mathbb{R}^n}{\text{minimize}} \quad \frac{1}{2}\|Kw-y\|^2+\frac{\lambda}{2}\|w\|_K^2, \tag{4.3}$$

where $K\in\mathbb{R}^{n\times n}$ is a kernel matrix with entries $K_{ij}=k(x_i,x_j)$ for a kernel function k, and $y\in\mathbb{R}^n$ is the target vector. The optimality conditions of (4.3) yield the linear system $(K+\lambda I)w^\star=y$. Linear systems of the form (4.2) also arise in Gaussian process (GP) inference [Rasmussen and Williams, 2005], where computing the posterior mean and variance requires solving dense linear systems involving a kernel matrix.

Direct methods such as Cholesky decomposition solve (4.1) in $\mathcal{O}(n^3)$ time with $\mathcal{O}(n^2)$ storage, limiting them to problems with $n\lesssim 10^4$. Iterative methods such as conjugate gradient (CG) scale more favorably, with per-iteration complexity $\mathcal{O}(n^2)$, but converge slowly when A is ill-conditioned. Ill-conditioning is common in practice: kernel matrices in machine learning often have rapidly decaying spectra, leading to large condition numbers [Caponnetto and DeVito, 2007, Bach, 2013, Tu et al., 2016, Ma and Belkin, 2017, Belkin, 2018]. `rlaopt` addresses this challenge by providing NyströmPCG, which uses a randomized Nyström preconditioner [Frangella et al., 2023] to dramatically accelerate the convergence of CG on ill-conditioned systems.

4.2.2 Empirical Risk Minimization with Constraints and Regularizers

The second problem class is composite convex optimization of the form

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad f(x)+\sum_{i=1}^{k} g_i(A_ix-b_i), \tag{4.4}$$

where $f\colon\mathbb{R}^n\to\mathbb{R}$ is smooth and convex, each $g_i\colon\mathbb{R}^{m_i}\to\mathbb{R}\cup\{+\infty\}$ is closed, convex, and proxable (i.e., its proximal operator can be evaluated efficiently), $A_i\in\mathbb{R}^{m_i\times n}$, and $b_i\in\mathbb{R}^{m_i}$. This formulation is flexible enough to encode a wide range of machine learning problems, including empirical risk minimization (ERM) with constraints and regularizers.

In a typical ERM setting, the smooth component f takes the form

$$f(x)=\frac{1}{N}\sum_{j=1}^{N}\ell_j(a_j^\top x), \tag{4.5}$$

where $\{(a_j,y_j)\}_{j=1}^N$ is a training set with $a_j\in\mathbb{R}^n$ and $y_j\in\mathbb{R}$, and ℓ_j is a loss function. The nonsmooth terms g_i encode regularizers and constraints such as ℓ_1 regularization, box constraints, elastic net penalties, and indicator functions of convex sets.

Several concrete problems fit naturally into this framework:

- **ℓ_1-regularized logistic regression:** f is the average logistic loss and $g(x) = \mu\|x\|_1$ for some regularization weight $\mu > 0$.
- **Bounded elastic net:** $f(w) = \frac{1}{2N}\|Xw-y\|_2^2$, with $g_1(w) = \lambda_1\|w\|_1 + \frac{\lambda_2}{2}\|w\|_2^2$ and g_2 encoding box constraints $0 \le w \le 1$.
- **Constrained multinomial regression:** f is the average cross-entropy loss for multiclass classification, with g encoding box constraints on the regression coefficients.

`rlaopt` provides several solvers for problems of the form (4.4). For problems where the full gradient of f is available, `rlaopt` supports NysADMM [Zhao et al., 2022] which combines ADMM with Nyström-based preconditioning to accelerate linear system subproblems that arise in ADMM. For large-scale ERM problems where only stochastic gradients of f are available, `rlaopt` supports SAPPHIRE [Frangella et al., 2024a, Sun et al., 2025], a family of preconditioned stochastic variance-reduced methods that uses randomized curvature estimates to achieve fast convergence with reliable default hyperparameters. `rlaopt` also supports (accelerated) proximal gradient for problems where preconditioning is not needed.

4.3 Related Work

We survey related work organized by the problem classes introduced in Section 4.2. For each class, we describe existing algorithmic approaches and software, and discuss how `rlaopt` relates to and improves upon them.

4.3.1 Large-Scale Linear Systems

Direct methods. Direct methods such as Cholesky decomposition are the standard approach for solving dense pd linear systems [Golub and Van Loan, 2013]. While they produce high-accuracy solutions, their $\mathcal{O}(n^3)$ computational cost and $\mathcal{O}(n^2)$ storage requirements render them impractical for problems with $n \gtrsim 10^4$. `rlaopt` targets the regime where direct methods are too expensive, providing iterative solvers that scale to much larger problem sizes.

Iterative methods and preconditioning. Conjugate gradient (CG) is the method of choice for large-scale pd linear systems, with per-iteration complexity $\mathcal{O}(n^2)$ for dense systems. However, CG converges slowly when the system is ill-conditioned, and effective preconditioners are essential for practical performance. Randomized Nyström preconditioning [Frangella et al., 2023] constructs a high-quality low-rank approximation of the coefficient matrix using sketching techniques from RandNLA [Halko et al., 2011, Martinsson and Tropp, 2020]. The resulting NyströmPCG method converges rapidly even on ill-conditioned problems, and its dominant cost—matrix-matrix products

for constructing the preconditioner—is highly GPU-amenable. `rlaopt` provides a GPU-enabled implementation of NyströmPCG that achieves up to 20× speedup over CPU.

Kernel methods. FALKON [Rudi et al., 2017] is a widely used solver for inducing points kernel ridge regression that uses Nyström preconditioning with CG. Gaussian process inference packages such as GPyTorch [Gardner et al., 2018] also rely on PCG-based linear solvers. `rlaopt` differs from these tools by targeting general pd linear systems rather than a specific application and providing a modeling language for specifying the problem.

4.3.2 Optimization Solvers

Interior-point methods. Interior-point methods (IPMs) are the gold standard for small-to-moderate-scale convex optimization, producing high-accuracy solutions with polynomial-time guarantees [Nesterov and Nemirovskii, 1994]. Software implementations such as Clarabel [Goulart and Chen, 2024] provide reliable IPM solvers, and recent work has extended these to GPU [Chen et al., 2025b]. However, IPMs rely on matrix factorizations whose cost grows cubically with problem size, limiting their applicability to large-scale machine learning problems. `rlaopt` takes a complementary approach, using first-order and operator splitting methods enhanced with RandNLA to handle problems at scales where IPMs are impractical.

Operator splitting methods. The alternating direction method of multipliers (ADMM) [Gabay and Mercier, 1976, Boyd et al., 2011] and related operator splitting methods [Ryu and Yin, 2022] decompose composite optimization problems into simpler subproblems that can be solved independently. ADMM is the basis for several widely used solvers, including OSQP [Stellato et al., 2020] for quadratic programs and SCS [O'Donoghue et al., 2016] for conic programs. A key bottleneck in ADMM is solving a subproblem corresponding to the primal update. NysADMM [Zhao et al., 2022] accelerates inexact ADMM by solving the primal subproblem with NyströmPCG, using a randomized Nyström preconditioner to handle ill-conditioning; for non-quadratic smooth losses, the subproblem is formed from a second-order approximation of the smooth term. GeNIOS [Diamandis et al., 2026] builds on this recipe with adaptive penalty parameter updates, periodic preconditioner refresh, an adaptive inexact PCG tolerance, ADMM over-relaxation, and primal/dual infeasibility detection, with an open-source Julia implementation. `rlaopt`'s NysADMM solver inherits these engineering choices from GeNIOS (with the exception of infeasibility detection), adapted to a PyTorch/GPU setting, and achieves 20–30× speedups over CPU for constrained optimization problems. A related line of work specializes operator splitting to linear programming: PDLP [Applegate et al., 2021] applies the primal-dual hybrid gradient method to a saddle-point formulation of LP with diagonal preconditioning, adaptive step sizes, and adaptive restarts, and cuPDLP [Lu and Yang, 2025] ports this approach to GPUs. These solvers share `rlaopt`'s philosophy of scaling on GPU through

matrix-vector products, but target LP specifically rather than the broader composite convex setting.

Proximal gradient methods. Proximal gradient methods [Parikh et al., 2014] and their accelerated variants are a natural fit for composite problems of the form (4.4) when the proximal operator of each g_i is cheap to evaluate. These methods have per-iteration costs dominated by gradient evaluations of the smooth component f, but converge slowly on ill-conditioned problems. `rlaopt` implements proximal gradient with optional Nyström-based preconditioning, which improves convergence on ill-conditioned objectives while preserving the simplicity of the proximal gradient framework.

Stochastic gradient methods. When the smooth component f has a finite-sum structure, stochastic gradient methods such as SGD [Robbins and Monro, 1951], Adam [Kingma and Ba, 2014], SVRG [Johnson and Zhang, 2013], SAGA [Defazio et al., 2014], and Katyusha [Allen-Zhu, 2018, Kovalev et al., 2020] achieve low per-iteration costs by operating on mini-batches. However, these methods are sensitive to hyperparameters and converge slowly on ill-conditioned problems [Nemirovski et al., 2009]. The PROMISE framework [Frangella et al., 2024a] and SketchySGD [Frangella et al., 2024b] address these limitations by using randomized curvature estimates as preconditioners, yielding methods with reliable default hyperparameters and fast convergence on ill-conditioned problems. SAPPHIRE [Sun et al., 2025] generalizes this line of work to proximal settings, supporting nonsmooth regularizers and constraints within the preconditioned stochastic gradient framework. `rlaopt` provides GPU-enabled implementations of SAPPHIRE methods, achieving 5–10$\times$ speedups over CPU on large-scale ERM problems.

Differentiable optimization. Several frameworks enable differentiating through the solution of optimization problems. cvxpylayers [Agrawal et al., 2019] embeds parametrized convex programs specified in CVXPY as differentiable layers, using implicit differentiation through the KKT conditions of the conic reformulation. This approach supports a broad class of disciplined convex programs but scales cubically in the cone program dimension, limiting its applicability to small-to-moderate problems. JAXopt [Blondel et al., 2022] provides a modular implicit differentiation framework in JAX with a wide range of solvers, including proximal gradient, L-BFGS, and OSQP. MPAX [Lu et al., 2024] is a JAX-based first-order solver for large-scale linear and quadratic programs that supports differentiation through unrolled solver iterations. These frameworks are complementary to `rlaopt`: they cover different problem classes or ecosystems, while `rlaopt` contributes differentiable, RandNLA-based solvers in PyTorch that are specifically designed for large-scale, ill-conditioned problems.

4.4 Modeling Language

A central design goal of `rlaopt` is to provide a simple, expressive interface for specifying optimization problems. To this end, we develop a modeling language inspired by disciplined convex programming [Grant et al., 2006] and CVXPY [Diamond and Boyd, 2016] that lets users construct objectives from composable building blocks using natural mathematical syntax. In this section, we describe the core abstractions of the modeling language and demonstrate its flexibility through examples.

4.4.1 Core Abstractions

The `rlaopt` modeling language is built around three core abstractions: *variables*, *atoms*, and *solvers*.

Variables. A `Variable` represents an unknown quantity to be optimized. Variables are created by specifying their shape and, optionally, a name and a device (CPU or GPU):

```
w = Variable((n,), name="w")
beta = Variable((p, K), name="beta", device="cuda")
```

Variables can appear in mathematical expressions involving matrix multiplication, addition, and subtraction, using standard Python operators. For instance, `X @ w + b - y` represents the affine expression $Xw + b - y$.

Atoms. An *atom* is a function with known mathematical properties (e.g., smoothness, or the availability of a proximal operator) that serves as a building block for constructing objectives. `rlaopt` provides atoms for common losses, regularizers, and constraints encountered in machine learning and scientific computing. Table 4.1 summarizes the available atoms.

Composing objectives. Objectives are constructed by combining atoms with the `+` operator, mirroring the mathematical structure of the problem. Scalar multiplication via `*` is also supported. For example, the bounded elastic net problem

$$\underset{w,b}{\text{minimize}} \quad \frac{1}{2N}\|Xw + b - y\|_2^2 + \lambda_1\|w\|_1 + \frac{\lambda_2}{2}\|w\|_2^2 \quad \text{subject to} \quad 0 \le w \le 1$$

is specified as:

```
w = Variable((X.shape[1],))
b = Variable((1,))
obj = SumSquares(X @ w + b - y) * (0.5 / N) \
```

Category	Atom	Mathematical form
General	`SumSquares(expr)`	$\|\text{expr}\|_2^2$
Linear model losses	`LinearRegression(w, loader)`	$\frac{1}{N}\sum_j (y_j - a_j^\top w)^2$
	`LogisticRegression(w, loader)`	$\frac{1}{N}\sum_j \log(1 + e^{-y_j a_j^\top w})$
	`MultinomialRegression(w, loader)`	$-\frac{1}{N}\sum_j \log(\text{softmax}(Xw)_{j,c_j})$
	`PoissonRegression(w, loader)`	Poisson negative log-likelihood
	`GammaRegression(w, loader)`	Gamma negative log-likelihood
	`InverseGaussianRegression(w, loader)`	Inverse Gaussian negative log-likelihood
	`HuberRegression(w, loader, δ)`	Huber loss with threshold δ
Regularizers	`L1Norm(w, λ)`	$\lambda\|w\|_1$
	`L2Norm(w, λ)`	$\lambda\|w\|_2$
	`NucNorm(W, λ)`	$\lambda\|W\|_*$ (nuclear norm)
	`ElasticNet(w, λ1, λ2)`	$\lambda_1\|w\|_1 + \frac{\lambda_2}{2}\|w\|_2^2$
Constraints	`Box(w, lower, upper)`	$\mathcal{I}[\text{lower} \le w \le \text{upper}]$
	`NonNegative(w)`	$\mathcal{I}[w \ge 0]$
	`Halfspace(w, c, upper)`	$\mathcal{I}[c^\top w \le \text{upper}]$
	`LinearEquality(w, A, b)`	$\mathcal{I}[Aw = b]$
	`Polyhedron(w, A, b, C, l, u)`	$\mathcal{I}[Aw = b,\ l \le Cw \le u]$
	`L1NormBall(w, r)`	$\mathcal{I}[\|w\|_1 \le r]$
	`L2NormBall(w, r)`	$\mathcal{I}[\|w\|_2 \le r]$
	`LInfNormBall(w, r)`	$\mathcal{I}[\|w\|_\infty \le r]$

Table 4.1: Atoms available in `rlaopt`. Linear model losses operate on data provided via a `DataLoader`. $\mathcal{I}[\cdot]$ denotes the indicator function of the given constraint set (zero when satisfied, $+\infty$ otherwise). $\|\cdot\|_*$ denotes the nuclear norm (sum of singular values).

```
    + ElasticNet(w, lam1, lam2) \
    + Box(w, 0.0, 1.0)
```

The resulting objective `obj` automatically tracks the variables and atoms that compose it.

Solvers. Once an objective has been defined, the user selects a solver and its configuration. `rlaopt` provides a unified solver interface with two modes of operation: a *stepped* mode for fine-grained control over the optimization loop, and a *direct* mode for one-call solving. A complete reference of all solver configuration parameters, termination criteria, and their defaults is provided in Appendix C.1. In stepped mode, the user initializes the solver state and calls `step` iteratively (note that the NysADMM solver is accessed via the `ADMM` class in the API):

```
solver = ADMM(obj, config=ADMMConfig())
variable_values = obj.variable_values
state = solver.init_state(variable_values)
for _ in range(num_iters):
```

```
    variable_values, state = solver.step(variable_values, state)
```

In direct mode, the user simply calls `solve`:

```
solver = ProxGrad(obj, config=ProxGradConfig(eta=0.5))
result = solver.solve()
x_sol = result.variable_values
```

Both modes return the solution as a dictionary mapping variable names to their optimal values.

Linear systems. For pd linear systems, `rlaopt` provides a separate `LinSys` interface. The user specifies a matrix, right-hand side, and regularization:

```
lin_sys = LinSys(A, b, reg=1e-2)
```

The linear system is then solved with NyströmPCG:

```
precond_config = NystromConfig(rank_init=50)
pcg_config = PCGConfig(preconditioner_config=precond_config)
solver = PCG(lin_sys, config=pcg_config)

params = lin_sys.w
state = solver.init_state(params)
for _ in range(100):
    params, state = solver.step(params, state)
```

4.4.2 Differentiating Through the Solver

A key feature of `rlaopt` is its native support for differentiating through the optimization solver. In modern machine learning pipelines, users are often interested in optimizing with respect to a parameter of the optimization problem itself—for example, tuning a regularization parameter to minimize validation error, or learning loss function parameters in an end-to-end pipeline [Agrawal et al., 2019, Blondel et al., 2022, Lu et al., 2024]. Formally, consider a parametrized optimization problem $x^\star(\theta) = \operatorname{argmin}_x \phi(x;\theta)$, where $\theta \in \mathbb{R}^d$ is an external parameter. A common use case is hyperparameter tuning, in which one seeks to minimize an outer objective that depends on the

solution of the inner problem:

$$\underset{\theta \in \mathbb{R}^d}{\text{minimize}} \quad \mathcal{L}(x^\star(\theta)), \tag{4.6}$$

where $\mathcal{L}$ is a validation loss or other performance metric. For example, one may wish to tune the regularization parameter μ in a lasso problem to minimize the prediction error on a held-out validation set:

$$\underset{\mu>0}{\text{minimize}} \quad \frac{1}{N_{\text{val}}}\|X_{\text{val}}\, x^\star(\mu) - y_{\text{val}}\|_2^2, \quad \text{where} \quad x^\star(\mu) = \underset{x}{\text{argmin}}\ \frac{1}{N_{\text{train}}}\|X_{\text{train}}x - y_{\text{train}}\|_2^2 + \mu\|x\|_1. \tag{4.7}$$

The solvers in `rlaopt` are implemented in a functional manner and support automatic differentiation by backpropagation through the optimization trajectory. Users can embed a solver within a differentiable computation graph by setting `detach=False` in the solver configuration. For the lasso tuning problem (4.7), this is expressed as:

```
def outer_objective(mu):
    x = Variable((p,), name="x")
    train_obj = SumSquares(A_train @ x - b_train) / N_train \
              + L1Norm(x, mu)
    solver = ProxGrad(train_obj, config, detach=False)
    result = solver.solve()
    x_sol = result.variable_values["x"]
    return torch.linalg.norm(A_val @ x_sol - b_val) ** 2 / N_val

mu = torch.tensor(1.0, requires_grad=True)
for _ in range(200):
    grad, value = torch.func.grad_and_value(outer_objective)(mu)
    mu = mu - eta * grad
```

PyTorch's automatic differentiation propagates gradients through the solver iterations, yielding the derivative of the validation loss with respect to μ without requiring the user to implement custom backward passes.

4.4.3 Comparison to Disciplined Convex Programming and CVXPY

The `rlaopt` modeling language shares disciplined convex programming and CVXPY's philosophy of composable atoms and natural mathematical syntax, but differs in a key respect. Both disciplined convex programming and CVXPY transform problems into a standard conic form before passing them to a backend solver [Grant et al., 2006, Diamond and Boyd, 2016], which can significantly increase the problem size and obscure the structure of the original problem. In contrast, `rlaopt`

preserves the composite structure $f(x) + \sum_i g_i(A_i x - b_i)$ and works directly with gradient and proximal oracles, avoiding unnecessary reformulation. This is particularly advantageous for machine learning problems such as logistic regression, where conic reformulation introduces many additional variables [Diamandis et al., 2026].

4.5 Automatic Detection of Problem Structure

A key feature of `rlaopt` is its ability to automatically detect the structure of a user-specified optimization problem and decompose it into a form amenable to the chosen solver. This section describes how `rlaopt` performs this decomposition for proximal gradient methods and ADMM-based methods.

4.5.1 Smooth-Nonsmooth Decomposition

Recall from Section 4.2 that `rlaopt` handles composite convex optimization problems of the form

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad f(x) + \sum_{i=1}^{k} g_i(A_i x - b_i), \tag{4.8}$$

where f is smooth and convex, and each g_i is closed, convex, and proxable. When a user constructs an objective by composing atoms (as described in Section 4.4), `rlaopt` must determine which terms constitute the smooth component f and which terms constitute the nonsmooth components g_i, along with their associated linear operators A_i and offsets b_i.

Every atom in `rlaopt` declares two key properties: whether it is *smooth* (i.e., differentiable everywhere) and whether it is *proxable* (i.e., its proximal operator can be evaluated efficiently). For example, `SumSquares` is smooth, while `L1Norm` and `Box` are nonsmooth but proxable. Given a composite objective constructed via the `+` operator, `rlaopt` partitions the terms:

- All atoms with the smooth property form the smooth component f.
- All atoms without the smooth property form the nonsmooth components $g_1, \ldots, g_k$.

This partition is performed automatically and requires no input from the user.

4.5.2 Splitting for Proximal Gradient Methods

For proximal gradient and accelerated proximal gradient methods, `rlaopt` requires that each nonsmooth atom g_i acts directly on a variable (rather than on an affine expression of variables). In this case, the proximal operator of g_i can be applied directly to the variable at each iteration, yielding the standard proximal gradient update

$$x^{k+1} = \mathbf{prox}_{\eta g}\big(x^k - \eta \nabla f(x^k)\big), \tag{4.9}$$

where $\eta > 0$ is the step size and $g = \sum_{i=1}^{k} g_i$.

`rlaopt` validates two conditions for proximal gradient splitting:

1. Each nonsmooth atom must be proxable, meaning it takes a raw variable as input.
2. The nonsmooth atoms must operate on *disjoint* sets of variables, so that their proximal operators can be applied independently.

If either condition is violated (for example, if a nonsmooth atom receives an affine expression as input), `rlaopt` raises an error indicating that the problem structure is incompatible with proximal gradient methods and suggesting the use of ADMM instead.

4.5.3 Splitting for ADMM

ADMM handles a broader class of problems than proximal gradient, as it allows nonsmooth atoms to receive affine expressions of the variables as input. When a nonsmooth atom g_i acts on an affine expression $A_i x - b_i$ rather than a raw variable, `rlaopt` introduces an auxiliary variable z_i and rewrites the problem in the consensus form

$$\begin{array}{ll} \underset{x,\, z_1, \ldots, z_k}{\text{minimize}} & f(x) + \sum_{i=1}^{k} g_i(z_i) \\ \text{subject to} & A_i x - z_i = b_i, \qquad i = 1, \ldots, k. \end{array} \tag{4.10}$$

Each atom provides a `decompose` method that performs this transformation. Given a nonsmooth atom g_i with input expression $A_i x - b_i$, the decomposition produces:

1. A new auxiliary variable z_i whose shape matches the output dimension of $A_i x - b_i$.
2. A new atom $g_i(z_i)$ that is proxable (since it now acts on a raw variable).
3. The linear operator A_i and offset b_i, extracted from the affine expression.

The linear operator A_i is represented implicitly as a `LinearOperator` object that computes matrix-vector products $v \mapsto A_i v$ and adjoint products $v \mapsto A_i^\top v$ without forming A_i explicitly. This is important for scalability, as the linear operators arising from data matrices in machine learning can be very large.

The ADMM algorithm then alternates between approximately solving the x-subproblem (a smooth optimization problem involving f and the quadratic penalty terms), applying the proximal operators of g_i to update each z_i, and updating the dual variables. The x-subproblem requires solving a linear system at each iteration, which `rlaopt` accelerates using Nyström-based preconditioning [Frangella et al., 2023, Zhao et al., 2022].

Solver	Applicable when
NyströmPCG	Problem is a pd linear system $Ax = b$.
Proximal gradient	All nonsmooth atoms act on raw variables and operate on disjoint variable sets.
NysADMM	Nonsmooth atoms may act on affine expressions of variables. Automatically introduces auxiliary variables and linear constraints.
SAPPHIRE	Smooth component f has a finite-sum structure $f(x) = \frac{1}{N}\sum_j f_j(x)$. All nonsmooth atoms act on raw variables.

Table 4.2: Conditions for solver applicability in `rlaopt`.

4.5.4 Solver Selection

Given a composite objective, the choice of solver depends on the problem structure. Table 4.2 summarizes the conditions under which each solver family is applicable.

In practice, when the user specifies a solver, `rlaopt` validates that the problem structure is compatible and raises an informative error if it is not. This design gives users explicit control over the solver while preventing misuse through automatic structural checks.

4.5.5 Example: Splitting in Action

We illustrate the splitting process with a concrete example. Consider the ℓ_1-regularized least-squares problem where an affine transformation of the decision variable is penalized:

$$\underset{w}{\text{minimize}} \quad \frac{1}{2N}\|Xw - y\|_2^2 + \lambda\|Cw - d\|_1, \tag{4.11}$$

where $C \in \mathbb{R}^{m\times n}$ and $d \in \mathbb{R}^m$. The user specifies this problem as:

```
w = Variable((n,))
obj = SumSquares(X @ w - y) * (0.5 / N) + L1Norm(C @ w - d, lam)
```

For a proximal gradient solver, this problem is *not* directly compatible, because the `L1Norm` atom receives the affine expression `C @ w - d` rather than a raw variable. For ADMM, `rlaopt` automatically detects this structure and decomposes the problem. The `L1Norm` atom's `decompose` method introduces an auxiliary variable $z \in \mathbb{R}^m$ and rewrites the problem as

$$\begin{aligned} \underset{w,\, z}{\text{minimize}} \quad & \tfrac{1}{2N}\|Xw - y\|_2^2 + \lambda\|z\|_1 \\ \text{subject to} \quad & Cw - z = d. \end{aligned}$$

The linear operator C and offset d are extracted from the affine expression, and the proximal operator of $\lambda\|\cdot\|_1$ (soft-thresholding) is applied to z at each ADMM iteration. The w-subproblem involves minimizing $\frac{1}{2N}\|Xw - y\|_2^2 + \frac{\rho}{2}\|Cw - z^k - d + u^k\|_2^2$, which amounts to solving a pd linear system that `rlaopt` accelerates with Nyström preconditioning.

4.6 Preliminary Experiments

We present a preliminary evaluation of `rlaopt` across the problem classes introduced in Section 4.2. For each example, we show the mathematical formulation, the `rlaopt` code, and the convergence behavior. We also demonstrate GPU speedups across all solver families. These experiments serve as an initial demonstration of `rlaopt`'s capabilities; a more comprehensive evaluation with additional datasets, baselines, and problem instances is ongoing. Experiments are run on a dual-socket AMD EPYC 7763 system (2×64 cores, 256 threads, 3.53 GHz max boost, 512 MiB L3 cache) with 1 TiB of RAM and NVIDIA RTX A6000 GPUs (48 GB). CPU experiments are limited to 16 cores due to policies on our computing cluster. All experiments use PyTorch 2.9.1 with CUDA 12.5.

4.6.1 Large-Scale Ridge Regression with NyströmPCG

Our first example is large-scale ridge regression, which reduces to solving the pd linear system

$$(X^\top X + \lambda I)w = X^\top y,$$

where $X \in \mathbb{R}^{N\times p}$ is the data matrix, $y \in \mathbb{R}^N$ is the target vector, and $\lambda > 0$ is the regularization parameter. In `rlaopt`, this problem is specified and solved as follows:

```
# Define the linear system
reg = 1e-2
b = X.T @ y
lin_sys = LinSys(X.T @ X, b, reg)

# Configure and run NystromPCG
precond_config = NystromConfig(rank_init=50, base_damping=0.0)
pcg_config = PCGConfig(preconditioner_config=precond_config)
solver = PCG(lin_sys, config=pcg_config)

params = lin_sys.w
state = solver.init_state(params)
for _ in range(100):
```

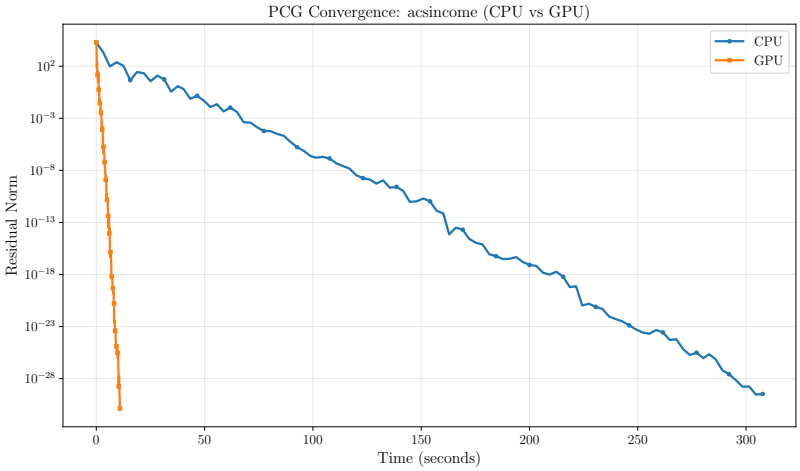


Figure 4.2: NyströmPCG convergence on ridge regression with the acsincome dataset.

```
    params, state = solver.step(params, state)
```

Fig. 4.2 shows the convergence of NyströmPCG on the acsincome dataset, which has size $N =$ 1,664,500 and $p = 3000$. NyströmPCG converges rapidly due to the randomized preconditioner, which effectively addresses the ill-conditioning of the Gram matrix $X^\top X$.

4.6.2 Multinomial Regression with Box Constraints via SAPPHIRE

Our second example is constrained multinomial logistic regression:

$$\begin{aligned} &\underset{\beta \in \mathbb{R}^{p \times K}}{\text{minimize}} && -\tfrac{1}{N} \textstyle\sum_{i=1}^{N} \log(\text{softmax}(X\beta)_{i,c_i}), \quad c_i \in \{0, 1, \ldots, K-1\}, \\ &\text{subject to} && -1 \leq \beta \leq 1, \end{aligned}$$

where $X \in \mathbb{R}^{N \times p}$ is the feature matrix and c_i is the class label for sample i. In `rlaopt`, this is expressed as:

```
# Define the objective
beta = Variable((X.shape[1], 10), name="beta", device=device)
dataset = Dataset(X, y, device=device)
loader = DataLoader(dataset, batch_size=4096)

model = MultinomialRegression(beta, loader, fit_intercept=False)
obj = model + Box(beta, lower=-1.0, upper=1.0)
```

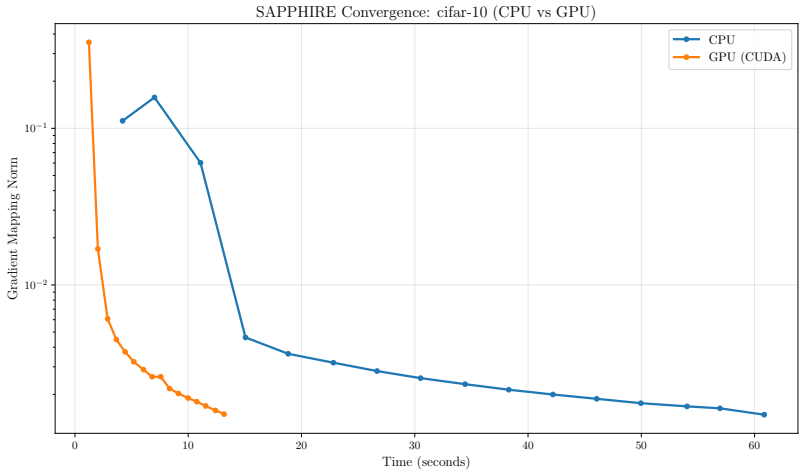


Figure 4.3: SAPPHIRE convergence on constrained multinomial regression with the CIFAR-10 dataset.

```
# Configure and run SAPPHIRE
solver = Sapphire(obj, config=SapphireConfig(eta=0.1, base_method="svrg"))
variable_values = obj.variable_values
state = solver.init_state(variable_values)
for _ in range(200):
    variable_values, state = solver.step(variable_values, state)
```

Fig. 4.3 shows the convergence of SAPPHIRE on a CIFAR-10 classification task, where $N = 50{,}000$ and $p = 3{,}072$. The GPU implementation of SAPPHIRE achieves a $5\times$ speedup over CPU, with both implementations converging towards the same solution.

4.6.3 Bounded Elastic Net via NysADMM

Our third example is the bounded elastic net problem:

$$\begin{array}{ll} \underset{w,b}{\text{minimize}} & \frac{1}{2N}\|Xw + b - y\|_2^2 + \lambda_1\|w\|_1 + \lambda_2\|w\|_2^2 \\ \text{subject to} & 0 \le w \le 1, \end{array}$$

where $\lambda_1, \lambda_2 > 0$ are regularization parameters. This problem combines a smooth least-squares loss with nonsmooth regularizers and box constraints, making it a natural fit for NysADMM. In `rlaopt`:

```
# Define the objective
lambd = 0.1 * torch.linalg.norm(X.T @ y, ord=float("inf")) / X.shape[0]
w = Variable((X.shape[1],))
b = Variable((1,))
obj = SumSquares(X @ w + b - y) * (0.5 / X.shape[0]) \
    + ElasticNet(w, lambd, lambd) \
    + Box(w, 0.0, 1.0)

# Configure and run NysADMM (API class: ADMM)
solver = ADMM(obj, config=ADMMConfig())
variable_values = obj.variable_values
state = solver.init_state(variable_values)
for _ in range(20):
    variable_values, state = solver.step(variable_values, state)
```

Fig. 4.4 shows the convergence of NysADMM on the e2006-tfidf dataset, which has size $N =$ 16,087 and $p =$ 150,360. NysADMM converges rapidly, with the GPU implementation achieving a 25× speedup over CPU.

4.6.4 GPU Speedups

A key advantage of `rlaopt` is that all solvers run on GPU with no code changes beyond moving the data to the device. Fig. 4.5 summarizes the CPU-to-GPU speedups across the three solver families. NyströmPCG achieves up to 20× speedup, NysADMM achieves 20–30× speedup, and SAPPHIRE achieves 5–10× speedup. The larger speedups for NyströmPCG and NysADMM reflect their reliance on dense matrix-matrix products, which are highly parallelizable on GPU. SAPPHIRE's stochastic mini-batch iterations are less compute-bound, yielding a more modest but still substantial speedup.

4.6.5 Differentiating Through the Solver

Finally, we demonstrate `rlaopt`'s ability to differentiate through the optimization solver. We consider the task of tuning the regularization parameter μ in a lasso problem to minimize the prediction error on a held-out validation set, as formulated in (4.7).

Fig. 4.6 shows the convergence of the outer gradient descent loop that optimizes μ. At each outer iteration, `rlaopt` solves the inner lasso problem using proximal gradient and then computes the gradient of the validation loss with respect to μ via PyTorch's automatic differentiation. The validation loss decreases steadily, demonstrating that `rlaopt` correctly propagates gradients through the solver and enables effective hyperparameter tuning.

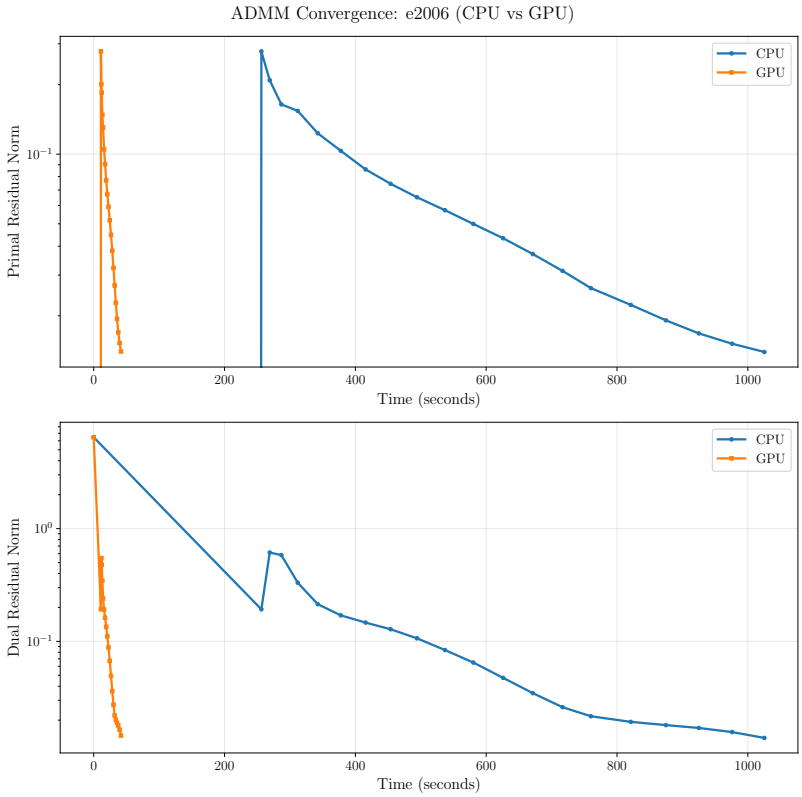
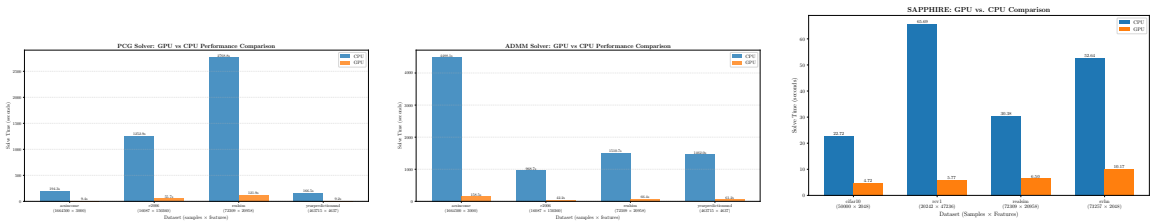


Figure 4.4: NysADMM convergence on the bounded elastic net problem with the E2006-tfidf dataset.

4.7 Conclusion

We have presented `rlaopt`, an open-source, PyTorch-based software package for large-scale optimization using randomized linear algebra. `rlaopt` addresses a significant gap between the algorithmic foundations of RandNLA and the software available to practitioners, providing GPU-enabled implementations of NyströmPCG, NysADMM, and SAPPHIRE within a unified framework. We have introduced a flexible modeling language inspired by CVXPY that allows users to specify optimization problems using natural mathematical syntax, and described how `rlaopt` automatically detects problem structure to perform the appropriate decomposition for each solver. Our preliminary experiments demonstrate that RandNLA-based preconditioning consistently outperforms standard baselines across problem classes, and that GPU acceleration yields speedups of 5–30$\times$ over CPU.

Two directions are planned for future work. First, we will expand the set of supported atoms and solvers, including NysNewton-CG [Rathore et al., 2024] (Chapter 3), which combines NyströmPCG with Newton's method, and ASkotch [Rathore et al., 2026] (Chapter 2), which combines RandNLA with sketch-and-project solvers [Gower and Richtárik, 2015] for linear systems. Second, we will extend `rlaopt` to support more scientific computing applications from RandNLA, including stochastic



(a) NyströmPCG: up to 20×. (b) NysADMM: 20–30×. (c) SAPPHIRE: 5–10×.

Figure 4.5: CPU-to-GPU speedups across all solver families in `rlaopt`.

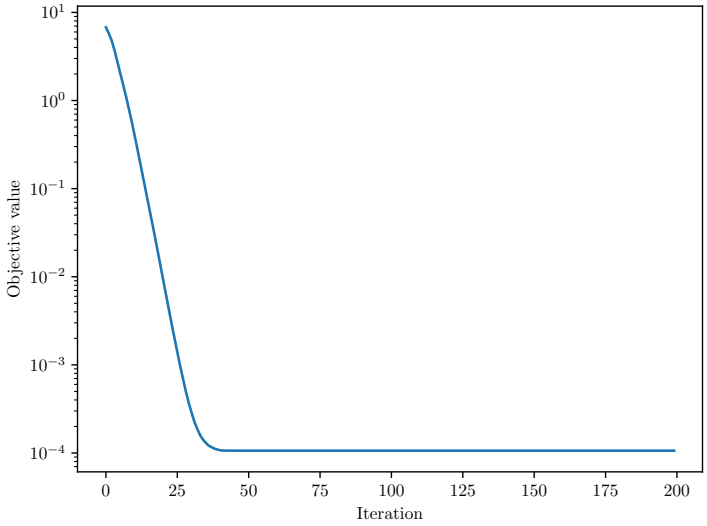


Figure 4.6: Differentiating through `rlaopt`'s proximal gradient solver enables gradient-based tuning of the lasso regularization parameter μ. The validation loss decreases as μ is optimized.

trace estimation [Hutchinson, 1990, Meyer et al., 2021] and spectral density estimation [Lin et al., 2016, Ubaru et al., 2017, Yao et al., 2020].

`rlaopt` is available at https://github.com/udellgroup/rlaopt, with documentation at https://rlaopt.readthedocs.io

Chapter 5

Conclusions

5.1 Summary of Contributions

This dissertation has studied ill-conditioning across three important settings in machine learning: kernel ridge regression, the optimization of physics-informed neural networks, and convex composite and stochastic optimization. Across these settings, we identified where ill-conditioning arises, explained why it limits the state of the art, and developed algorithms, analyses, and software that correct it.

In Chapter 2, we developed `ASkotch`, an iterative method for full KRR that combines sketch-and-project with the randomized Nyström approximation. `ASkotch` enjoys condition-number-free linear convergence under mild assumptions on the effective dimension of the kernel matrix, has per-iteration cost linear in n and storage cost independent of n, and outperforms the state of the art for full and inducing-points KRR across 23 large-scale problems. `ASkotch` is the first method to scale full KRR to a dataset with $n = 10^8$ samples on a single 48 GB GPU, a regime where PCG cannot complete a single iteration.

In Chapter 3, we showed that the PINN loss is severely ill-conditioned due to the differential operators in the residual term, and that quasi-Newton methods improve the conditioning of the loss by $1000\times$ or more. We proved that an ill-conditioned linear differential operator induces an ill-conditioned PINN loss, giving a theoretical justification for these empirical findings. Building on this analysis, we introduced NysNewton-CG (NNCG), a second-order optimizer that uses a randomized Nyström approximation of the Hessian to precondition a Newton-CG solve, and showed that running NNCG after the Adam+L-BFGS pipeline rescues PINN training on problems where Adam+L-BFGS stalls at a high loss.

In Chapter 4, we developed `rlaopt`, an open-source, PyTorch-based software package that makes scalable preconditioned optimization accessible to practitioners. `rlaopt` provides GPU-enabled implementations of NyströmPCG, NysADMM, and SAPPHIRE, together with a flexible CVXPY-style

modeling language and support for differentiating through the solver. Across our benchmarks, GPU execution yields speedups of up to 20× for NyströmPCG, 20–30× for NysADMM, and 5–10× for SAPPHIRE over CPU implementations.

Taken together, these contributions show that ill-conditioning need not be an obstacle to scalable machine learning: with the right combination of algorithms, analysis, and software, we can obtain training approaches that are both scalable and robust to conditioning. We close the dissertation by discussing directions for future research that build on the ideas developed here.

5.2 Future Directions

The ideas developed in this dissertation open several promising directions for future research.

Sketch-and-project solvers for dense, positive-definite PDE discretizations. `ASkotch` solves large, dense, positive-definite kernel linear systems by combining sketch-and-project with the randomized Nyström approximation. Several PDE discretization frameworks give rise to linear systems with exactly this structure: kernel and Gaussian-process collocation methods for elliptic PDEs produce dense, symmetric positive-definite systems [Chen et al., 2021], and Galerkin discretizations of boundary integral methods for Laplace, Helmholtz, and elasticity problems also yield dense, symmetric positive-definite systems [Sauter and Schwab, 2011]. Adapting `ASkotch`-style sketch-and-project solvers to these systems is a natural next step, and could yield memory-efficient, condition-number-free iterative PDE solvers in settings where classical preconditioners are expensive to construct or apply.

Constrained formulations of PINNs. The standard PINN loss is a weighted sum of the PDE residual, boundary, and initial condition terms, and performance is sensitive to how these weights are chosen. An alternative is to formulate PINN training as a constrained optimization problem, where the PDE residual, boundary, and initial conditions are all enforced as constraints. For forward problems, the result is a feasibility problem; for inverse problems, the data-fitting loss becomes the objective subject to these constraints. This sidesteps the loss-weighting problem, though the resulting formulation could be challenging to solve because the neural network parameterization makes the constraints nonconvex.

Better optimizers for physics-informed learning. The analysis in Chapter 3 shows that ill-conditioned differential operators induce an ill-conditioned PINN loss. We expect similar ill-conditioning to arise in physics-informed operator-learning methods, such as physics-informed DeepONets [Wang et al., 2021c] and physics-informed neural operators [Li et al., 2024], since these methods incorporate the same differential operators in their training losses. Recent work also shows that preconditioning is essential for mitigating spectral bias in physics-informed learning [Khodakarami et al., 2026], and pushing physics-informed learning to machine precision: the multi-stage neural

network framework of Wang and Lai [2024] uses Adam followed by L-BFGS to reach near-machine-precision accuracy on function approximation and PDE solving, and Wang et al. [2025] relies on a high-precision Gauss-Newton optimizer to find new unstable singularities in fluid dynamics. The insights developed in this dissertation about ill-conditioning and preconditioning with randomized low-rank approximations could inform the design of new optimizers for physics-informed operator learning, high-precision multi-stage training, and singularity discovery.

Appendix A

Supplementary Material for Chapter 2

A.1 Additional Algorithm Details

We provide additional details for the algorithms proposed in this paper. Appendix A.1.1 describes the practical implementation of the randomized Nyström approximation and provides pseudocode for `Nyström`. Appendix A.1.2 describes how we compute preconditioned smoothness constants and provides pseudocode for `get_L`.

A.1.1 Randomized Nyström Approximation: Implementation

Here, we present a practical implementation of the Nyström approximation from Section 2.2.3 in `Nyström` (Algorithm 3). `Nyström` is based on Tropp et al. [2017, Algorithm 3]. eps(x) is defined as the positive distance between x and the next largest floating point number of the same precision as x. The resulting Nyström approximation $\hat{M}$ is given by $\hat{U}\,\mathrm{diag}(\hat{\Lambda})\hat{U}^T$, where $\hat{U} \in \mathbb{R}^{p\times r}$ is an orthogonal matrix that contains the approximate top-r eigenvectors of M, and $\hat{\Lambda} \in \mathbb{R}^r$ contains the top-r eigenvalues of M. The Nyström approximation is psd but may have eigenvalues that are equal to 0. In our algorithms, this approximation is always used in conjunction with a regularizer to ensure positive definiteness.

The dominant costs are in computing the (shifted) sketch Y_Δ, which has complexity $\mathcal{O}(p^2r)$, and computing an SVD of B, which has complexity $\mathcal{O}(pr^2)$. In total, the overall complexity of the algorithm is $\mathcal{O}(p^2r + pr^2)$.

Algorithm 3 `Nyström`

Require: psd matrix $M \in \mathbb{R}^{p\times p}$, approximation rank $r \le p$

$\Omega \leftarrow \text{randn}(p, r)$ ▷ Test matrix
$\Omega \leftarrow \text{thin\_qr}(\Omega)$ ▷ Orthogonalize test matrix
$\Delta \leftarrow \text{eps}(\Omega.\text{dtype}) \cdot \text{Tr}(M)$ ▷ Compute shift
$Y_\Delta \leftarrow (M + \Delta I)\Omega$ ▷ Compute sketch, adding shift for stability
$C \leftarrow \text{chol}(\Omega^T Y_\Delta)$ ▷ Cholesky decomposition: $C^T C = \Omega^T Y_\Delta$
$B \leftarrow YC^{-1}$ ▷ Triangular solve
$[\hat{U}, \Sigma, \sim] \leftarrow \text{svd}(B, 0)$ ▷ Thin SVD
$\hat{\Lambda} \leftarrow \max\{0, \text{diag}(\Sigma^2 - \Delta I)\}$ ▷ Compute eigs, and remove shift with element-wise max
return $\hat{U}\,\text{diag}(\hat{\Lambda})\hat{U}^T$

Applying the Nyström Approximation to a Vector

In our algorithms, we often perform computations of the form $(\hat{M}+\rho I)^{-1}g = (\hat{U}\,\text{diag}(\hat{\Lambda})\hat{U}^T+\rho I)^{-1}g$. This computation can be performed in $\mathcal{O}(rp)$ time using the Woodbury formula [Higham, 2002]:

$$(\hat{U}\,\text{diag}(\hat{\Lambda})\hat{U}^T + \rho I)^{-1}g = \hat{U}\left(\text{diag}(\hat{\Lambda}) + \rho I\right)^{-1}\hat{U}^T g + \frac{1}{\rho}(g - \hat{U}\hat{U}^T g). \tag{A.1}$$

We also use the randomized Nyström approximation to compute preconditioned smoothness constants in `get_L` (Algorithm 4). This computation requires the calculation $(P+\rho I)^{-1/2}v$ for some $v \in \mathbb{R}^p$, which can also be performed in $\mathcal{O}(pr)$ time using the Woodbury formula:

$$(\hat{U}\,\text{diag}(\hat{\Lambda})\hat{U}^T + \rho I)^{-1/2}v = \hat{U}\left(\text{diag}(\hat{\Lambda}) + \rho I\right)^{-1/2}\hat{U}^T v + \frac{1}{\sqrt{\rho}}(v - \hat{U}\hat{U}^T v). \tag{A.2}$$

In single precision, (A.1) is unreliable for computing $(P + \rho I)^{-1}g$. This instability arises due to roundoff error: the derivation of (A.1) assumes that $\hat{U}^T\hat{U} = I$, but we empirically observe that orthogonality does not hold in single precision. To improve stability, we compute a Cholesky decomposition LL^T of $\rho\,\text{diag}(\hat{\Lambda}^{-1}) + \hat{U}^T\hat{U}$, which takes $\mathcal{O}(pr^2)$ time since we form $\hat{U}^T\hat{U}$. Using the Woodbury formula and Cholesky factors,

$$\begin{aligned}(\hat{U}\,\text{diag}(\hat{\Lambda})\hat{U}^T + \rho I)^{-1}g &= \frac{1}{\rho}g - \frac{1}{\rho}\hat{U}(\rho\,\text{diag}(\hat{\Lambda}^{-1}) + \hat{U}^T\hat{U})^{-1}\hat{U}^T g \\ &= \frac{1}{\rho}g - \frac{1}{\rho}\hat{U}L^{-T}L^{-1}\hat{U}^T g.\end{aligned}$$

This computation can be performed in $\mathcal{O}(pr)$ time, since the $\mathcal{O}(r^2)$ cost of triangular solves with L^T and L is negligible compared to the $\mathcal{O}(pr)$ cost of multiplication with $\hat{U}^T$ and $\hat{U}$.

Even in single precision, we find that using (A.2) in `get_L` works well in practice.

A.1.2 Computing Preconditioned Smoothness Constants

Here, we provide the details on the randomized powering procedure (`get_L`, Algorithm 4) from Section 2.2.4 for automatically computing the preconditioned smoothness constant. Given a symmetric

matrix H, preconditioner P, and damping ρ, `get_L` uses randomized powering [Martinsson and Tropp, 2020] to compute

$$\lambda_1((P+\rho I)^{-1/2}H(P+\rho I)^{-1/2}).$$

The algorithm only requires matrix-vector products with the matrices H and $(P+\rho I)^{-1/2}$. When P is calculated using `Nyström`, we can efficiently compute a matrix-vector product with $(P+\rho I)^{-1/2}$ using (A.2). In practice, we find that 10 iterations of randomized powering are sufficient for estimating the preconditioned smoothness constant.

Algorithm 4 `get_L`

Require: symmetric matrix H, preconditioner P, damping ρ, maximum iterations $N \leftarrow 10$
  $v^0 \leftarrow \text{randn}(P.\text{shape}[0])$
  $v^0 \leftarrow v^0/\|v^0\|_2$ ▷ Normalize
  for $i = 0, 1, \ldots, N-1$ **do**
    $v^{i+1} \leftarrow (P+\rho I)^{-1/2} v^i$
    $v^{i+1} \leftarrow H v^{i+1}$
    $v^{i+1} \leftarrow (P+\rho I)^{-1/2} v^{i+1}$
    $v^{i+1} \leftarrow v^{i+1}/\|v^{i+1}\|_2$ ▷ Normalize
  end for
  $\lambda \leftarrow (v^{N-1})^T v^N$
  return λ

A.2 Proofs of Results Appearing in the Main Paper

This appendix provides proofs for every result in the main paper whose proof was omitted. In particular, we provide detailed arguments for the RLS-to-DPP reduction in Section 2.5.2 and the convergence of `ASkotch` (Theorem 2.17).

A.2.1 Proof of Lemma 2.3

Proof. Consider the matrix

$$K_\lambda^{1/2} I_{\mathcal{B}}^T \left(\hat{K}_{\mathcal{B}\mathcal{B}} + \rho I\right)^{-1} I_{\mathcal{B}} K_\lambda^{1/2}.$$

This can be rewritten as

$$K_\lambda^{1/2} I_{\mathcal{B}}^T (K_{\mathcal{B}\mathcal{B}} + \lambda I)^{-1/2} \left[(K_{\mathcal{B}\mathcal{B}} + \lambda I)^{1/2} \left(\hat{K}_{\mathcal{B}\mathcal{B}} + \rho I\right)^{-1} (K_{\mathcal{B}\mathcal{B}} + \lambda I)^{1/2}\right] (K_{\mathcal{B}\mathcal{B}} + \lambda I)^{-1/2} I_{\mathcal{B}} K_\lambda^{1/2}.$$

From this, we deduce

$$\sigma_{P_{\mathcal{B}}} \Pi_\lambda \preceq K_\lambda^{1/2} I_{\mathcal{B}}^T \left(\hat{K}_{\mathcal{B}\mathcal{B}} + \rho I\right)^{-1} I_{\mathcal{B}} K_\lambda^{1/2} \preceq L_{P_B} \Pi_\lambda.$$

By definition, $\hat{L}_{P_{\mathcal{B}}} \geq L_{P_{\mathcal{B}}}$, so

$$\sigma_{P_{\mathcal{B}}} \Pi_\lambda \preceq K_\lambda^{1/2} I_{\mathcal{B}}^T \left(\hat{K}_{\mathcal{B}\mathcal{B}} + \rho I\right)^{-1} I_{\mathcal{B}} K_\lambda^{1/2} \preceq \hat{L}_{P_B} \Pi_\lambda.$$

Multiplying by $\hat{L}_{P_\mathcal{B}}^{-1}$, we establish the claim. □

A.2.2 ARLS-to-DPP Reduction

Here we prove the ARLS-to-DPP reduction in Lemma 2.12.

Proof idea. To prove Lemma 2.12, we first establish that the leverage score estimates $\{\tilde{\ell}_i\}_{i=1}^n$ are rough approximations for the marginal probabilities (i.e., *marginal overestimates*) of any given element $i \in [n]$ being sampled into a set distributed according to j-DPP(A) (Lemma A.1). We must then combine this guarantee with existing results on DPP coupling [Dereziński and Yang, 2024]. However, since prior work focuses on uniform sampling rather than ARLS sampling, we construct a *subdivision*, which uses the marginal overestimates to create an equivalent DPP for which our ARLS sampling maps to uniform sampling (Lemma A.6 and Lemma A.8). This equivalence allows us to call upon existing guarantees for coupling a DPP with uniform sampling (Lemma A.4).

The relationship between leverage score estimates and marginal probabilities is shown in Appendix A.2.2 and the subdivision construction is shown in Appendix A.2.2. The proof of Lemma 2.12 is given in Appendix A.2.2.

Leverage Score Estimates are Rough Approximations of Marginals

We start by showing that approximate RLSs are marginal overestimates of a j-DPP(A).

Lemma A.1 (ARLS are marginal overestimates)**.** *Given $A \in \mathbb{S}_+^n$, $k = \Omega(\log n)$, and $\tilde{\lambda} > 0$ such that $d^{\tilde{\lambda}}(A) \geq 4k$, let $\{\tilde{\ell}_i\}_{i=1}^n$ be c-approximations of $\tilde{\lambda}$-ridge leverage scores of A. For $j \in \mathcal{I} = \left[2k - \sqrt{6k\log\left(\frac{2}{\delta}\right)}, 2k + \sqrt{6k\log\left(\frac{2}{\delta}\right)}\right]$, let $\ell_{i|j} \coloneqq \Pr(i \in \mathcal{B}_{j\text{-DPP}})$ be the i-th marginal probability of $\mathcal{B}_{j\text{-DPP}} \sim j\text{-DPP}(A/\tilde{\lambda})$. Then, $2\tilde{\ell}_i \geq \ell_{i|j}$ for all i.*

The proof of Lemma A.1 requires (i) Lemma A.2, which shows that the size of a DPP sample is close to its expected size with high probability and (ii) Lemma A.3, which shows that $\ell_{i|j}$ is non-decreasing in j:

Lemma A.2. *Given any $A \in \mathbb{S}_+^n$, let $\mathcal{B}_{\text{DPP}} \sim \text{DPP}(A)$ and $\mathbb{E}[|\mathcal{B}_{\text{DPP}}|]$ be its expected size. Then with probability $1-\delta$, we have $\left||\mathcal{B}_{\text{DPP}}| - \mathbb{E}[|\mathcal{B}_{\text{DPP}}|]\right| \leq \sqrt{3\mathbb{E}[|\mathcal{B}_{\text{DPP}}|]\log\left(\frac{2}{\delta}\right)}$.*

Lemma A.3 (Monotonicity)**.** *For any $0 < j_1 \leq j_2$, we have $\ell_{i|j_1} \leq \ell_{i|j_2}$.*

The proofs of Lemmas A.2 and A.3 are deferred to Appendix A.2.2, respectively.

Proof of Lemma A.1. Define $\tilde{\mathcal{B}}_{\text{DPP}} \sim \text{DPP}(A/\tilde{\lambda})$ with $\mathbb{E}[|\tilde{\mathcal{B}}_{\text{DPP}}|] = d^{\tilde{\lambda}} \coloneqq d^{\tilde{\lambda}}(A) \geq 4k$, and denote its i-th marginal probabilities as $\ell_i^{\tilde{\lambda}} = \Pr(i \in \tilde{\mathcal{B}}_{\text{DPP}})$. By Lemma 2.9, $\{\ell_i^{\tilde{\lambda}}\}_{i=1}^n$ are the $\tilde{\lambda}$-ridge leverage scores of A. Also let $w_{\tilde{\lambda},l} \coloneqq \Pr(|\tilde{\mathcal{B}}_{\text{DPP}}| = l)$ be the probability of the set $\tilde{\mathcal{B}}_{\text{DPP}}$ having size l. Then,

we can express the marginals of $\tilde{\mathcal{B}}_{\text{DPP}}$ via the law of total probability:

$$\ell_i^{\tilde{\lambda}} = \Pr(i \in \tilde{\mathcal{B}}_{\text{DPP}}) = \sum_l \Pr(i \in \tilde{\mathcal{B}}_{\text{DPP}} \mid |\tilde{\mathcal{B}}_{\text{DPP}}| = l) \cdot \Pr(|\tilde{\mathcal{B}}_{\text{DPP}}| = l) = \sum_l \ell_{i|l} \cdot w_{\tilde{\lambda},l},$$

where we use the fact that $\Pr(i \in \tilde{\mathcal{B}}_{\text{DPP}} \mid |\tilde{\mathcal{B}}_{\text{DPP}}| = l) = \Pr(i \in \mathcal{B}_{l\text{-DPP}}) = \ell_{i|l}$. Defining the interval $\tilde{\mathcal{I}} \coloneqq \left[d^{\tilde{\lambda}} - \sqrt{3d^{\tilde{\lambda}} \log\left(\frac{2}{\delta}\right)}, d^{\tilde{\lambda}} + \sqrt{3d^{\tilde{\lambda}} \log\left(\frac{2}{\delta}\right)}\right]$, by applying Lemma A.2 to $\tilde{\mathcal{B}}_{\text{DPP}}$, we have $|\tilde{\mathcal{B}}_{\text{DPP}}| \in \tilde{\mathcal{I}}$ holds with probability $1 - \delta$.

Since we assume $\delta = n^{-\mathcal{O}(1)}$ and $d^{\tilde{\lambda}} \geq 4k = \Omega(\log n)$ (with a sufficiently large constant), the infimum of $\tilde{\mathcal{I}}$ can be bounded as $d^{\tilde{\lambda}} - \sqrt{3d^{\tilde{\lambda}} \log\left(\frac{2}{\delta}\right)} \geq \frac{3}{4} d^{\tilde{\lambda}} \geq 3k$. Similarly, the supremum of $\mathcal{I}$ can be bounded as $2k + \sqrt{6k \log\left(\frac{2}{\delta}\right)} \leq 3k$, showing that $\sup \mathcal{I} \leq \inf \tilde{\mathcal{I}}$. By combining this result with Lemma A.3, we have the following for any $j \in \mathcal{I}$:

$$\ell_i^{\tilde{\lambda}} = \sum_{l \in \tilde{\mathcal{I}}} \ell_{i|l} \cdot w_{\tilde{\lambda},l} + \sum_{l \notin \tilde{\mathcal{I}}} \ell_{i|l} \cdot w_{\tilde{\lambda},l} \geq \sum_{l \in \tilde{\mathcal{I}}} \ell_{i|l} \cdot w_{\tilde{\lambda},l} \geq (1-\delta) \cdot \ell_{i|3k} \geq (1-\delta) \cdot \ell_{i|j}.$$

Suppose we have RLS c-approximations $\{\tilde{\ell}_i\}_{i=1}^n$ such that $\tilde{\ell}_i \geq \ell_i^{\tilde{\lambda}}$ (see Definition 2.5). Since $\delta \leq 1/2$, this implies that $2\tilde{\ell}_i \geq \ell_i^{\tilde{\lambda}}/(1-\delta) \geq \ell_{i|j}$ for any $j \in \mathcal{I}$, i.e., the RLS approximations are (up to a factor of 2) marginal overestimates for j-DPP(A). □

Relating ARLS Sampling to DPPs Using Subdivisions

The remainder of our analysis builds on the following lemma from Dereziński and Yang [2024], which couples a fixed-size DPP, $\mathcal{B}_{j\text{-DPP}} \sim j\text{-DPP}(A)$, with uniform sampling.

Lemma A.4 (Dereziński and Yang [2024], Lemma 6.6)**.** *Let $\mathcal{B}_{j\text{-DPP}} \sim j\text{-DPP}(A)$ be a fixed-size DPP sample where $A \in \mathbb{S}_+^n$ and $\log n < j < n$. Suppose the marginal probabilities of j-DPP(A) are near-uniform, that is, $\Pr(i \in \mathcal{B}_{j\text{-DPP}}) \leq cj/n$ holds for all $i \in [n]$ for some $c > 1$. Let $\mathcal{B}$ be an i.i.d. uniform sample from $[n]$ of size $\mathcal{O}(cj \log^3 n)$. Then, there is a coupling between $\mathcal{B}_{j\text{-DPP}}$ and $\mathcal{B}$, such that the joint random variable $(\mathcal{B}_{j\text{-DPP}}, \mathcal{B})$ with probability $1 - n^{-\mathcal{O}(1)}$ satisfies $\mathcal{B}_{j\text{-DPP}} \subseteq \mathcal{B}$.*

Unfortunately, we cannot use Lemma A.4 directly, because the "near-uniform marginals" assumption does not hold in our setting. To address this, we rely on a subdivision process, which transforms a probability distribution (such as a fixed-size DPP) into another distribution which is equivalent (in some sense) and has nearly uniform marginals. This approach, first introduced by Anari and Dereziński [2020], uniformizes the marginals by enlarging the groundset $[n]$ through selective duplication.

Definition A.5 (Subdivision of j-DPP, inspired by Anari and Dereziński [2020])**.** Given $A \in \mathbb{S}_+^n$, define the distribution $\mu \coloneqq j\text{-DPP}(A)$. Let matrix X be such that $A = XX^\top$, and denote $x_i^\top$ as its i-th row. Assume we have marginal overestimates $\{\tilde{\ell}_i\}_{i=1}^n$ such that $\tilde{\ell}_i \geq \Pr_{\mathcal{S} \sim \mu}(i \in \mathcal{S})$ for all

$i \in [n]$. Denote $\tilde{\ell} = \sum_i \tilde{\ell}_i$, let $t_i \coloneqq \left\lceil \frac{n}{\tilde{\ell}} \tilde{\ell}_i \right\rceil$ and $\tilde{n} = \sum_i t_i$. For each $i \in [n]$, we create t_i copies of $\frac{1}{\sqrt{t_i}} x_i^\top$ and let the collection of all these vectors form the new matrix $\tilde{X}$. Let $\tilde{A} = \tilde{X}\tilde{X}^\top \in \mathbb{S}_+^{\tilde{n}}$, we define the subdivision process of μ as $\mu' = j\text{-DPP}(\tilde{A})$.

We start by stating that the subdivision process produces a distribution with near-uniform marginals [Anari et al., 2024].

Lemma A.6 (Anari et al. [2024], Proposition 23)**.** *Let distribution $\mu = j\text{-DPP}(A)$, and $\mu' = j\text{-DPP}(\tilde{A})$ be the subdivision process of μ. Then, μ' has near-uniform marginals: for all $i^{(j)} \in [\tilde{n}]$ we have $\Pr_{\tilde{\mathcal{S}} \sim \mu'}(i^{(j)} \in \tilde{\mathcal{S}}) \leq \frac{\tilde{\ell}}{n} \leq \frac{2\tilde{\ell}}{\tilde{n}}$.*

The following lemma shows an equivalence between $\mu = j\text{-DPP}(A)$ and its subdivision (up to a mapping of the elements), which allows us to transform the problem of sampling from μ to sampling from its subdivision μ'.

Lemma A.7 (Equivalence between j-DPP and its subdivision)**.** *Let distribution $\mu = j\text{-DPP}(A)$, and $\mu' = j\text{-DPP}(\tilde{A})$ be its subdivision process. Let $\tilde{\mathcal{B}} \sim j\text{-DPP}(\tilde{A})$ and define a function $\pi : [\tilde{n}] \to [n]$ which maps the t_i duplicates (in $[\tilde{n}]$) of element $i \in [n]$ back to i. Then, $\pi\big(\tilde{\mathcal{B}}\big) \sim j\text{-DPP}(A)$.*

Proof. By the definition of the subdivision, if two identical rows are sampled from $\tilde{X}$, then the determinant of the corresponding principal submatrix is 0, as is the probability measure μ'. Thus without loss of generality, we can assume that we only sample distinct rows from $\tilde{X}$. Let $\tilde{\mathcal{B}} = \{i_1^{(l_1)}, \ldots, i_j^{(l_j)}\} \subseteq \binom{[\tilde{n}]}{j}$ be a subset sampled from μ', and let $\hat{\mathcal{B}} = \pi\big(\tilde{\mathcal{B}}\big) \coloneqq \{\pi(i_1^{(l_1)}), \ldots, \pi(i_j^{(l_j)})\} = \{i_1, \ldots, i_j\} \subseteq \binom{[n]}{j}$. Then, from definition we have

$$\mu'(\tilde{\mathcal{B}}) \propto \det(\tilde{X}_{\tilde{\mathcal{B}}} \tilde{X}_{\tilde{\mathcal{B}}}^\top) = \frac{\det(X_{\hat{\mathcal{B}}} X_{\hat{\mathcal{B}}}^\top)}{t_{i_1} \cdots t_{i_j}} \propto \frac{\mu(\hat{\mathcal{B}})}{t_{i_1} \cdots t_{i_j}}.$$

Note that there are precisely $t_{i_1} \cdots t_{i_j}$ different sets of duplicates that map to the same set $\hat{\mathcal{B}}$, so a random set $\tilde{\mathcal{B}} \sim \mu' = j\text{-DPP}(\tilde{A})$ after mapping with π is distributed exactly according to $\mu = j\text{-DPP}(A)$. □

Having shown that sampling from μ is equivalent to sampling from its subdivision μ', we need to establish a sampling scheme on $[n]$. Intuitively, importance sampling on $[n]$ based on probabilities proportional to $\{\tilde{\ell}_i\}_{i=1}^n$ should be equivalent to uniform sampling on $[\tilde{n}]$. However, the definition of t_i introduces rounding error. To address this issue, we use the ARLS sampling scheme introduced in Definition 2.5. In the following result, we show the equivalence between this sampling scheme and uniform sampling on the enlarged subdivision ground set $[\tilde{n}]$.

Lemma A.8 (From subdivided uniform sampling to marginal sampling)**.** *Let $\{\tilde{\ell}_i\}_{i=1}^n$ be the marginal overestimates, and define $p_i \coloneqq \frac{\tilde{\ell}}{n} \cdot \left\lceil \frac{n}{\tilde{\ell}} \tilde{\ell}_i \right\rceil$ where $\tilde{\ell} = \sum_i \tilde{\ell}_i$. Let $\mathfrak{o}$ be the importance sampling distribution defined on $[n]$ according to probabilities $\{p_i\}_{i=1}^n$, and $\mathfrak{o}'$ be the uniform sampling distribution defined on $[\tilde{n}]$. Let $I \sim \mathfrak{o}$, and let $\hat{I}$ be a uniformly random element in $\pi^{-1}(I)$. Then $\hat{I} \sim \mathfrak{o}'$.*

Proof. For any $i \in [n]$ and $l \in [t_i]$, we have the following:

$$\begin{aligned}\Pr(\hat{I} = i^{(l)}) &= \Pr(I = i) \cdot \Pr(\hat{I} = i^{(l)} \mid I = i) \\ &= \frac{p_i}{\sum_{j=1}^n p_j} \cdot \frac{1}{t_i} \\ &= \frac{\frac{\tilde{\ell}}{n} t_i}{\frac{\tilde{\ell}}{n} \sum_{j=1}^n t_j} \cdot \frac{1}{t_i} \\ &= \frac{1}{\sum_{j=1}^n t_j} \\ &= \frac{1}{\tilde{n}}.\end{aligned}$$

Therefore, $\hat{I}$ is uniformly random in $[\tilde{n}]$. □

Proof of Lemma 2.12

With all the pieces in place, we now prove Lemma 2.12.

Proof. Suppose that $\{\tilde{\ell}_i\}_{i=1}^n$ are c-approximations of $\tilde{\lambda}$-ridge leverage scores of A as in Definition 2.5. Then, according to Lemma A.1, $\{2\tilde{\ell}_i\}_{i=1}^n$ are the marginal overestimates of j-DPP(A) for any $j \in \mathcal{I}$. Thus by Lemma A.8, our ARLS$_c^{\tilde{\lambda}}$-sampling is equivalent (up to the mapping π) to uniform sampling on the subdivision ground set $[\tilde{n}]$.

Now for $\mu = j$-DPP(A), we look at its subdivision process $\mu' = j$-DPP$(\tilde{A})$. According to Lemma A.6, μ' has near-uniform marginals, i.e., $\Pr_{\mathcal{S}\sim\mu'}(i \in \mathcal{S}) \leq 2cj/n$. Thus by Lemma A.4, if we let $\mathcal{B}'$ be a i.i.d. uniform sample from $[\tilde{n}]$ of size $\mathcal{O}(cj \log^3 \tilde{n}) = \mathcal{O}(ck \log^3 n)$, then there is a coupling $(\mathcal{B}', \mathcal{B}'_{j\text{-DPP}})$, such that with probability $1 - n^{-\mathcal{O}(1)}$, we have $\mathcal{B}'_{j\text{-DPP}} \subseteq \mathcal{B}'$, where $\mathcal{B}'_{j\text{-DPP}} \sim \mu'$.

For one side, by Lemma A.7 we know that $\pi(\mathcal{B}'_{j\text{-DPP}})$ is distributed identically to the sample $\mathcal{B}_{j\text{-DPP}}$, where π is the mapping from Lemma A.7. For the other side, by Lemma A.8, we know that $\pi(\mathcal{B}')$ is distributed according to ARLS$_c^{\tilde{\lambda}}$-sampling, i.e., the same as $\mathcal{B}$ from the statement of Lemma 2.12. We conclude that for any $j \in \mathcal{I}$, if we do ARLS$_c^{\tilde{\lambda}}$-sampling with sample size $b = \mathcal{O}(ck \log^3 n)$ and obtain $\mathcal{B}$, then we can couple $\mathcal{B}$ with $\mathcal{B}_{j\text{-DPP}}$ such that $\mathcal{B}_{j\text{-DPP}} \subseteq \mathcal{B}$ with probability $1 - n^{-\mathcal{O}(1)}$. □

Proof of Lemma A.2

To prove Lemma A.2, we start with the following result from Kulesza and Taskar [2012], which relates the cardinality of a random-size DPP to a sum of independent Bernoulli random variables.

Lemma A.9 (Kulesza and Taskar [2012], Algorithm 1 and Theorem 2.3)**.** *For $A \in \mathbb{S}_+^n$, let $A = \sum_i \lambda_i u_i u_i^\top$ be its eigendecomposition where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. Suppose we independently sample $\gamma_i \sim \text{Bernoulli}\left(\frac{\lambda_i}{\lambda_i + 1}\right)$ for $i \in [n]$ and we also sample $\mathcal{B}_{\text{DPP}} \sim \text{DPP}(A)$. Then $|\mathcal{B}_{\text{DPP}}| \overset{d}{=} \sum_{i=1}^n \gamma_i$.*

We also require the following Chernoff bound, which is a classic result in probability theory:

Lemma A.10 (Chernoff bound)**.** *Let $\bar{Z} = \sum_{i=1}^n Z_i$ where $Z_i \sim$ Bernoulli(p_i) are independent Bernoulli random variables. Let $\mu = \mathbb{E}[\bar{Z}]$, then for all $\epsilon \in (0,1)$ we have*

$$\Pr(|\bar{Z} - \mu| \geq \epsilon\mu) \leq 2\exp(-\epsilon^2\mu/3).$$

Proof of Lemma A.2. Let $\{\lambda_i\}_{i=1}^n$ be the eigenvalues of A in decreasing order, and let $\gamma_i \sim$ Bernoulli $\left(\frac{\lambda_i}{\lambda_i+1}\right)$ be n independent random variables. By Lemma A.9, $\mathbb{E}[|\mathcal{B}_{\text{DPP}}|] = \sum_i \mathbb{E}[\gamma_i] = \sum_i \frac{\lambda_i}{\lambda_i+1}$. Applying Lemma A.10 to $\{\gamma_i\}_{i=1}^n$, we obtain

$$\Pr\left(\left|\sum_i \gamma_i - \sum_i \mathbb{E}[\gamma_i]\right| \geq \epsilon \sum_i \mathbb{E}[\gamma_i]\right) \leq 2\exp\left(-\epsilon^2 \sum_i \mathbb{E}[\gamma_i]/3\right),$$

which gives that with probability $1-\delta$, $\left||\mathcal{B}_{\text{DPP}}| - \mathbb{E}[|\mathcal{B}_{\text{DPP}}|]\right| \leq \sqrt{3\mathbb{E}[|\mathcal{B}_{\text{DPP}}|]\log\left(\frac{2}{\delta}\right)}$. □

Proof of Lemma A.3

The proof of Lemma A.3 relies on elementary symmetric polynomials (Definition A.11) and Newton's inequalities (Lemma A.12).

Definition A.11 (Elementary symmetric polynomial)**.** Given vector $\Lambda = (\lambda_1, \ldots, \lambda_n) \in \mathbb{R}^n$, we define its i-th elementary symmetric polynomial as

$$s_i(\Lambda) := \sum_{\mathcal{S} \in \binom{[n]}{i}} \prod_{j \in \mathcal{S}} \lambda_j.$$

Lemma A.12 (Newton's inequalities)**.** *For non-negative real numbers $\lambda_1, \ldots, \lambda_n$, let s_j be the j-th elementary symmetric polynomial in $\lambda_1, \ldots, \lambda_n$. If we denote $\bar{s}_j = s_j / \binom{n}{j}$, then $\bar{s}_j^2 \geq \bar{s}_{j+1}\bar{s}_{j-1}$.*

Proof of Lemma A.3. Recall that we define $\ell_{i|j} = \Pr(i \in \mathcal{B}_{j\text{-DPP}})$, and it can also be written as $\ell_{i|j} = \Pr(i \in \mathcal{B}_{\text{DPP}} \mid |\mathcal{B}_{\text{DPP}}| = j)$, where $\mathcal{B}_{\text{DPP}} \sim \text{DPP}(A/\bar{\lambda})$ is a random-size DPP sample. Next, we use Kulesza and Taskar [2012, Eq. 5.33]:

$$\ell_{i|j} = \Pr(i \in \mathcal{B}_{\text{DPP}} \mid |\mathcal{B}_{\text{DPP}}| = j) = \sum_{p=1}^n (e_i^T v_p)^2 \lambda_p \frac{s_{j-1}(\Lambda_{-p})}{s_j(\Lambda)},$$

where $\{e_i\}_{i=1}^n$ are the standard basis vectors, $\{v_p\}_{p=1}^n$ are the eigenvectors of A, and $\lambda_p := \lambda_p(A/\bar{\lambda})$.

In order to show that $\ell_{i|j}$ is non-decreasing in j, it suffices to show that $\frac{s_{j-1}(\Lambda_{-p})}{s_j(\Lambda)}$ is non-decreasing in j for any given $p \in [n]$. Denote $\bar{s}_j = s_j(\Lambda)/\binom{[n]}{j}$ as the mean of the j-th elementary symmetric

polynomial, then by Lemma A.12 we have

$$1 \geq \frac{\bar{s}_{j-1}\bar{s}_{j+1}}{\bar{s}_j^2} = \frac{\binom{[n]}{j}^2}{\binom{[n]}{j-1}\binom{[n]}{j+1}} \frac{s_{j-1}(\Lambda)s_{j+1}(\Lambda)}{s_j^2(\Lambda)} = \frac{(j+1)(n-j+1)}{j(n-j)} \frac{s_{j-1}(\Lambda)s_{j+1}(\Lambda)}{s_j^2(\Lambda)},$$

which gives

$$\frac{s_{j+1}(\Lambda)}{s_j(\Lambda)} \leq \frac{j(n-j)}{(j+1)(n-j+1)} \frac{s_j(\Lambda)}{s_{j-1}(\Lambda)} < \frac{s_j(\Lambda)}{s_{j-1}(\Lambda)}.$$

Notice that $s_{j+1}(\Lambda) = s_{j+1}(\Lambda_{-p}) + \lambda_p \cdot s_j(\Lambda_{-p})$, and by using this fact we have

$$\frac{s_j(\Lambda_{-p})}{s_{j+1}(\Lambda)} = \frac{s_j(\Lambda_{-p})}{s_{j+1}(\Lambda_{-p}) + \lambda_p \cdot s_j(\Lambda_{-p})} = \frac{1}{\lambda_p + \frac{s_{j+1}(\Lambda_{-p})}{s_j(\Lambda_{-p})}}.$$

We have shown that $\frac{s_{j+1}(\Lambda_{-p})}{s_j(\Lambda_{-p})}$ is decreasing in j when p is fixed, so

$$\frac{s_j(\Lambda_{-p})}{s_{j+1}(\Lambda)} = \frac{1}{\lambda_p + \frac{s_{j+1}(\Lambda_{-p})}{s_j(\Lambda_{-p})}} > \frac{1}{\lambda_p + \frac{s_j(\Lambda_{-p})}{s_{j-1}(\Lambda_{-p})}} = \frac{s_{j-1}(\Lambda_{-p})}{s_j(\Lambda)},$$

which shows that $\frac{s_{j-1}(\Lambda_{-p})}{s_j(\Lambda)}$ is increasing in j for any $p \in [n]$. □

A.2.3 Proof of Corollary 2.14

Proof. Theorem 2.13 guarantees that

$$\hat{\mu} \geq \frac{\lambda}{16\rho\bar{\kappa}_{k:n}} \frac{k}{n}.$$

We upper bound $\bar{\kappa}_{k:n}$ to deduce the claim. By definition,

$$\begin{aligned}
\bar{\kappa}_{k:n} &= \frac{1}{n-k} \sum_{j>k} \frac{\lambda_j(K_\lambda)}{\lambda_{\min}(K_\lambda)} \\
&= \frac{1}{n-k} \sum_{j>k} \frac{\lambda_j(K) + \lambda}{\lambda_{\min}(K) + \lambda} \\
&\leq \frac{1}{n-k} \sum_{j>k} \frac{2\lambda}{\lambda_{\min}(K) + \lambda} \\
&\leq 2.
\end{aligned}$$

Here the third inequality uses $\lambda_j(K) \leq \lambda$ whenever $j \geq 2d^\lambda(K)$ [Frangella et al., 2023, Lemma 5.4]. Thus, $\bar{\kappa}_{k:n}^{-1} \geq 1/2$, which yields the claim. □

A.2.4 Proof of Proposition 2.15

We begin with the following preliminary technical result.

Lemma A.13. *Let $K \in \mathbb{S}^n_+$, $I_\mathcal{B} \in \mathbb{R}^{b\times n}$ be a row-selection matrix generated via an arbitrary sampling scheme. Set $K_{\mathcal{B}\mathcal{B}} = I_\mathcal{B} K I_\mathcal{B}^T$. Then, for any $\rho > 0$,*

$$d^\rho(K_{\mathcal{B}\mathcal{B}}) \leq \sum_{j=1}^{b} \frac{\lambda_j(K)}{\lambda_j(K)+\rho}.$$

Proof. As $K_{\mathcal{B}\mathcal{B}} = I_\mathcal{B} K I_\mathcal{B}^T$, it is a compression of K to the subspace spanned by the columns of $I_\mathcal{B}^T$. Consequently, Cauchy's Interlacing Theorem [Bhatia, 2013, Corollary III.1.5] yields

$$\lambda_j(K_{\mathcal{B}\mathcal{B}}) \leq \lambda_j(K).$$

Now, by definition of $d^\rho(\cdot)$ and the fact that $f(x) = x/(x+\rho)$ is increasing in x for $\rho > 0$, we have

$$d^\rho(K_{\mathcal{B}\mathcal{B}}) = \sum_{j=1}^{b} \frac{\lambda_j(K_{\mathcal{B}\mathcal{B}})}{\lambda_j(K_{\mathcal{B}\mathcal{B}})+\rho} \leq \sum_{j=1}^{b} \frac{\lambda_j(K)}{\lambda_j(K)+\rho}.$$

□

We now commence the proof of Proposition 2.15.

Proof of Proposition 2.15. We begin by observing that Lemma A.13 yields $d^\rho(K_{\mathcal{B}\mathcal{B}}) \leq d^\rho(\lfloor K \rfloor_b)$. As $\hat{K}_{\mathcal{B}\mathcal{B}}$ is constructed from a sparse sign embedding with $r = \mathcal{O}\left(d^\rho(\lfloor K\rfloor_b)\log\left(\frac{d^\rho(\lfloor K\rfloor_b)}{\delta}\right)\right)$ columns and $\zeta = \mathcal{O}\left(\log(\frac{d^\rho(\lfloor K\rfloor_b)}{\delta})\right)$ non-zeros per column, Lemma 4.6 in Dereziński et al. [2025b] implies

$$\|K_{\mathcal{B}\mathcal{B}} - \hat{K}_{\mathcal{B}\mathcal{B}}\| \leq 2\lambda_{r+1}(K_{\mathcal{B}\mathcal{B}}) + \frac{1}{r}\sum_{j>r}\lambda_j(K_{\mathcal{B}\mathcal{B}}), \text{ with probability at least } 1-\delta.$$

Now, combining Lemma 5.4 in Frangella et al. [2023] with $r = \mathcal{O}\left(d^\rho(\lfloor K\rfloor_b)\log\left(\frac{d^\rho(\lfloor K\rfloor_b)}{\delta}\right)\right)$ yields

$$\lambda_{r+1}(K_{\mathcal{B}\mathcal{B}}) \leq \frac{\rho}{4}, \quad \frac{1}{r}\sum_{j>r}\lambda_j(K_{\mathcal{B}\mathcal{B}}) \leq \frac{\rho}{2}.$$

Thus,

$$\|K_{\mathcal{B}\mathcal{B}} - \hat{K}_{\mathcal{B}\mathcal{B}}\| \leq \rho.$$

Combining this last display with the relation that $\hat{K}_{\mathcal{B}\mathcal{B}} \preceq K_{\mathcal{B}\mathcal{B}}$ and our hypothesis that $\rho \geq \lambda$, we

find

$$\begin{aligned} K_{\mathcal{BB}} + \lambda I \preceq K_{\mathcal{BB}} + \rho I = \hat{K}_{\mathcal{BB}} + \rho I + \left(K_{\mathcal{BB}} - \hat{K}_{\mathcal{BB}}\right) \\ \preceq \hat{K}_{\mathcal{BB}} + \rho I + \rho I \preceq (1+1)\left(\hat{K}_{\mathcal{BB}} + \rho I\right) \\ = 2(\hat{K}_{\mathcal{BB}} + \rho I). \end{aligned}$$

Moreover,

$$K_{\mathcal{BB}} + \lambda I = K_{\mathcal{BB}} + \frac{\lambda}{\rho}\rho I \succeq \frac{\lambda}{\rho}(K_{\mathcal{BB}} + \rho I).$$

Combining these bounds with $K_{\mathcal{BB}} \succeq \hat{K}_{\mathcal{BB}}$, we deduce

$$\frac{\lambda}{\rho}\left(\hat{K}_{\mathcal{BB}} + \rho I\right) \preceq K_{\mathcal{BB}} + \lambda I \preceq 2\left(\hat{K}_{\mathcal{BB}} + \rho I\right).$$

The preceding display immediately implies

$$\frac{\lambda}{\rho} \leq \sigma_{P_{\mathcal{B}}} \leq L_{P_{\mathcal{B}}} \leq 2.$$

As $\hat{L}_{P_{\mathcal{B}}} = \max\{L_{P_{\mathcal{B}}}, 1\}$, it follows that $\hat{L}_{P_{\mathcal{B}}} \leq 2$. Invoking Lemma 2.3 we conclude the proof.

□

A.2.5 Proof of Theorem 2.17

In this section, we prove Theorem 2.17 by analyzing the convergence of `ASkotch`. We begin with some preliminaries and notation.

Preliminaries and Notation

The convergence analysis of `ASkotch` is based on a Lyapunov function argument. For NSAP applied to a symmetric psd matrix $A \in \mathbb{R}^n$, Gower et al. [2018] establishes convergence in terms of the Lyapunov function

$$\Delta_t \coloneqq \|v_t - w_\star\|^2_{B^{1/2}\mathbb{E}[\Pi_{\mathcal{B}}]^+ B^{1/2}} + \frac{1}{\mu}\|w_t - w_\star\|^2_B,$$

where $B \in \mathbb{S}^n_{++}$, $\Pi_{\mathcal{B}} \coloneqq B^{-1/2}AS^T\left(SAB^{-1}AS^T\right)^+ SAB^{-1/2}$, and $\mu \coloneqq \lambda^+_{\min}(\mathbb{E}[\Pi_{\mathcal{B}}])$.

In particular, they establish the following result:

Proposition A.14 (Gower et al. [2018], Eq. 39)**.** *Suppose that*

$$\operatorname{null}(\mathbb{E}[\Pi_{\mathcal{B}}]) = \operatorname{null}(A).$$

Then at iteration t, NSAP satisfies

$$\mathbb{E}[\Delta_t \mid w_{t-1}, v_{t-1}, z_{t-1}] \leq \left(1 - \sqrt{\frac{\mu}{\nu}}\right) \Delta_{t-1},$$

where

$$\mu = \lambda_{\min}\left(\mathbb{E}[\Pi_{\mathcal{B}}]\right), \quad \nu \coloneqq \lambda_{\max}\left(\mathbb{E}\left[\left(\mathbb{E}[\Pi_{\mathcal{B}}]^{-1/2}\Pi_{\mathcal{B}}\mathbb{E}[\Pi_{\mathcal{B}}]^{-1/2}\right)^2\right]\right).$$

Proposition A.14 introduces a new parameter, ν, the largest eigenvalue of the second moment of the normalized projection matrix. Similar to how $\hat{\mu}$ is an analogue μ for analyzing `ASkotch`, there is an analogue to ν for analyzing `ASkotch`:

$$\hat{\nu} \coloneqq \lambda_{\max}\left(\mathbb{E}\left[\left(\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]^{-1/2}\hat{\Pi}_{\mathcal{B},\rho}\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]^{-1/2}\right)^2\right]\right).$$

This new parameter $\hat{\nu}$ corresponds to the second moment of the normalized approximate projection matrix. To establish our convergence result, we need to upper bound $\hat{\nu}$. The following lemma does precisely this.

Lemma A.15 (Upper bound on $\hat{\nu}$)**.** *Suppose that $\mathcal{B}$ and $\hat{K}_{\mathcal{B}\mathcal{B}}$ are constructed according to the hypotheses of Theorem 2.17, then*

$$\hat{\nu} \leq \frac{8(\rho + \bar{\lambda})}{\lambda} \leq 8\left(\frac{\rho}{\lambda} + \frac{n}{k}\right),$$

where $\bar{\lambda}$ is as in Lemma 2.10.

Proof. The proof parallels the proof of Theorem 3.7 in Dereziński et al. [2025c]. Our hypothesis on the construction of $\mathcal{B}$ and $\hat{K}_{\mathcal{B}\mathcal{B}}$ allows us to invoke Theorem 2.13 to reach

$$\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}] \succeq \frac{1}{8\kappa_\rho} K_\lambda (K_\lambda + \bar{\lambda})^{-1},$$

where $\kappa_\rho = \rho/\lambda \geq 1$. The preceding display implies

$$\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]^{-1} \preceq 8\kappa_\rho (I + \bar{\lambda} K_\lambda^{-1}).$$

To upper bound $\hat{\nu}$, we observe that it may be rewritten as

$$\hat{\nu} = \left\| \mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]^{-1/2} \mathbb{E}\left[\hat{\Pi}_{\mathcal{B},\rho}\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]^{-1}\hat{\Pi}_{\mathcal{B},\rho}\right] \mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]^{-1/2} \right\|.$$

Combining this with the upper bound on $\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]^{-1}$ and the fact that $\hat{\Pi}_{\mathcal{B},\rho} \preceq I$, we deduce

$$\begin{aligned}
\hat{\nu} &\leq 8\kappa_\rho \left\| \mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]^{-1/2} \mathbb{E}\left[\hat{\Pi}_{\mathcal{B},\rho}^2 + \bar{\lambda}\hat{\Pi}_{\mathcal{B},\rho} K_\lambda^{-1} \hat{\Pi}_{\mathcal{B},\rho}\right] \mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]^{-1/2} \right\| \\
&\leq 8\kappa_\rho \left\| \mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]^{-1/2} \mathbb{E}\left[\hat{\Pi}_{\mathcal{B},\rho} + \bar{\lambda}\hat{\Pi}_{\mathcal{B},\rho} K_\lambda^{-1} \hat{\Pi}_{\mathcal{B},\rho}\right] \mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]^{-1/2} \right\| \\
&\leq 8\kappa_\rho \left(1 + \bar{\lambda} \left\| \mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]^{-1/2} \mathbb{E}\left[\hat{\Pi}_{\mathcal{B},\rho} K_\lambda^{-1} \hat{\Pi}_{\mathcal{B},\rho}\right] \mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]^{-1/2} \right\| \right). \qquad (*)
\end{aligned}$$

Now, using the definition of $\hat{\Pi}_{\mathcal{B},\rho}$, we can express the middle term as follows:

$$\begin{aligned}
\hat{\Pi}_{\mathcal{B},\rho} K_\lambda^{-1} \hat{\Pi}_{\mathcal{B},\rho} &= \frac{1}{\hat{L}_{P_\mathcal{B}}^2} K_\lambda^{1/2} I_\mathcal{B}^T (\hat{K}_{\mathcal{B}\mathcal{B}} + \rho I)^{-1} I_\mathcal{B} (K_\lambda^{1/2} K_\lambda^{-1} K_\lambda^{1/2}) I_\mathcal{B}^T (\hat{K}_{\mathcal{B}\mathcal{B}} + \rho I)^{-1} I_\mathcal{B} K_\lambda^{1/2} \\
&= \frac{1}{\hat{L}_{P_\mathcal{B}}^2} K_\lambda^{1/2} I_\mathcal{B}^T (\hat{K}_{\mathcal{B}\mathcal{B}} + \rho I)^{-1} I_\mathcal{B} I_\mathcal{B}^T (\hat{K}_{\mathcal{B}\mathcal{B}} + \rho I)^{-1} I_\mathcal{B} K_\lambda^{1/2} \\
&= \frac{1}{\hat{L}_{P_\mathcal{B}}^2} K_\lambda^{1/2} I_\mathcal{B}^T (\hat{K}_{\mathcal{B}\mathcal{B}} + \rho I)^{-2} I_\mathcal{B} K_\lambda^{1/2} \\
&\preceq \frac{1}{\rho \hat{L}_{P_\mathcal{B}}^2} K_\lambda^{1/2} I_\mathcal{B}^T (\hat{K}_{\mathcal{B}\mathcal{B}} + \rho I)^{-1} I_\mathcal{B} K_\lambda^{1/2} \\
&= \frac{1}{\rho \hat{L}_{P_\mathcal{B}}} \hat{\Pi}_{\mathcal{B},\rho} \preceq \frac{1}{\rho} \hat{\Pi}_{\mathcal{B},\rho}.
\end{aligned}$$

Here, the last inequality uses that by definition $\hat{L}_{P_\mathcal{B}} \geq 1$. By plugging the preceding upper bound into $(*)$, we reach

$$\begin{aligned}
\hat{\nu} &\leq 8\kappa_\rho \left(1 + \frac{\bar{\lambda}}{\rho} \left\| \mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]^{-1/2} \mathbb{E}\left[\hat{\Pi}_{\mathcal{B},\rho} K_\lambda^{-1} \hat{\Pi}_{\mathcal{B},\rho}\right] \mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]^{-1/2} \right\| \right) \\
&= 8\kappa_\rho + \frac{8\kappa_\rho \bar{\lambda}}{\rho} \\
&= \frac{8\left(\rho + \bar{\lambda}\right)}{\lambda}.
\end{aligned}$$

This proves the first inequality. To prove the second inequality, recall that Lemma 2.10 implies

$\bar{\lambda} \leq \frac{1}{k}\sum_{j>k}\lambda_j(K_\lambda)$. Combining this with the preceding display, we conclude

$$\begin{aligned}\hat{\nu} &\leq \frac{8\left(\rho + \frac{1}{k}\sum_{j>k}\lambda_j(K_\lambda)\right)}{\lambda} \\ &= 8\frac{\rho + \frac{n-k}{k}\lambda + \frac{1}{k}\sum_{j>k}\lambda_j(K)}{\lambda} \\ &\leq 8\frac{\rho + \frac{n-k}{k}\lambda + \lambda}{\lambda} \\ &= 8\left(\frac{\rho}{\lambda} + \frac{n}{k}\right).\end{aligned}$$

Here, the second inequality follows from the fact that $\frac{1}{k}\sum_{j>k}\lambda_j(K) \leq \lambda$ when $k \geq d^\lambda(K)$ [Frangella et al., 2023, Lemma 5.4]. □

Convergence Proof of `ASkotch`

To show the convergence of `ASkotch`, we use the Lyapunov function

$$\tilde{\Delta}_t \coloneqq \|v_t - w_\star\|^2_{K_\lambda^{1/2}\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]^{-1}K_\lambda^{1/2}} + \frac{1}{\hat{\mu}}\|w_t - w_\star\|^2_{K_\lambda}, \tag{A.3}$$

where $\hat{\mu} = \lambda_{\min}(\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}])$. The Lyapunov function (A.3) is identical to the one for NSAP except $\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]$ replaces $\mathbb{E}[\Pi_{\mathcal{B}}]$ and $\hat{\mu}$ replaces μ. The replacement stems from the fact that `ASkotch` computes the search direction with $\frac{1}{\hat{L}_{P_{\mathcal{B}}}}(\hat{K}_{\mathcal{B}\mathcal{B}} + \rho I)^{-1}$ instead of $(K_{\mathcal{B}\mathcal{B}} + \lambda I)^{-1}$.

Proof of Theorem 2.17, Item 2. The only difference between `ASkotch` and NSAP is that $\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]$ replaces $\mathbb{E}[\Pi_{\mathcal{B}}]$, so we can apply the same argument as Gower et al. [2018] to show the convergence for the modified Lyapunov function (A.3). Thus, we only need to ensure $\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]$ satisfies the following condition from Proposition A.14:

$$\text{null}(\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]) = \text{null}(K_\lambda) = \{0\}.$$

Indeed, this is necessary for the modified Lyapunov function to make sense; otherwise, $\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]$ is singular. Under the hypotheses of Theorem 2.17, Theorem 2.13 guarantees $\hat{\mu} > 0$, which immediately implies

$$\text{null}(\mathbb{E}[\hat{\Pi}_{\mathcal{B},\rho}]) = \text{null}(K_\lambda) = \{0\}.$$

Thus, we can invoke arguments in Gower et al. [2018] to arrive at a modified version of Proposition A.14 where μ and ν are replaced by $\hat{\mu}$ and $\hat{\nu}$ respectively. Applying this modified version of

Proposition A.14, we deduce

$$\mathbb{E}[\tilde{\Delta}_t \mid w_{t-1}, v_{t-1}, z_{t-1}] \leq \left(1 - \sqrt{\frac{\hat{\mu}}{\hat{\nu}}}\right) \tilde{\Delta}_{t-1}.$$

Applying the law of total expectation yields

$$\mathbb{E}[\tilde{\Delta}_t] \leq \left(1 - \sqrt{\frac{\hat{\mu}}{\hat{\nu}}}\right)^t \tilde{\Delta}_0.$$

Using the definition of $\tilde{\Delta}_t$, we reach

$$\begin{aligned}
\mathbb{E}[\|w_t - w_\star\|_{K_\lambda}^2] &\leq \left(1 - \sqrt{\frac{\hat{\mu}}{\hat{\nu}}}\right)^t \left(\hat{\mu}\|w_0 - w_\star\|^2_{K_\lambda^{1/2}\mathbb{E}[\hat{\Pi}_\rho]^{-1}K_\lambda^{1/2}} + \|w_0 - w_\star\|_{K_\lambda}^2\right) \\
&\leq \left(1 - \sqrt{\frac{\hat{\mu}}{\hat{\nu}}}\right)^t \left(\hat{\mu} \cdot \frac{1}{\hat{\mu}}\|w_0 - w_\star\|_{K_\lambda}^2 + \|w_0 - w_\star\|_{K_\lambda}^2\right) \\
&= 2\left(1 - \sqrt{\frac{\hat{\mu}}{\hat{\nu}}}\right)^t \|w_0 - w_\star\|_{K_\lambda}^2. \qquad (*)
\end{aligned}$$

To obtain a fine-grained convergence rate, we apply our bounds on $\hat{\mu}$ and $\hat{\nu}$. Corollary 2.14 lower bounds $\hat{\mu}$ as

$$\hat{\mu} \geq \frac{\lambda}{32\rho}\frac{k}{n}.$$

On the other hand, Lemma A.15 upper bounds $\hat{\nu}$ as

$$\hat{\nu} \leq 8\left(\frac{\rho}{\lambda} + \frac{n}{k}\right) \leq 16 \max\left\{\frac{\rho}{\lambda}, \frac{n}{k}\right\}.$$

Thus,

$$\frac{1}{\hat{\nu}} \geq \frac{1}{16} \min\left\{\frac{\lambda}{\rho}, \frac{k}{n}\right\}.$$

Combining these lower bounds, we deduce

$$\sqrt{\frac{\hat{\mu}}{\hat{\nu}}} \geq \frac{1}{16\sqrt{2}} \min\left\{\sqrt{\frac{\lambda}{\rho}}\frac{k}{n}, \sqrt{\frac{k}{n}}\frac{\lambda}{\rho}\right\}.$$

Combining the preceding display with $(*)$, we conclude the following convergence guarantee:

$$\mathbb{E}[\|w_t - w_\star\|_{K_\lambda}^2] \leq 2\left(1 - \frac{1}{16\sqrt{2}} \min\left\{\sqrt{\frac{\lambda}{\rho}}\frac{k}{n}, \sqrt{\frac{k}{n}}\frac{\lambda}{\rho}\right\}\right)^t \|w_0 - w_\star\|_{K_\lambda}^2.$$

□

A.2.6 Proof of Corollary 2.18

Proof. Our hypothesis that $\max_{i\in[n]} \ell_i^{\tilde{\lambda}}(K_\lambda) = \Theta\left(\frac{d^{\tilde{\lambda}}(K_\lambda)}{n}\right)$ allows us to invoke Corollary 2.11 to ensure that the conclusion of Theorem 2.17 holds when `ASkotch` uses uniform sampling with a blocksize $b = \Theta\left(k \log^3 n\right)$. Moreover, the assumption on ρ guarantees that we are in convergence regime (i). Hence, the number of iterations required to obtain an ϵ-approximate solution is $\mathcal{O}\left(\sqrt{c_\rho}\frac{n}{k}\log\left(\frac{1}{\epsilon}\right)\right)$. When a sparse sign embedding is used to construct $\hat{K}_{\mathcal{B}\mathcal{B}}$, the per-iteration cost of `ASkotch` is $\tilde{\mathcal{O}}(nb + br^2)$. Combining this with the iteration complexity bound and $b = \Theta(k \log^3 n)$, we conclude that the total cost of `ASkotch` is given by

$$\tilde{\mathcal{O}}\left(\sqrt{c_\rho}\left(n^2 + nr^2\right)\log\left(\frac{1}{\epsilon}\right)\right).$$

The claim now follows by observing that $r = \tilde{\mathcal{O}}(\sqrt{n})$ as $d^\rho(\lfloor K \rfloor_b) \leq d^\lambda(K) = \mathcal{O}(\sqrt{n})$. □

Appendix B

Supplementary Material for Chapter 3

B.1 Additional Details on Problem Setup

Here we present the differential equations that we study in our experiments.

B.1.1 Convection

The one-dimensional convection problem is a hyperbolic PDE that can be used to model fluid flow, heat transfer, and biological processes. The convection PDE we study is

$$\begin{aligned} \frac{\partial u}{\partial t} + \beta \frac{\partial u}{\partial x} &= 0, \quad x \in (0, 2\pi), t \in (0, 1), \\ u(x, 0) &= \sin(x), \quad x \in [0, 2\pi], \\ u(0, t) &= u(2\pi, t), \quad t \in [0, 1]. \end{aligned}$$

The analytical solution to this PDE is $u(x, t) = \sin(x - \beta t)$. We set $\beta = 40$ in our experiments.

B.1.2 Reaction

The one-dimensional reaction problem is a non-linear ODE which can be used to model chemical reactions. The reaction ODE we study is

$$\begin{aligned} \frac{\partial u}{\partial t} - \rho u(1-u) = 0, \quad & x \in (0, 2\pi), t \in (0, 1) \\ u(x, 0) = \exp\left(-\frac{(x-\pi)^2}{2(\pi/4)^2}\right), \quad & x \in [0, 2\pi], \\ u(0, t) = u(2\pi, t), \quad & t \in [0, 1]. \end{aligned}$$

The analytical solution to this ODE is $u(x,t) = \frac{h(x)e^{\rho t}}{h(x)e^{\rho t}+1-h(x)}$, where $h(x) = \exp\left(-\frac{(x-\pi)^2}{2(\pi/4)^2}\right)$. We set $\rho = 5$ in our experiments.

B.1.3 Wave

The one-dimensional wave problem is a hyperbolic PDE that often arises in acoustics, electromagnetism, and fluid dynamics. The wave PDE we study is

$$\begin{aligned} \frac{\partial^2 u}{\partial t^2} - 4\frac{\partial^2 u}{\partial x^2} = 0, \quad & x \in (0, 1), t \in (0, 1), \\ u(x, 0) = \sin(\pi x) + \frac{1}{2}\sin(\beta \pi x), \quad & x \in [0, 1], \\ \frac{\partial u(x, 0)}{\partial t} = 0, \quad & x \in [0, 1], \\ u(0, t) = u(1, t) = 0, \quad & t \in [0, 1]. \end{aligned}$$

The analytical solution to this PDE is $u(x,t) = \sin(\pi x)\cos(2\pi t) + \frac{1}{2}\sin(\beta\pi x)\cos(2\beta\pi t)$. We set $\beta = 5$ in our experiments.

B.2 Why Can Low Losses Correspond to Large L2RE?

In Fig. 3.2, there are several instances on the convection PDE and reaction ODE where the PINN loss is close to 0, but the L2RE of the PINN solution is close to 1. Rohrhofer et al. [2023] demonstrate that PINNs can be attracted to points in the loss landscape that minimize the residual portion of the PINN loss, $\frac{1}{2n_{\text{res}}}\sum_{i=1}^{n_{\text{res}}}\left(\mathcal{D}[u(x_r^i; w), x_r^i]\right)^2$, to 0. However, these can correspond to trivial solutions: for the convection PDE, the residual portion is equal to 0 for any constant function u; for the reaction ODE, the residual portion is equal to 0 for constant $u = 0$ or $u = 1$.

To show that the PINN is indeed learning a trivial solution, we visualize two solutions with small residual loss but large L2RE in Fig. B.1. The second column of Fig. B.1 shows the PINN solutions are close to 0 almost everywhere in the domain. Interestingly, the PINN solutions correctly learn

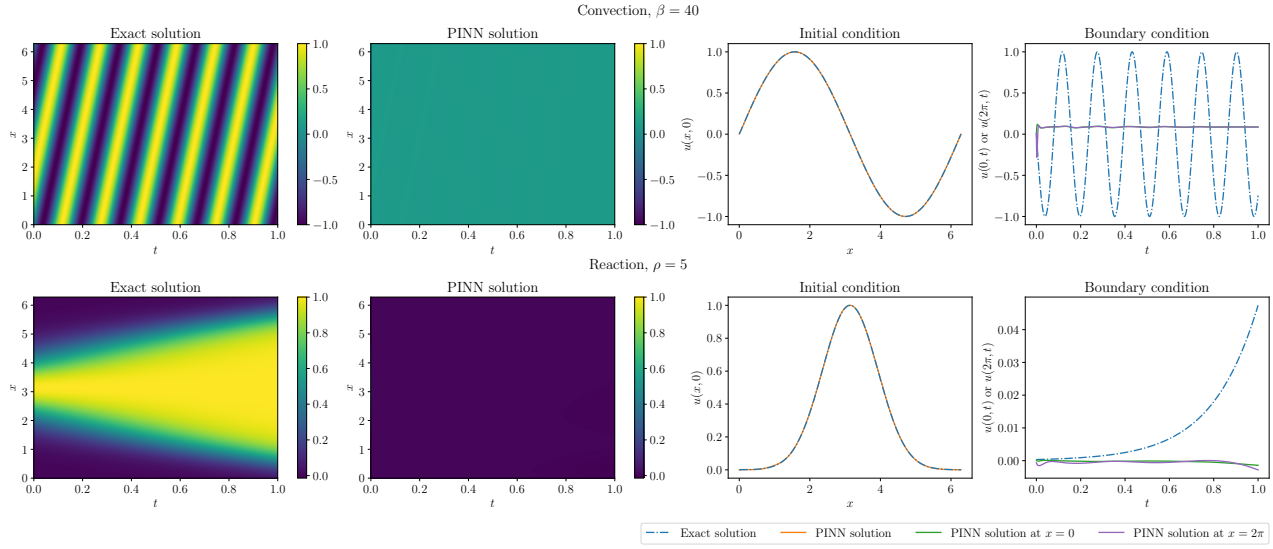


Figure B.1: The first two columns from the left display the exact solutions and PINN solutions. The PINN fails to learn the exact solution, which leads to large L2RE. Moreover, the PINN solutions are effectively constant over the domain. The third and fourth columns from the left display the PINN solutions at the initial time ($t = 0$) and the boundaries ($x = 0$ and $x = 2\pi$). The PINN solutions learn the initial conditions, but they do not learn the boundary conditions.

the initial condition. However, the PINN solutions for the convection PDE and reaction ODE do not match the exact solution at the boundaries. One approach for alleviating this training issue would be to (adaptively) reweight the residual, initial condition, and boundary condition terms in the PINN loss [Wang et al., 2021a, 2022b].

B.3 Computing the Spectral Density of the L-BFGS-Preconditioned Hessian

B.3.1 How L-BFGS Preconditions

To minimize (3.2), L-BFGS uses the update

$$w_{k+1} = w_k - \eta H_k \nabla L(w_k), \tag{B.1}$$

where H_k is a matrix approximating the inverse Hessian. We now show how (B.1) is equivalent to preconditioning the objective (3.2). Define the coordinate transformation $w = H_k^{1/2} z$. By the chain

rule, $\nabla L(z) = H_k^{1/2}\nabla L(w)$ and $H_L(z) = H_k^{1/2}H_L(w)H_k^{1/2}$. Thus, (B.1) is equivalent to

$$\begin{aligned} z_{k+1} &= z_k - \eta\nabla L(z_k), \\ w_{k+1} &= H_k^{1/2} z_{k+1}. \end{aligned} \tag{B.2}$$

Eq. (B.2) reveals how L-BFGS preconditions (3.2). L-BFGS first takes a step in the *preconditioned* z-space, where the conditioning is determined by $H_L(z)$, the preconditioned Hessian. Since H_k approximates $H_L^{-1}(w)$, $H_k^{1/2}H_L(w)H_k^{1/2} \approx I_p$, so the condition number of $H_L(z)$ is much smaller than that of $H_L(w)$. Consequently, L-BFGS can take a step that makes more progress than a method like gradient descent, which performs no preconditioning at all. In the second phase, L-BFGS maps the progress in the preconditioned space back to the original space. Thus, L-BFGS is able to make superior progress by transforming (3.2) to another space where the conditioning is more favorable, which enables it to compute an update that better reduces the loss in (3.2).

B.3.2 Preconditioned Spectral Density Computation

Here we discuss how to compute the spectral density of the Hessian after preconditioning by L-BFGS. This is the procedure we use to generate the figures in Section 3.5.3.

L-BFGS stores a set of vector pairs given by the difference in consecutive iterates and gradients from most recent m iterations (we use $m = 100$ in our experiments). To compute the update direction $H_k\nabla f_k$, L-BFGS combines the stored vector pairs with a recursive scheme [Nocedal and Wright, 2006]. Defining

$$s_k = x_{k+1} - x_k, \quad y_k = \nabla f_{k+1} - \nabla f_k, \quad \rho_k = \frac{1}{y_k^T s_k}, \quad \gamma_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}, \quad V_k = I - \rho_k y_k s_k^T, \quad H_k^0 = \gamma_k I,$$

the formula for H_k can be written as

$$H_k = (V_{k-1}^T V_{k-m}^T)H_k^0(V_{k-m}V_{k-1}) + \sum_{l=2}^{m} \rho_{k-l}(V_{k-1}^T \cdots V_{k-l+1}^T)s_{k-l}s_{k-l}^T(V_{k-l+1}\cdots V_{k-1}) + \rho_{k-1}s_{k-1}s_{k-1}^T.$$

Expanding the terms, we have for $j \in \{1, 2, \ldots, i\}$,

$$V_{k-i}\cdots V_{k-1} = I - \sum_{j=1}^{i} \rho_{k-j}y_{k-j}\tilde{v}_{k-j}^T \quad \text{where} \quad \tilde{v}_{k-j} = s_{k-j} - \sum_{l=1}^{j-1}(\rho_{k-l}y_{k-l}^T s_{k-j})\tilde{v}_{k-l}.$$

It follows that

$$H_k = (I - \tilde{Y}\tilde{V}^T)^T\gamma_k I(I - \tilde{Y}\tilde{V}^T) + \tilde{S}\tilde{S}^T = \begin{bmatrix}\sqrt{\gamma_k}(I - \tilde{Y}\tilde{V}^T)^T & \tilde{S}\end{bmatrix}\begin{bmatrix}\sqrt{\gamma_k}(I - \tilde{Y}\tilde{V}^T) \\ \tilde{S}^T.\end{bmatrix} = \tilde{H}_k\tilde{H}_k^T,$$

where

$$\tilde{Y} = \begin{bmatrix} | & & | \\ \rho_{k-1}y_{k-1} & \cdots & \rho_{k-m}y_{k-m} \\ | & & | \end{bmatrix},$$

$$\tilde{V} = \begin{bmatrix} | & & | \\ \tilde{v}_{k-1} & \cdots & \tilde{v}_{k-m} \\ | & & | \end{bmatrix},$$

$$\tilde{S} = \begin{bmatrix} | & & | \\ \tilde{s}_{k-1} & \cdots & \tilde{s}_{k-m} \\ | & & | \end{bmatrix}, \quad \tilde{s}_{k-1} = \sqrt{\rho_{k-1}}s_{k-1},\ \tilde{s}_{k-l} = \sqrt{\rho_{k-l}}(V_{k-1}^T \cdots V_{k-l+1}^T)s_{k-l} \text{ for } 2 \leq l \leq m.$$

We now apply Algorithm 5 to unroll the above recurrence relations to compute columns of $\tilde{Y}, \tilde{S}$ and $\tilde{V}$.

Algorithm 5 Unrolling the L-BFGS Update

Require: saved directions $\{y_i\}_{i=k-1}^{k-m}$, saved steps $\{s_i\}_{i=k-1}^{k-m}$, saved inverse of inner products $\{\rho_i\}_{i=k-1}^{k-m}$

$\tilde{y}_{k-1} = \rho_{k-1}y_{k-1}$

$\tilde{v}_{k-1} = s_{k-1}$

$\tilde{s}_{k-1} = \sqrt{\rho_{k-1}}s_{k-1}$

for $i = k-2, \ldots, k-m$ **do**

  $\tilde{y}_i = \rho_i y_i$

  Set $\alpha = 0$

  for $j = k-1, \ldots, i+1$ **do**

    $\alpha = \alpha + (\tilde{y}_j^T s_i)\tilde{v}_j$

  end for

  $\tilde{v}_i = s_i - \alpha$

  $\tilde{s}_i = \sqrt{\rho_i}(s_i - \alpha)$

end for

Ensure: vectors $\{\tilde{y}_i, \tilde{v}_i, \tilde{s}_i\}_{i=k-1}^{k-m}$

Since (non-zero) eigenvalues of $\tilde{H}_k^T H_L(w)\tilde{H}_k$ equal the eigenvalues of the preconditioned Hessian $H_k H_L(w) = \tilde{H}_k\tilde{H}_k^T H_L(w)$ (Theorem 1.3.22 of Horn and Johnson [2012]), we can analyze the spectrum of $\tilde{H}_k^T H_L(w)\tilde{H}_k$ instead. This is advantageous since methods for calculating the spectral density of neural network Hessians are only compatible with symmetric matrices.

Since $\tilde{H}_k^T H_L(w)\tilde{H}_k$ is symmetric, we can use stochastic Lanczos quadrature (SLQ) [Golub and Meurant, 2009, Lin et al., 2016] to compute spectral density of this matrix. SLQ only requires

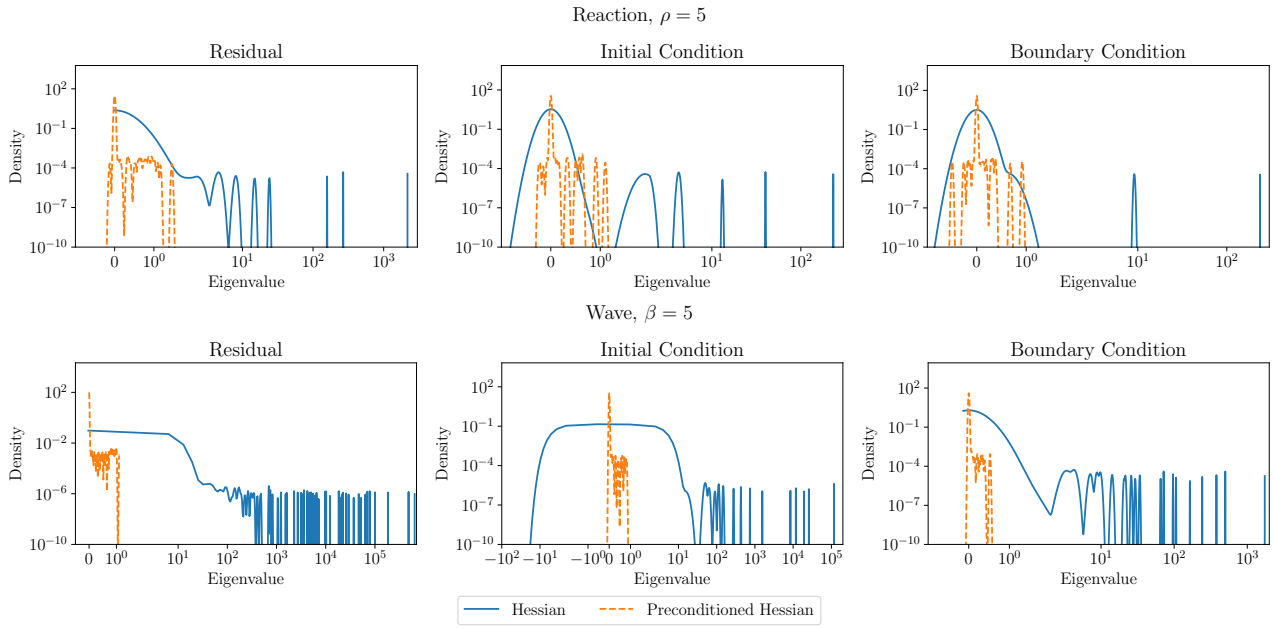


Figure B.2: Spectral density of the Hessian and the preconditioned Hessian of each loss component after 41000 iterations of Adam+L-BFGS for the reaction and wave problems. The plots show the loss landscape of each component is ill-conditioned, and the conditioning of each loss component is improved by L-BFGS.

matrix-vector products with $\tilde{H}_k$ and Hessian-vector products, the latter of which may be efficiently computed via automatic differentiation; this is precisely what PyHessian does to compute spectral densities [Yao et al., 2020].

Algorithm 6 Performing matrix-vector product

Require: matrices $\tilde{Y}$, $\tilde{V}$, $\tilde{S}$ formed from resulting vectors from unrolling, vector v, and saved scaling factor for initializing diagonal matrix γ_k

Split vector v of length size$(w) + m$ into v_1 of size size(w) and v_2 of size m

$v' = \sqrt{\gamma_k}(v_1 - \tilde{V}\tilde{Y}^T v_1) + \tilde{S}v_2$

Perform Hessian-vector-product on v', and obtain v''

Stack $\sqrt{\gamma_k}(v'' - \tilde{Y}\tilde{V}^T v'')$ and $\tilde{S}^T v''$, and obtain v'''

Ensure: resulting vector v'''

By combining the matrix-vector product procedure described in Algorithm 6 with the Hessian-vector product operation, we are able to obtain spectral information of the preconditioned Hessian.

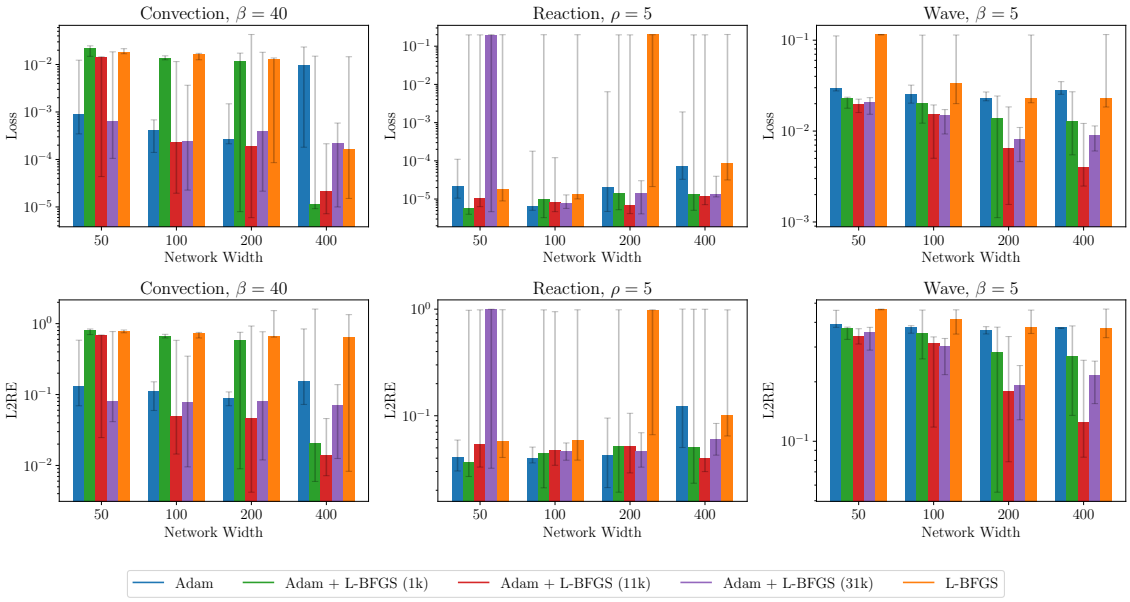


Figure B.3: Performance of Adam, L-BFGS, and Adam+L-BFGS after tuning. We find the learning rate $\eta^\star$ for each network width and optimization strategy that attains the lowest loss (L2RE) across all random seeds. The min, median, and max loss (L2RE) are calculated by taking the min, median, and max of the losses (L2REs) for learning rate η^* across all random seeds. Each bar on the plot corresponds to the median, while the top and bottom error bars correspond to the max and min, respectively. The smallest min loss and L2RE are always attained by one of the Adam+L-BFGS strategies; the smallest median loss and L2RE are nearly always attained by one of the Adam+L-BFGS strategies.

B.4 Adam+L-BFGS Generally Gives the Best Performance

Fig. B.3 shows that Adam+L-BFGS typically yields the best performance on both loss and L2RE across network widths.

B.5 Additional Details on Under-optimization

B.5.1 Early Termination of L-BFGS

Fig. B.4 explains why L-BFGS terminates early for the convection, reaction, and wave problems. We evaluate the loss at 10^4 uniformly spaced points in the interval $[0, 1]$. The orange stars in Fig. B.4 are step sizes that satisfy the strong Wolfe conditions and the red dots are step sizes that L-BFGS examines during the line search.

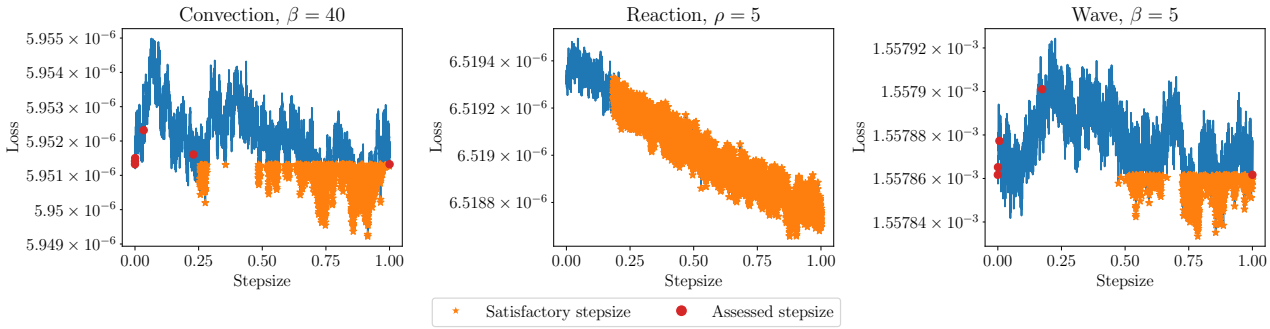


Figure B.4: Loss evaluated along the L-BFGS search direction at different stepsizes after 41000 iterations of Adam+L-BFGS. For convection and wave, the line search does not find a stepsize that satisfies the strong Wolfe conditions, even though there are plenty of such points. For reaction, the slope of the objective used in the line search procedure at the current iterate is less than a pre-defined threshold 10^{-9}, so L-BFGS terminates without performing any line-search.

B.5.2 NysNewton-CG (NNCG)

Here we present the NNCG algorithm (Algorithm 7) introduced in Section 3.7.2 and its associated subroutines RandomizedNyströmApproximation (Algorithm 8), NyströmPCG (Algorithm 9), and Armijo (Algorithm 10). At each iteration, NNCG first checks whether the Nyström preconditioner (stored in U and $\hat{\Lambda}$) for the NyströmPCG method needs to be updated. If so, the preconditioner is recomputed using the RandomizedNyströmApproximation subroutine. From here, the Newton step d_k is computed using NyströmPCG; we warm start the PCG algorithm using the Newton step d_{k-1} from the previous iteration. After computing the Newton step, we compute the step size η_k using Armijo line search — this guarantees that the loss will decrease when we update the parameters. Finally, we update the parameters using η_k and d_k.

In our experiments, we set $\eta = 1, K = 2000, s = 60, F = 20, \epsilon = 10^{-16}, M = 1000, \alpha = 0.1$, and $\beta = 0.5$. We tune $\mu \in [10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]$; we find that $\mu = 10^{-2}, 10^{-1}$ work best in practice. Figs. 3.1 and 3.4 show the NNCG run that attains the lowest loss after tuning μ.

Algorithm 7 NysNewton-CG (NNCG)

Require: Initialization w_0, max. learning rate η, number of iterations K, preconditioner sketch size s, preconditioner update frequency F, damping parameter μ, CG tolerance ϵ, CG max. iterations M, backtracking parameters α, β

$d_{-1} = 0$

for $k = 0, \ldots, K-1$ **do**

  if k is a multiple of F **then**

    $[U, \hat{\Lambda}] = \text{RandomizedNyströmApproximation}(H_L(w_k), s)$ ▷ Update Nyström preconditioner every F iterations

  end if

  $d_k = \text{NyströmPCG}(H_L(w_k), \nabla L(w_k), d_{k-1}, U, \hat{\Lambda}, s, \mu, \epsilon, M)$ ▷ Damped Newton step $(H_L(w_k) + \mu I)^{-1} \nabla L(w_k)$

  $\eta_k = \text{Armijo}(L, w_k, \nabla L(w_k), -d_k, \eta)$ ▷ Compute step size via line search

  $w_{k+1} = w_k - \eta_k d_k$ ▷ Update parameters

end for

The RandomizedNyströmApproximation subroutine (Algorithm 8) is used in NNCG to compute the preconditioner for NyströmPCG. The algorithm returns the top-s approximate eigenvectors and eigenvalues of the input matrix M. Within NNCG, the sketch computation $Y = MQ$ is implemented using Hessian-vector products. The portion in red is a fail-safe that allows for the preconditioner to be computed when H is an indefinite matrix. For further details, please see Frangella et al. [2023].

Algorithm 8 RandomizedNyströmApproximation

Require: Symmetric matrix M, sketch size s

$S = \text{randn}(p, s)$ ▷ Generate test matrix

$Q = \text{qr\_econ}(S)$

$Y = MQ$ ▷ Compute sketch

$\nu = \sqrt{p}\text{eps}(\text{norm}(Y, 2))$ ▷ Compute shift

$Y_\nu = Y + \nu Q$ ▷ Add shift for stability

$\lambda = 0$ ▷ Additional shift may be required for positive definiteness

$C = \text{chol}(Q^T Y_\nu)$ ▷ Cholesky decomposition: $C^T C = Q^T Y_\nu$

if chol fails **then**

Compute $[W, \Gamma] = \text{eig}(Q^T Y_\nu)$ ▷ $Q^T Y_\nu$ is small and square

Set $\lambda = \lambda_{\min}(Q^T Y_\nu)$

$R = W(\Gamma + |\lambda| I)^{-1/2} W^T$

$B = YR$ ▷ R is psd

else

$B = YC^{-1}$ ▷ Triangular solve

end if

$[\hat{V}, \Sigma, \sim] = \text{svd}(B, 0)$ ▷ Thin SVD

$\hat{\Lambda} = \max\{0, \Sigma^2 - (\nu + |\lambda| I)\}$ ▷ Compute eigs, and remove shift with element-wise max

Return: $\hat{V}, \hat{\Lambda}$

The NyströmPCG subroutine (Algorithm 9) is used in NNCG to compute the damped Newton step. The preconditioner P and its inverse P^{-1} are given by

$$P = \frac{1}{\hat{\lambda}_s + \mu} U(\hat{\Lambda} + \mu I)U^T + (I - UU^T)$$
$$P^{-1} = (\hat{\lambda}_s + \mu)U(\hat{\Lambda} + \mu I)^{-1}U^T + (I - UU^T).$$

Within NNCG, the matrix-vector product involving the Hessian (i.e., $A = H_L(w_k)$) is implemented using Hessian-vector products. For further details, please see Frangella et al. [2023].

Algorithm 9 NyströmPCG

Require: Psd matrix A, right-hand side b, initial guess x_0, approx. eigenvectors U, approx. eigenvalues $\hat{\Lambda}$, sketch size s, damping parameter μ, CG tolerance ϵ, CG max. iterations M

$r_0 = b - (A + \mu I)x_0$
$z_0 = P^{-1}r_0$
$p_0 = z_0$
$k = 0$ ▷ Iteration counter
while $\|r_0\|_2 \geq \varepsilon$ and $k < M$ **do**
  $v = (A + \mu I)p_0$
  $\alpha = (r_0^T z_0)/(p_0^T v_0)$ ▷ Compute step size
  $x = x_0 + \alpha p_0$ ▷ Update solution
  $r = r_0 - \alpha v$ ▷ Update residual
  $z = P^{-1}r$
  $\beta = (r^T z)/(r_0^T z_0)$
  $x_0 \leftarrow x, r_0 \leftarrow r, p_0 \leftarrow z + \beta p_0, z_0 \leftarrow z, k \leftarrow k + 1$
end while
Return: x

The Armijo subroutine (Algorithm 10) is used in NNCG to guarantee that the loss decreases at every iteration. The function oracle is implemented in PyTorch using a *closure*. At each iteration, the subroutine checks whether the *sufficient decrease condition* has been met; if not, it shrinks the step size by a factor of β. For further details, please see Nocedal and Wright [2006].

Algorithm 10 Armijo

Require: Function oracle f, current iterate x, current gradient $\nabla f(x)$, search direction d, initial step size t, backtracking parameters α, β

while $f(x + td) > f(x) + \alpha t(\nabla f(x)^T d)$ **do**
  $t \leftarrow \beta t$ ▷ Shrink step size
end while
Return: t

B.5.3 Wall-clock Times for L-BFGS and NNCG

Table B.1 summarizes the per-iteration wall-clock times of L-BFGS and NNCG on each PDE. The large gap on wave (compared to reaction and convection) is because NNCG has to compute hessian-vector products involving second derivatives, while this is not the case for the two other PDEs.

Table B.1: Per-iteration times (in seconds) of L-BFGS and NNCG on each PDE.

Optimizer	Convection	Reaction	Wave
L-BFGS	4.6e-2	3.6e-2	9.0e-2
NNCG	2.5e-1	7.2e-1	2.9e1
Time Ratio	5.43	20	322.22

B.6 Ill-Conditioned Differential Operators Lead to Difficult Optimization Problems

In this section, we state and prove the formal version of Theorem 3.4. The overall structure of the proof is based on showing the conditioning of the Gauss-Newton matrix of the population PINN loss is controlled by the conditioning of the differential operator. We then show the empirical Gauss-Newton matrix is close to its population counterpart by using matrix concentration techniques. Finally, as the conditioning of H_L at a minimizer is controlled by the empirical Gauss-Newton matrix, we obtain the desired result.

B.6.1 Preliminaries

Similar to De Ryck et al. [2023], we consider a general linear PDE with Dirichlet boundary conditions:

$$\begin{aligned} &\mathcal{D}[u](x) = f(x), \quad x \in \Omega, \\ &u(x) = g(x), \quad x \in \partial\Omega, \end{aligned}$$

where $u : \mathbb{R}^d \mapsto \mathbb{R}$, $f : \mathbb{R}^d \mapsto \mathbb{R}$ and Ω is a bounded subset of $\mathbb{R}^d$. The "population" PINN objective for this PDE is

$$L_\infty(w) = \frac{1}{2}\int_\Omega \left(\mathcal{D}[u(x;w)] - f(x)\right)^2 d\mu(x) + \frac{\lambda}{2}\int_{\partial\Omega} \left(u(x;w) - g(x)\right)^2 d\sigma(x).$$

λ can be any positive real number; we set $\lambda = 1$ in our experiments. Here μ and σ are probability measures on Ω and $\partial\Omega$ respectively, from which the data is sampled. The empirical PINN objective is given by

$$L(w) = \frac{1}{2n_{\text{res}}}\sum_{i=1}^{n_{\text{res}}} \left(\mathcal{D}[u(x_r^i;w)] - f(x_i)\right)^2 + \frac{\lambda}{2n_{\text{bc}}}\sum_{j=1}^{n_{\text{bc}}} \Big(u(x_b^j;w) - g(x_j)\Big)^2.$$

Moreover, throughout this section we use the notation $\langle f, g\rangle_{L^2(\Omega)}$ to denote the standard L^2-inner product on Ω:

$$\langle f, g\rangle_{L^2(\Omega)} = \int_\Omega f g d\mu(x).$$

Lemma B.1. *The Hessian of the $L_\infty(w)$ is given by*

$$H_{L_\infty}(w) = \int_\Omega \mathcal{D}[\nabla_w u(x;w)]\mathcal{D}[\nabla_w u(x;w)]^T d\mu(x) + \int_\Omega \mathcal{D}[\nabla_w^2 u(x;w)]\left(\mathcal{D}[\nabla_w u(x;w)] - f(x)\right) d\mu(x)$$
$$+ \lambda \int_{\partial\Omega} \nabla_w u(x;w)\nabla_w u(x;w)^T d\sigma(x) + \lambda \int_{\partial\Omega} \nabla_w^2 u(x;w)\left(u(x;w) - g(x)\right) d\sigma(x).$$

The Hessian of $L(w)$ is given by

$$H_L(w) = \frac{1}{n_{\text{res}}}\sum_{i=1}^{n_{\text{res}}} \mathcal{D}[\nabla_w u(x_r^i;w)]\mathcal{D}[\nabla_w u(x_r^i;w)]^T + \frac{1}{n_{\text{res}}}\sum_{i=1}^{n_{\text{res}}} \mathcal{D}[\nabla_w^2 u(x_i^r;w)]\left(\mathcal{D}[\nabla_w u(x_r^i;w)] - f(x_r^i)\right) \tag{B.3}$$
$$+ \frac{\lambda}{n_{\text{bc}}}\sum_{j=1}^{n_{\text{bc}}} \nabla_w u(x_b^j;w)\nabla_w u(x_b^j;w)^T + \frac{\lambda}{n_{\text{bc}}}\sum_{j=1}^{n_{\text{bc}}} \nabla_w^2 u(x_b^j;w)\left(u(x_b^j;w) - g(x_j)\right).$$

In particular, for $w_\star \in \mathcal{W}_\star$

$$H_L(w_\star) = G_r(w) + G_b(w).$$

Here

$$G_r(w) \coloneqq \frac{1}{n_{\text{res}}}\sum_{i=1}^{n_{\text{res}}} \mathcal{D}[\nabla_w u(x_i;w_\star)]\mathcal{D}[\nabla_w u(x_i;w_\star)]^T, \quad G_b(w) = \frac{\lambda}{n_{\text{bc}}}\sum_{j=1}^{n_{\text{bc}}} \nabla_w u(x_b^j;w_\star)\nabla_w u(x_b^j;w_\star)^T.$$

Define the maps $\mathcal{F}_{\text{res}}(w) = \begin{bmatrix} \mathcal{D}[u(x_r^1;w)] \\ \vdots \\ \mathcal{D}[u(x_r^{n_{\text{res}}};w)] \end{bmatrix}$, and $\mathcal{F}_{\text{bc}}(w) = \begin{bmatrix} u(x_b^1;w) \\ \vdots \\ u(x_b^{n_{\text{bc}}};w)] \end{bmatrix}$. We have the following important lemma, which follows via routine calculation.

Lemma B.2. *Let $n = n_{\text{res}} + n_{\text{bc}}$. Define the map $\mathcal{F} : \mathbb{R}^p \to \mathbb{R}^n$, by stacking $\mathcal{F}_{\text{res}}(w), \mathcal{F}_{\text{bc}}(w)$. Then, the Jacobian of $\mathcal{F}$ is given by*

$$J_{\mathcal{F}}(w) = \begin{bmatrix} J_{\mathcal{F}_{\text{res}}}(w) \\ J_{\mathcal{F}_{\text{bc}}}(w). \end{bmatrix}$$

Moreover, the tangent kernel $K_{\mathcal{F}}(w) = J_{\mathcal{F}}(w)J_{\mathcal{F}}(w)^T$ is given by

$$K_{\mathcal{F}}(w) = \begin{bmatrix} J_{\mathcal{F}_{\text{res}}}(w)J_{\mathcal{F}_{\text{res}}}(w)^T & J_{\mathcal{F}_{\text{res}}}(w)J_{\mathcal{F}_{\text{bc}}}(w)^T \\ J_{\mathcal{F}_{\text{bc}}}(w)J_{\mathcal{F}_{\text{res}}}(w)^T & J_{\mathcal{F}_{\text{bc}}}(w)J_{\mathcal{F}_{\text{bc}}}(w)^T \end{bmatrix} = \begin{bmatrix} K_{\mathcal{F}_{\text{res}}}(w) & J_{\mathcal{F}_{\text{res}}}(w)J_{\mathcal{F}_{\text{bc}}}(w)^T \\ J_{\mathcal{F}_{\text{bc}}}(w)J_{\mathcal{F}_{\text{res}}}(w)^T & K_{\mathcal{F}_{\text{bc}}}(w) \end{bmatrix}.$$

B.6.2 Relating $G_\infty(w)$ to $\mathcal{D}$

Isolate the population Gauss-Newton matrix for the residual:

$$G_\infty(w) = \int_\Omega \mathcal{D}[\nabla_w u(x;w)]\mathcal{D}[\nabla_w u(x;w)]^T d\mu(x).$$

Analogous to De Ryck et al. [2023] we define the functions $\phi_i(x;w) = \partial_{w_i} u(x;w)$ for $i \in \{1\dots,p\}$. From this and the definition of $G_\infty(w)$, it follows that $(G_\infty(w))_{ij} = \langle \mathcal{D}[\phi_i], \mathcal{D}[\phi_j]\rangle_{L^2(\Omega)}$.

Similar to De Ryck et al. [2023] we can associate each $w \in \mathbb{R}^p$ with a space of functions $\mathcal{H}(w) = \text{span}\,(\phi_1(x;w),\dots,\phi_p(x;w)) \subset L^2(\Omega)$. We also define two linear maps associated with $\mathcal{H}(w)$:

$$T(w)v = \sum_{i=1}^p v_i \phi_i(x;w),$$

$$T^*(w)f = \left(\langle f, \phi_1\rangle_{L^2(\Omega)}, \dots, \langle f, \phi_p\rangle_{L^2(\Omega)}\right).$$

From these definitions, we establish the following lemma.

Lemma B.3 (Characterizing $G_\infty(w)$)**.** *Define $\mathcal{A} = \mathcal{D}^*\mathcal{D}$. Then the matrix $G_\infty(w)$ satisfies*

$$G_\infty(w) = T^*(w)\mathcal{A}T(w).$$

Proof. Let e_i and e_j denote the ith and jth standard basis vectors in $\mathbb{R}^p$. Then,

$$\begin{aligned}(G_\infty(w))_{ij} &= \langle \mathcal{D}[\phi_i](w), \mathcal{D}[\phi_j](w)\rangle_{L^2(\Omega)} = \langle \phi_i(w), \mathcal{D}^*\mathcal{D}[\phi_j(w)]\rangle_{L^2(\Omega)} = \langle Te_i, \mathcal{D}^*\mathcal{D}[Te_j]\rangle_{L^2(\Omega)} \\ &= \langle e_i, (T^*\mathcal{D}^*\mathcal{D}T)[e_j]\rangle_{L^2(\Omega)},\end{aligned}$$

where the second equality follows from the definition of the adjoint. Hence, using $\mathcal{A} = \mathcal{D}^*\mathcal{D}$, we conclude $G_\infty(w) = T^*(w)\mathcal{A}T(w)$. □

Define the kernel integral operator $\mathcal{K}_\infty(w) : L^2(\Omega) \to \mathcal{H}$ by

$$\mathcal{K}_\infty(w)[f](x) = T(w)T^*(w)f = \sum_{i=1}^p \langle f, \phi_i(x;w)\rangle \phi_i(x;w), \tag{B.4}$$

and the kernel matrix $A(w)$ with entries $A_{ij}(w) = \langle \phi_i(x;w), \phi_j(x;w)\rangle_{L^2(\Omega)}$.

Using Theorem B.3 and applying the same logic as in the proof of Theorem 2.4 in De Ryck et al. [2023], we obtain the following theorem.

Theorem B.4. *Suppose that the matrix $A(w)$ is invertible. Then the eigenvalues of $G_\infty(w)$ satisfy*

$$\lambda_j(G_\infty(w)) = \lambda_j(\mathcal{A} \circ \mathcal{K}_\infty(w)), \quad \textit{for all } j \in [p].$$

B.6.3 $G_r(w)$ Concentrates Around $G_\infty(w)$

In order to relate the conditioning of the population objective to the empirical objective, we must relate the population Gauss-Newton residual matrix to its empirical counterpart. We accomplish this by showing $G_r(w)$ concentrates around $G_\infty(w)$. To this end, we recall the following variant of the intrinsic dimension matrix Bernstein inequality from Tropp [2015].

Theorem B.5 (Intrinsic Dimension Matrix Bernstein)**.** *Let $\{X_i\}_{i\in[n]}$ be a sequence of independent mean zero random matrices of the same size. Suppose that the following conditions hold:*

$$\|X_i\| \leq B,\ \sum_{i=1}^{n} \mathbb{E}[X_i X_i^T] \preceq V_1,\ \sum_{i=1}^{n} \mathbb{E}[X_i^T X_i] \preceq V_2.$$

Define

$$\mathcal{V} = \begin{bmatrix} V_1 & 0 \\ 0 & V_2 \end{bmatrix},\ \varsigma^2 = \max\{\|V_1\|, \|V_2\|\},$$

and the intrinsic dimension $d_{\text{int}} = \frac{\text{trace}(\mathcal{V})}{\|\mathcal{V}\|}$.
Then for all $t \geq \varsigma + \frac{B}{3}$,

$$\mathbb{P}\left(\left\|\sum_{i=1}^{n} X_i\right\| \geq t\right) \leq 4d_{\text{int}} \exp\left(-\frac{3}{8}\min\left\{\frac{t^2}{\varsigma^2}, \frac{t}{B}\right\}\right).$$

Next, we recall two key concepts from the kernel ridge regression literature and approximation via sampling literature: γ-*effective dimension* and γ-*ridge leverage coherence* [Bach, 2013, Cohen et al., 2017, Rudi et al., 2017].

Definition B.6 (γ-Effective dimension and γ-ridge leverage coherence)**.** Let $\gamma > 0$. Then the γ-effective dimension of $G_\infty(w)$ is given by

$$d_{\text{eff}}^{\gamma}(G_\infty(w)) = \text{trace}\left(G_\infty(w)(G_\infty(w) + \gamma I)^{-1}\right).$$

The γ-ridge leverage coherence is given by

$$\chi^{\gamma}(G_\infty(w)) = \sup_{x\in\Omega} \frac{\left\|(G_\infty(w) + \gamma I)^{-1/2}\mathcal{D}[\nabla_w u(x;w)]\right\|^2}{\mathbb{E}_{x\sim\mu}\left\|(G_\infty(w) + \gamma I)^{-1/2}\mathcal{D}[\nabla_w u(x;w)]\right\|^2} = \frac{\sup_{x\in\Omega}\left\|(G_\infty(w) + \gamma I)^{-1/2}\mathcal{D}[\nabla_w u(x;w)]\right\|^2}{d_{\text{eff}}^{\gamma}(G_\infty(w))}.$$

Observe that $d_{\text{eff}}^{\gamma}(G_\infty(w))$ only depends upon γ and w, while $\chi^{\gamma}(G_\infty(w))$ only depends upon γ, w, and Ω. Moreover, $\chi^{\gamma}(G_\infty(w)) < \infty$ as Ω is bounded.

We prove the following lemma using the γ-effective dimension and γ-ridge leverage coherence in conjunction with Theorem B.5.

Lemma B.7 (Finite-sample approximation)**.** *Let* $0 < \gamma < \lambda_1(G_\infty(w))$*. If* $n_{\text{res}} \geq 40\chi^\gamma(G_\infty(w))d^\gamma_{\text{eff}}(G_\infty(w))\log\left(\frac{8d^\gamma_{\text{eff}}(}{}\right.$ *then with probability at least* $1-\delta$

$$\frac{1}{2}\left[G_\infty(w) - \gamma I\right] \preceq G_r(w) \preceq \frac{1}{2}\left[3G_\infty(w) + \gamma I.\right]$$

Proof. Let $x_i = (G_\infty(w)+\gamma I)^{-1/2}\mathcal{D}[\nabla_w u(x_i; w)]$, and $X_i = \frac{1}{n_{\text{res}}}\left(x_i x_i^T - D_\gamma\right)$, where $D_\gamma = G_\infty(w)\left(G_\infty(w) + \gamma I\right)^{-1}$. Clearly, $\mathbb{E}[X_i] = 0$. Moreover, the X_i's are bounded as

$$\begin{aligned}\|X_i\| &= \max\left\{\frac{\lambda_{\max}(X_i)}{n_{\text{res}}}, -\frac{\lambda_{\min}(X_i)}{n_{\text{res}}}\right\} \leq \max\left\{\frac{\|x_i\|^2}{n_{\text{res}}}, \frac{\lambda_{\max}(-X_i)}{n_{\text{res}}}\right\} \leq \max\left\{\frac{\chi^\gamma(G_\infty(w))d^\gamma_{\text{eff}}(G_\infty(w))}{n_{\text{res}}}, \frac{1}{n_{\text{res}}}\right\} \\ &= \frac{\chi^\gamma(G_\infty(w))d^\gamma_{\text{eff}}(G_\infty(w))}{n_{\text{res}}}.\end{aligned}$$

Thus, it remains to verify the variance condition. We have

$$\begin{aligned}\sum_{i=1}^{n_{\text{res}}} \mathbb{E}[X_i X_i^T] = n_{\text{res}}\mathbb{E}[X_1^2] &= n_{\text{res}} \times \frac{1}{n_{\text{res}}^2}\mathbb{E}[(x_1 x_1^T - D_\gamma)^2] \preceq \frac{1}{n_{\text{res}}}\mathbb{E}[\|x_1\|^2 x_1 x_1^T] \\ &\preceq \frac{\chi^\gamma(G_\infty(w))d^\gamma_{\text{eff}}(G_\infty(w))}{n_{\text{res}}} D_\gamma.\end{aligned}$$

Hence, the conditions of Theorem B.5 hold with $B = \frac{\chi^\gamma(G_\infty(w))d^\gamma_{\text{eff}}(G_\infty(w))}{n_{\text{res}}}$ and $V_1 = V_2 = \frac{\chi^\gamma(G_\infty(w))d^\gamma_{\text{eff}}(G_\infty(w))}{n_{\text{res}}} D_\gamma$. Now $1/2 \leq \|\mathcal{V}\| \leq 1$ as $n_{\text{res}} \geq \chi^\gamma(G_\infty(w))d^\gamma_{\text{eff}}(G_\infty(w))$ and $\gamma \leq \lambda_1\left(G_\infty(w)\right)$. Moreover, as $V_1 = V_2$ we have $d_{\text{int}} \leq 4d^\gamma_{\text{eff}}(G_\infty(w))$. So, setting

$$t = \sqrt{\frac{8\chi^\gamma(G_\infty(w))d^\gamma_{\text{eff}}(G_\infty(w))\log\left(\frac{8d^\gamma_{\text{eff}}(G_\infty(w))}{\delta}\right)}{3n_{\text{res}}}} + \frac{8\chi^\gamma(G_\infty(w))d^\gamma_{\text{eff}}(G_\infty(w))\log\left(\frac{8d^\gamma_{\text{eff}}(G_\infty(w))}{\delta}\right)}{3n_{\text{res}}}$$

and using $n_{\text{res}} \geq 40\chi^\gamma(G_\infty(w))d^\gamma_{\text{eff}}(G_\infty(w))\log\left(\frac{8d^\gamma_{\text{eff}}(G_\infty(w))}{\delta}\right)$, we conclude

$$\mathbb{P}\left(\left\|\sum_{i=1}^{n_{\text{res}}} X_i\right\| \geq \frac{1}{2}\right) \leq \delta.$$

Now, $\|\sum_{i=1}^{n_{\text{res}}} X_i\| \leq \frac{1}{2}$ implies

$$-\frac{1}{2}\left[G_\infty(w) + \gamma I\right] \preceq G_r(w) - G_\infty(w) \preceq \frac{1}{2}\left[G_\infty(w) + \gamma I\right].$$

The claim now follows by rearrangement. □

By combining Theorem B.4 and Theorem B.7, we show that if the spectrum of $\mathcal{A}\circ\mathcal{K}_\infty(w)$ decays, then the spectrum of the empirical Gauss-Newton matrix also decays with high probability.

Proposition B.8 (Spectrum of empirical Gauss-Newton matrix decays fast)**.** *Suppose the eigenvalues of $\mathcal{A} \circ \mathcal{K}_\infty(w)$ satisfy $\lambda_j(\mathcal{A} \circ \mathcal{K}_\infty(w)) \leq Cj^{-2\alpha}$, where $\alpha > 1/2$ and $C > 0$ is some absolute constant. Then if $\sqrt{n_{\text{res}}} \geq 40C_1\chi^\gamma(G_\infty(w)) \log\left(\frac{1}{\delta}\right)$, for some absolute constant C_1, it holds that*

$$\lambda_{n_{\text{res}}}(G_r(w)) \leq n_{\text{res}}^{-\alpha}$$

with probability at least $1 - \delta$.

Proof. The hypotheses on the decay of the eigenvalues implies $d_{\text{eff}}^\gamma(G_\infty(w)) \leq C_1\gamma^{-\frac{1}{2\alpha}}$ (see Appendix C of Bach [2013]). Consequently, given $\gamma = n_{\text{res}}^{-\alpha}$, we have $d_{\text{eff}}^\gamma(G_\infty(w)) \leq C_1 n_{\text{res}}^{\frac{1}{2}}$. Combining this with our hypotheses on n_{res}, it follows $n_{\text{res}} \geq 40C_1\chi^\gamma(G_\infty(w))d_{\text{eff}}^\gamma(G_\infty(w)) \log\left(\frac{8d_{\text{eff}}^\gamma(G_\infty(w))}{\delta}\right)$. Hence Theorem B.7 implies with probability at least $1 - \delta$ that

$$G_r(w) \preceq \frac{1}{2}\left(3G_\infty(w) + \gamma I\right),$$

which yields for any $1 \leq r \leq n$

$$\lambda_{n_{\text{res}}}(G_r(w)) \leq \frac{1}{2}\left(3\lambda_r(G_\infty(w)) + \gamma\right).$$

Combining the last display with $n_{\text{res}} \geq 3d_{\text{eff}}^\gamma(G_\infty(w))$, Lemma 5.4 of Frangella et al. [2023] guarantees $\lambda_r(G_\infty(w)) \leq \gamma/3$, and so

$$\lambda_{n_{\text{res}}}(G_r(w)) \leq \frac{1}{2}\left(3\lambda_r(G_\infty(w)) + \gamma\right) \leq \gamma \leq n_{\text{res}}^{-\alpha}.$$

□

B.6.4 Formal Statement of Theorem 3.4 and Proof

Theorem B.9 (An ill-conditioned differential operator leads to hard optimization)**.** *Fix $w_\star \in \mathcal{W}_\star$, and let $\mathcal{S}$ be a set containing $w_\star$ for which $\mathcal{S}$ is μ-PŁ*. Let $\alpha > 1/2$. If the eigenvalues of $\mathcal{A} \circ \mathcal{K}_\infty(w_\star)$ satisfy $\lambda_j(\mathcal{A} \circ \mathcal{K}_\infty(w_\star)) \leq Cj^{-2\alpha}$ and $\sqrt{n_{\text{res}}} \geq 40C_1\chi^\gamma(G_\infty(w_\star)) \log\left(\frac{1}{\delta}\right)$, then*

$$\kappa_L(\mathcal{S}) \geq C_2 n_{\text{res}}^\alpha,$$

with probability at least $1 - \delta$. Here C, C_1, and C_2 are absolute constants.

Proof. By the assumption on n_{res}, the conditions of Theorem B.8 are met, so,

$$\lambda_{n_{\text{res}}}(G_r(w_\star)) \leq n_{\text{res}}^{-\alpha}.$$

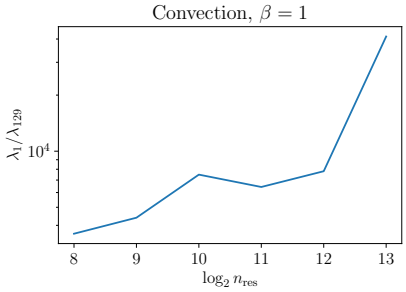


Figure B.5: Estimated condition number after 41000 iterations of Adam+L-BFGS with different number of residual points from 255×100 grid on the interior. Here λ_i denotes the ith largest eigenvalue of the Hessian. The model has 2 layers and the hidden layer has width 32. The plot shows κ_L grows polynomially in the number of residual points.

with probability at least $1 - \delta$. By definition $G_r(w_\star) = J_{\mathcal{F}_{\text{res}}}(w_\star)^T J_{\mathcal{F}_{\text{res}}}(w_\star)$, consequently,

$$\lambda_{n_{\text{res}}}(K_{\mathcal{F}_{\text{res}}}(w_\star)) = \lambda_{n_{\text{res}}}(G_r(w_\star)) \leq n_{\text{res}}^{-\alpha}.$$

Now, the PŁ*-constant for $\mathcal{S}$, satisfies $\mu = \inf_{w \in \mathcal{S}} \lambda_n(K_{\mathcal{F}}(w))$ [Liu et al., 2022]. Combining this with the expression for $K_{\mathcal{F}}(w_\star)$ in Theorem B.2, we reach

$$\mu \leq \lambda_n(K_{\mathcal{F}}(w_\star)) \leq \lambda_{n_{\text{res}}}(K_{\mathcal{F}_{\text{res}}}(w_\star)) \leq n_{\text{res}}^{-\alpha},$$

where the second inequality follows from Cauchy's Interlacing theorem. Recalling that $\kappa_L(\mathcal{S}) = \frac{\sup_{w \in \mathcal{S}} \|H_L(w)\|}{\mu}$, and $H_L(w_\star)$ is symmetric psd, we reach

$$\kappa_L(\mathcal{S}) \geq \frac{\lambda_1(H_L(w_\star))}{\mu} \overset{(1)}{\geq} \frac{\lambda_1(G_r(w_\star)) + \lambda_p(G_b(w_\star))}{\mu} \overset{(2)}{=} \frac{\lambda_1(G_r(w_\star))}{\mu} \overset{(3)}{\geq} C_3 \lambda_1(G_\infty(w_\star)) n_{\text{res}}^{\alpha}.$$

Here (1) uses $H_L(w_\star) = G_r(w_\star) + G_b(w_\star)$ and Weyl's inequalities, (2) uses $p \geq n_{\text{res}} + n_{\text{bc}}$, so that $\lambda_p(G_b(w_\star)) = 0$. Inequality (3) uses the upper bound on μ and the lower bound on $G_r(w)$ given in Theorem B.7. Hence, the claim follows with $C_2 = C_3 \lambda_1(G_\infty(w_\star))$. □

B.6.5 κ Grows with the Number of Residual Points

Figure B.5 plots the ratio $\lambda_1(H_L)/\lambda_{129}(H_L)$ near a minimizer $w_\star$. This ratio is a lower bound for the condition number of H_L, and is computationally tractable to compute. We see that the estimate of the κ grows polynomially with n_{res}, which provides empirical verification for Theorem 3.4.

B.7 Convergence of GDND (Algorithm 2)

In this section, we provide the formal version of Theorem 3.5 and its proof. However, this is delayed till Appendix B.7.4, as the theorem is a consequence of a series of results. Before jumping to the theorem, we recommend reading the statements in the preceding subsections to understand the statement and corresponding proof.

B.7.1 Overview and Notation

Recall, we are interested in minimizing the objective in (3.2):

$$L(w) = \frac{1}{2n_{\text{res}}} \sum_{i=1}^{n_{\text{res}}} \left(\mathcal{D}[u(x_r^i; w)]\right)^2 + \frac{1}{2n_{\text{bc}}} \sum_{j=1}^{n_{\text{bc}}} \Big(\mathcal{B}[u(x_b^j; w)]\Big)^2,$$

where $\mathcal{D}$ is the differential operator defining the PDE and $\mathcal{B}$ is the operator defining the boundary conditions. Define

$$\mathcal{F}(w) = \begin{bmatrix} \frac{1}{\sqrt{n_{\text{res}}}}\mathcal{D}[u(x_r^1; w)] \\ \vdots \\ \frac{1}{\sqrt{n_{\text{res}}}}\mathcal{D}[u(x_r^{n_{\text{res}}}; w)] \\ \frac{1}{\sqrt{n_{\text{bc}}}}\mathcal{B}[u(x_b^1; w)] \\ \vdots \\ \frac{1}{\sqrt{n_{\text{bc}}}}\mathcal{B}[u(x_b^{n_{\text{bc}}}; w)] \end{bmatrix}, \; y = 0$$

Using the preceding definitions, our objective may be rewritten as:

$$L(w) = \frac{1}{2}\|\mathcal{F}(w) - y\|^2.$$

Throughout the appendix, we work with the condensed expression for the loss given above. We denote the $(n_{\text{res}} + n_{\text{bc}}) \times p$ Jacobian matrix of $\mathcal{F}$ by $J_{\mathcal{F}}(w)$. The tangent kernel at w is given by the $n \times n$ matrix $K_{\mathcal{F}}(w) = J_{\mathcal{F}}(w)J_{\mathcal{F}}(w)^T$. The closely related Gauss-Newton matrix is given by $G(w) = J_{\mathcal{F}}(w)^T J_{\mathcal{F}}(w)$.

B.7.2 Global Behavior: Reaching a Small Ball About a Minimizer

We begin by showing that under appropriate conditions, gradient descent outputs a point close to a minimizer after a fixed number of iterations. We first start with the following assumption which is common in the neural network literature [Liu et al., 2022, 2023].

Assumption B.10. The mapping $\mathcal{F}(w)$ is $\mathcal{L}_{\mathcal{F}}$-Lipschitz, and the loss $L(w)$ is β_L-smooth.

Under Theorem B.10 and a PŁ*-condition, we have the following theorem of Liu et al. [2022], which shows gradient descent converges linearly.

Theorem B.11. *Let w_0 denote the network weights at initialization. Suppose Theorem B.10 holds, and that $L(w)$ is μ-PŁ* in $B(w_0, 2R)$ with $R = \frac{2\sqrt{2\beta_L L(w_0)}}{\mu}$. Then the following statements hold:*

1. *The intersection $B(w_0, R) \cap \mathcal{W}_\star$ is non-empty.*
2. *Gradient descent with step size $\eta = 1/\beta_L$ satisfies:*

$$w_{k+1} = w_k - \eta \nabla L(w_k) \in B(w_0, R) \text{ for all } k \geq 0,$$
$$L(w_k) \leq \left(1 - \frac{\mu}{\beta_L}\right)^k L(w_0).$$

For wide neural networks, it is known that the μ-PŁ*condition in Theorem B.11 hold with high probability, see Liu et al. [2022] for details.

We also recall the following lemma from Liu et al. [2023].

Lemma B.12 (Descent Principle)**.** *Let $L : \mathbb{R}^p \mapsto [0, \infty)$ be differentiable and μ-PŁ* in the ball $B(w, r)$. Suppose $L(w) < \frac{1}{2}\mu r^2$. Then the intersection $B(w, r) \cap \mathcal{W}_\star$ is non-empty, and*

$$\frac{\mu}{2}\text{dist}^2(w, \mathcal{W}_\star) \leq L(w).$$

Let $\mathcal{L}_{H_L}$ be the Hessian Lipschitz constant in $B(w_0, 2R)$, and $\mathcal{L}_{J_\mathcal{F}} = \sup_{w \in B(w_0, 2R)} \|H_\mathcal{F}(w)\|$, where $\|H_\mathcal{F}(w)\| = \max_{i \in [n]} \|H_{\mathcal{F}_i}(w)\|$. Define $M = \max\{\mathcal{L}_{H_L}, \mathcal{L}_{J_\mathcal{F}}, \mathcal{L}_\mathcal{F}\mathcal{L}_{J_\mathcal{F}}, 1\}$, $\varepsilon_{\text{loc}} = \frac{\varepsilon\mu^{3/2}}{4M}$, where $\varepsilon \in (0, 1)$. By combining Theorem B.11 and Theorem B.12, we are able to establish the following important corollary, which shows gradient descent outputs a point close to a minimizer.

Corollary B.13 (Getting close to a minimizer)**.** *Set $\rho = \min\left\{\frac{\varepsilon_{\text{loc}}}{19\sqrt{\frac{\beta_L}{\mu}}}, \sqrt{\mu}R, R\right\}$. Run gradient descent for $k = \frac{\beta_L}{\mu}\log\left(\frac{4\max\{2\beta_L, 1\}L(w_0)}{\mu\rho^2}\right)$ iterations, gradient descent outputs a point w_{loc} satisfying*

$$L(w_{\text{loc}}) \leq \frac{\mu\rho^2}{4}\min\left\{1, \frac{1}{2\beta_L}\right\},$$

$$\|w_{\text{loc}} - w_\star\|_{H_L(w_\star)+\mu I} \leq \rho, \text{ for some } w_\star \in \mathcal{W}_\star.$$

Proof. The first claim about $L(w_{\text{loc}})$ is an immediate consequence of Theorem B.11. For the second claim, consider the ball $B(w_{\text{loc}}, \rho)$. Observe that $B(w_{\text{loc}}, \rho) \subset B(w_0, 2R)$, so L is μ-PŁ* in $B(w_{\text{loc}}, \rho)$. Combining this with $L(w_{\text{loc}}) \leq \frac{\mu\rho^2}{4}$, Theorem B.12 guarantees the existence of $w_\star \in B(w_{\text{loc}}, \rho) \cap \mathcal{W}_\star$,

with $\|w_{\text{loc}} - w_\star\| \leq \sqrt{\frac{2}{\mu} L(w_{\text{loc}})}$. Hence Cauchy-Schwarz yields

$$\begin{aligned}\|w_{\text{loc}} - w_\star\|_{H_L(w_\star)+\mu I} &\leq \sqrt{\beta_L + \mu}\|w_{\text{loc}} - w_\star\| \leq \sqrt{2\beta_L}\|w_{\text{loc}} - w_\star\| \\ &\leq 2\sqrt{\frac{\beta_L}{\mu} L(w_{\text{loc}})} \leq 2 \times \sqrt{\frac{\beta_L}{\mu}\frac{\mu\rho^2}{8\beta_L}} \leq \rho,\end{aligned}$$

which proves the claim. □

B.7.3 Fast Local Convergence of Damped Newton's Method

In this section, we show damped Newton's method with fixed stepsize exhibits fast linear convergence in an appropriate region about the minimizer $w_\star$ from Theorem B.13. Fix $\varepsilon \in (0, 1)$, then the region of local convergence is given by:

$$\mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star) = \left\{w \in \mathbb{R}^p : \|w - w_\star\|_{H_L(w_\star)+\mu I} \leq \varepsilon_{\text{loc}}\right\},$$

where $\varepsilon_{\text{loc}} = \frac{\varepsilon\mu^{3/2}}{4M}$ as above. Note that $w_{\text{loc}} \in \mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star)$.

We now prove several lemmas, that are essential to the argument. We begin with the following elementary technical result, which shall be used repeatedly below.

Lemma B.14 (Sandwich lemma)**.** *Let A be a symmetric matrix and B be a symmetric positive-definite matrix. Suppose that A and B satisfy $\|A - B\| \leq \varepsilon\lambda_{\min}(B)$ where $\varepsilon \in (0, 1)$. Then*

$$(1 - \varepsilon)B \preceq A \preceq (1 + \varepsilon)B.$$

Proof. By hypothesis, it holds that

$$-\varepsilon\lambda_{\min}(B)I \preceq A - B \preceq \varepsilon\lambda_{\min}(B)I.$$

So using $B \succeq \lambda_{\min}(B)I$, and adding B to both sides, we reach

$$(1 - \varepsilon)B \preceq A \preceq (1 + \varepsilon)B.$$

□

The next result describes the behavior of the damped Hessian in $\mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star)$.

Lemma B.15 (Damped Hessian in $\mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star)$)**.** *Suppose that $\gamma \geq \mu$ and $\varepsilon \in (0, 1)$.*

1. *(Positive-definiteness of damped Hessian in $\mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star)$) For any $w \in \mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star)$,*

$$H_L(w) + \gamma I \succeq \left(1 - \frac{\varepsilon}{4}\right)\gamma I.$$

2. (Damped Hessians stay close in $\mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star)$) For any $w, w' \in \mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star)$,

$$(1-\varepsilon)\left[H_L(w)+\gamma I\right] \preceq H_L(w')+\gamma I \preceq (1+\varepsilon)\left[H_L(w)+\gamma I\right].$$

Proof. We begin by observing that the damped Hessian at $w_\star$ satisfies

$$\begin{aligned} H_L(w_\star)+\gamma I &= G(w_\star)+\gamma I + \frac{1}{n}\sum_{i=1}^{n}\left[\mathcal{F}(w_\star)-y\right]_i H_{\mathcal{F}_i}(w_\star) \\ &= G(w_\star)+\gamma I \succeq \gamma I. \end{aligned}$$

Thus, $H_L(w_\star)+\gamma I$ is positive definite. Now, for any $w \in \mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star)$, it follows from Lipschitzness of H_L that

$$\|(H_L(w)+\gamma I)-(H_L(w_\star)+\gamma I)\| \leq \mathcal{L}_{H_L}\|w-w_\star\| \leq \frac{\mathcal{L}_{H_L}}{\sqrt{\gamma}}\|w-w_\star\|_{H_L(w_\star)+\gamma I} \leq \frac{\varepsilon\mu}{4}.$$

As $\lambda_{\min}\left(H_L(w_\star)+\gamma I\right) \geq \gamma > \mu$, we may invoke Theorem B.14 to reach

$$\left(1-\frac{\varepsilon}{4}\right)\left[H_L(w_\star)+\gamma I\right] \preceq H_L(w)+\gamma I \preceq \left(1+\frac{\varepsilon}{4}\right)\left[H_L(w_\star)+\gamma I\right].$$

This immediately yields

$$\lambda_{\min}\left(H_L(w)+\gamma I\right) \geq \left(1-\frac{\varepsilon}{4}\right)\gamma \geq \frac{3}{4}\gamma,$$

which proves item 1. To see the second claim, observe for any $w, w' \in \mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star)$ the triangle inequality implies

$$\|(H_L(w')+\gamma I)-(H_L(w)+\gamma I)\| \leq \frac{\varepsilon\mu}{2} \leq \frac{2}{3}\varepsilon\left(\frac{3}{4}\gamma\right).$$

As $\lambda_{\min}\left(H_L(w)+\gamma I\right) \geq \frac{3}{4}\gamma$, it follows from Theorem B.14 that

$$\left(1-\frac{2}{3}\varepsilon\right)\left[H_L(w)+\gamma I\right] \preceq H_L(w')+\gamma I \preceq \left(1+\frac{2}{3}\varepsilon\right)\left[H_L(w)+\gamma I\right],$$

which establishes item 2. □

The next result characterizes the behavior of the tangent kernel and Gauss-Newton matrix in $\mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star)$.

Lemma B.16 (Tangent kernel and Gauss-Newton matrix in $\mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star)$)**.** *Let $\gamma \geq \mu$. Then for any $w, w' \in \mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star)$, the following statements hold:*

1. (Tangent kernels stay close)

$$\left(1-\frac{\varepsilon}{2}\right)K_{\mathcal{F}}(w_\star) \preceq K_{\mathcal{F}}(w) \preceq \left(1+\frac{\varepsilon}{2}\right)K_{\mathcal{F}}(w_\star)$$

2. *(Gauss-Newton matrices stay close)*

$$\left(1-\frac{\varepsilon}{2}\right)\left[G(w)+\gamma I\right] \preceq G(w_\star)+\gamma I \preceq \left(1+\frac{\varepsilon}{2}\right)\left[G(w)+\gamma I\right]$$

3. *(Damped Hessian is close to damped Gauss-Newton matrix)*

$$(1-\varepsilon)\left[G(w)+\gamma I\right] \preceq H_L(w)+\gamma I \preceq (1+\varepsilon)\left[G(w)+\gamma I\right].$$

4. *(Jacobian has full row-rank) The Jacobian satisfies* $\operatorname{rank}(J_{\mathcal{F}}(w)) = n$.

Proof. 1. Observe that

$$\begin{aligned}\|K_{\mathcal{F}}(w)-K_{\mathcal{F}}(w_\star)\| &= \|J_{\mathcal{F}}(w)J_{\mathcal{F}}(w)^T - J_{\mathcal{F}}(w_\star)J_{\mathcal{F}}(w_\star)^T\| \\ &= \left\|\left[J_{\mathcal{F}}(w)-J_{\mathcal{F}}(w_\star)\right]J_{\mathcal{F}}(w)^T + J_{\mathcal{F}}(w_\star)\left[J_{\mathcal{F}}(w)-J_{\mathcal{F}}(w_\star)\right]^T\right\| \\ &\leq 2\mathcal{L}_{\mathcal{F}}\mathcal{L}_{J_{\mathcal{F}}}\|w-w_\star\| \leq \frac{2\mathcal{L}_{\mathcal{F}}\mathcal{L}_{J_{\mathcal{F}}}}{\sqrt{\gamma}}\|w-w_\star\|_{H_L(w_\star)+\gamma I} \leq \frac{\varepsilon\mu^{3/2}}{\sqrt{\gamma}} \leq \frac{\varepsilon}{2}\mu,\end{aligned}$$

where in the first inequality we applied the fundamental theorem of calculus to reach

$$\|J_{\mathcal{F}}(w)-J_{\mathcal{F}}(w_\star)\| \leq \mathcal{L}_{J_{\mathcal{F}}}\|w-w_\star\|.$$

Hence the claim follows from Theorem B.14.

2. By an analogous argument to item 1, we find

$$\|(G(w)+\gamma I)-(G(w_\star)+\gamma I)\| \leq \frac{\varepsilon}{2}\mu,$$

so the result again follows from Theorem B.14.

3. First observe $H_L(w_\star)+\gamma I = G(w_\star)+\gamma I$. Hence the proof of Theorem B.15 implies,

$$\left(1-\frac{\varepsilon}{4}\right)\left[G(w_\star)+\gamma I\right] \preceq H_L(w)+\gamma I \preceq \left(1+\frac{\varepsilon}{4}\right)\left[G(w_\star)+\gamma I\right].$$

Hence the claim now follows from combining the last display with item 2.

4. This last claim follows immediately from item 1, as for any $w \in \mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star)$,

$$\sigma_n\left(J_{\mathcal{F}}(w)\right) = \sqrt{\lambda_{\min}(K_{\mathcal{F}}(w))} \geq \sqrt{\left(1-\frac{\varepsilon}{2}\right)\mu} > 0.$$

Here the last inequality uses $\lambda_{\min}(K_{\mathcal{F}}(w_\star)) \geq \mu$, which follows as $w_\star \in B(w_0, 2R)$.

□

The next lemma is essential to proving convergence. It shows in $\mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star)$ that $L(w)$ is uniformly smooth with respect to the damped Hessian, with nice smoothness constant $(1+\varepsilon)$. Moreover, it establishes that the loss is uniformly PŁ*with respect to the damped Hessian in $\mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star)$.

Lemma B.17 (Preconditioned smoothness and PŁ*)**.** *Suppose $\gamma \geq \mu$. Then for any $w, w', w'' \in \mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star)$, the following statements hold:*

1. $L(w'') \leq L(w') + \langle \nabla L(w'), w'' - w' \rangle + \frac{1+\varepsilon}{2}\|w'' - w'\|^2_{H_L(w)+\gamma I}$.
2. $\frac{\|\nabla L(w)\|^2_{(H_L(w)+\gamma I)^{-1}}}{2} \geq \frac{1}{1+\varepsilon}\frac{1}{(1+\gamma/\mu)}L(w)$.

Proof. 1. By Taylor's theorem

$$L(w'') = L(w') + \langle \nabla L(w'), w'' - w' \rangle + \int_0^1 (1-t)\|w'' - w'\|^2_{H_L(w'+t(w''-w'))}dt$$

Note $w' + t(w'' - w') \in \mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star)$ as $\mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star)$ is convex. Thus we have,

$$\begin{aligned} L(w'') &\leq L(w') + \langle \nabla L(w'), w'' - w' \rangle + \int_0^1 (1-t)\|w'' - w'\|^2_{H_L(w'+t(w''-w'))+\gamma I}dt \\ &\leq L(w') + \langle \nabla L(w'), w'' - w' \rangle + \int_0^1 (1-t)(1+\varepsilon)\|w'' - w'\|^2_{H_L(w)+\gamma I}dt \\ &= L(w') + \langle \nabla L(w'), w'' - w' \rangle + \frac{(1+\varepsilon)}{2}\|w'' - w'\|^2_{H_L(w)+\gamma I}. \end{aligned}$$

2. Observe that

$$\frac{\|\nabla L(w)\|^2_{(H_L(w)+\gamma I)^{-1}}}{2} = \frac{1}{2}(\mathcal{F}(w) - y)^T \left[J_\mathcal{F}(w)\left(H_L(w) + \gamma I\right)^{-1} J_\mathcal{F}(w)^T\right](\mathcal{F}(w) - y).$$

Now,

$$\begin{aligned} J_\mathcal{F}(w)\left(H_L(w) + \gamma I\right)^{-1} J_\mathcal{F}(w)^T &\succeq \frac{1}{(1+\varepsilon)} J_\mathcal{F}(w)\left(G(w) + \gamma I\right)^{-1} J_\mathcal{F}(w)^T \\ &= \frac{1}{(1+\varepsilon)} J_\mathcal{F}(w)\left(J_\mathcal{F}(w)^T J_\mathcal{F}(w) + \gamma I\right)^{-1} J_\mathcal{F}(w)^T \end{aligned}$$

Theorem B.16 guarantees $J_\mathcal{F}(w)$ has full row-rank, so the SVD yields

$$J_\mathcal{F}(w)\left(J_\mathcal{F}(w)^T J_\mathcal{F}(w) + \gamma I\right)^{-1} J_\mathcal{F}(w)^T = U\Sigma^2(\Sigma^2 + \gamma I)^{-1}U^T \succeq \frac{\mu}{\mu+\gamma} I.$$

Hence

$$\frac{\|\nabla L(w)\|^2_{(H_L(w)+\gamma I)^{-1}}}{2} \geq \frac{\mu}{(1+\varepsilon)(\mu+\gamma)}\frac{1}{2}\|\mathcal{F}(w) - y\|^2 = \frac{\mu}{(1+\varepsilon)(\mu+\gamma)}L(w).$$

□

Lemma B.18 (Local preconditioned-descent)**.** *Run Phase II of Algorithm 2 with* $\eta_{\mathrm{DN}} = (1+\varepsilon)^{-1}$ *and* $\gamma = \mu$*. Suppose that* $\tilde{w}_k, \tilde{w}_{k+1} \in \mathcal{N}_{\varepsilon_{\mathrm{loc}}}(w_\star)$*, then*

$$L(\tilde{w}_{k+1}) \leq \left(1 - \frac{1}{2(1+\varepsilon)^2}\right) L(\tilde{w}_k).$$

Proof. As $\tilde{w}_k, \tilde{w}_{k+1} \in \mathcal{N}_{\varepsilon_{\mathrm{loc}}}(w_\star)$, item 1 of Theorem B.17 yields

$$L(\tilde{w}_{k+1}) \leq L(\tilde{w}_k) - \frac{\|\nabla L(\tilde{w}_k)\|^2_{(H_L(\tilde{w}_k)+\mu I)^{-1}}}{2(1+\varepsilon)}.$$

Combining the last display with the preconditioned PŁ*condition, we conclude

$$L(\tilde{w}_{k+1}) \leq \left(1 - \frac{1}{2(1+\varepsilon)^2}\right) L(\tilde{w}_k).$$

□

The following lemma describes how far an iterate moves after one-step of Phase II of Algorithm 2.

Lemma B.19 (1-step evolution)**.** *Run Phase II of Algorithm 2 with* $\eta_{\mathrm{DN}} = (1+\varepsilon)^{-1}$ *and* $\gamma \geq \mu$*. Suppose* $\tilde{w}_k \in \mathcal{N}_{\frac{\varepsilon}{3}}(w_\star)$*, then* $\tilde{w}_{k+1} \in \mathcal{N}_{\varepsilon_{\mathrm{loc}}}(w_\star)$*.*

Proof. Let $P = H_L(\tilde{w}_k) + \gamma I$. We begin by observing that

$$\|\tilde{w}_{k+1} - w_\star\|_{H_L(w_\star)+\mu I} \leq \sqrt{1+\varepsilon}\|\tilde{w}_{k+1} - w_\star\|_P.$$

Now,

$$\begin{aligned}
\|\tilde{w}_{k+1} - w_\star\|_P &= \frac{1}{1+\varepsilon}\|\nabla L(\tilde{w}_k) - \nabla L(w_\star) - (1+\varepsilon)P(w_\star - \tilde{w}_k)\|_{P^{-1}} \\
&= \frac{1}{1+\varepsilon}\left\|\int_0^1 \left[\nabla^2 L(w_\star + t(w_k - w_\star)) - (1+\varepsilon)P\right] dt(w_\star - \tilde{w}_k)\right\|_{P^{-1}} \\
&= \frac{1}{1+\varepsilon}\left\|\int_0^1 \left[P^{-1/2}\nabla^2 L(w_\star + t(w_k - w_\star))P^{-1/2} - (1+\varepsilon)I\right] dt P^{1/2}(w_\star - \tilde{w}_k)\right\| \\
&\leq \frac{1}{1+\varepsilon}\int_0^1 \left\|P^{-1/2}\nabla^2 L(w_\star + t(w_k - w_\star))P^{-1/2} - (1+\varepsilon)I\right\| dt \|\tilde{w}_k - w_\star\|_P
\end{aligned}$$

We now analyze the matrix $P^{-1/2}\nabla^2 L(w_\star + t(w_k - w_\star))P^{-1/2}$. Observe that

$$\begin{aligned}
&P^{-1/2}\nabla^2 L(w_\star + t(w_k - w_\star))P^{-1/2} = P^{-1/2}(\nabla^2 L(w_\star + t(w_k - w_\star)) + \gamma I - \gamma I)P^{-1/2} \\
&= P^{-1/2}(\nabla^2 L(w_\star + t(w_k - w_\star)) + \gamma I)P^{-1/2} - \gamma P^{-1} \succeq (1-\varepsilon)I - \gamma P^{-1} \succeq -\varepsilon I.
\end{aligned}$$

Moreover,

$$P^{-1/2}\nabla^2 L(w_\star + t(w_k - w_\star))P^{-1/2} \preceq P^{-1/2}(\nabla^2 L(w_\star + t(w_k - w_\star)) + \gamma I)P^{-1/2} \preceq (1+\varepsilon)I.$$

Hence,

$$0 \preceq (1+\varepsilon)I - P^{-1/2}\nabla^2 L(w_\star + t(w_k - w_\star))P^{-1/2} \preceq (1+2\varepsilon)I,$$

and so

$$\|\tilde{w}^{k+1} - w_\star\|_P \leq \frac{1+2\varepsilon}{1+\varepsilon}\|\tilde{w}_k - w_\star\|_P.$$

Thus,

$$\|\tilde{w}^{k+1} - w_\star\|_{H_L(w_\star)+\mu I} \leq \frac{1+2\varepsilon}{\sqrt{1+\varepsilon}}\|\tilde{w}_k - w_\star\|_P \leq (1+2\varepsilon)\|\tilde{w}_k - w_\star\|_{H_L(w_\star)+\mu I} \leq \varepsilon_{\rm loc}.$$

□

The following lemma is key to establishing fast local convergence; it shows that the iterates produced by damped Newton's method remain in $\mathcal{N}_{\varepsilon_{\rm loc}}(w_\star)$, the region of local convergence.

Lemma B.20 (Staying in $\mathcal{N}_{\varepsilon_{\rm loc}}(w_\star)$)**.** *Suppose that $w_{\rm loc} \in \mathcal{N}_\rho(w_\star)$, where $\rho = \frac{\varepsilon_{\rm loc}}{19\sqrt{\beta_L/\mu}}$. Run Phase II of Algorithm 2 with $\gamma = \mu$ and $\eta = (1+\varepsilon)^{-1}$, then $\tilde{w}_{k+1} \in \mathcal{N}_{\varepsilon_{\rm loc}}(w_\star)$ for all $k \geq 1$.*

Proof. In the argument that follows $\kappa_P = 2(1+\varepsilon)^2$. The proof is via induction. Observe that if $w_{\rm loc} \in \mathcal{N}_\varrho(w_\star)$ then by Theorem B.19, $\tilde{w}_1 \in \mathcal{N}_{\varepsilon_{\rm loc}}(w_\star)$. Now assume $\tilde{w}_j \in \mathcal{N}_{\varepsilon_{\rm loc}}(w_\star)$ for $j = 2, \ldots, k$. We shall show $\tilde{w}_{k+1} \in \mathcal{N}_{\varepsilon_{\rm loc}}(w_\star)$. To this end, observe that

$$\|\tilde{w}_{k+1} - w_\star\|_{H_L(w_\star)+\mu I} \leq \|w_{\rm loc} - w_\star\|_{H_L(w_\star)+\mu I} + \frac{1}{1+\varepsilon}\sum_{j=1}^{k}\|\nabla L(w_j)\|_{(H_L(w_\star)+\mu I)^{-1}}$$

Now,

$$\begin{aligned}\|\nabla L(w_j)\|_{(H_L(w_\star)+\mu I)^{-1}} &\leq \frac{1}{\sqrt{\mu}}\|\nabla L(w_j)\|_2 \leq \sqrt{\frac{2\beta_L}{\mu}L(w_j)} \\ &\leq \sqrt{\frac{2\beta_L}{\mu}}\left(1 - \frac{1}{\kappa_P}\right)^{j/2}\sqrt{L(w_{\rm loc})},\end{aligned}$$

Here the second inequality follows from $\|\nabla L(w)\| \leq \sqrt{2\beta_L L(w)}$, and the last inequality follows from

Theorem B.18, which is applicable as $\tilde{w}_0, \ldots, \tilde{w}_k \in \mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star)$. Thus,

$$\begin{aligned}
\|\tilde{w}_{k+1} - w_\star\|_{H_L(w_\star)+\mu I} &\leq \rho + \sqrt{\frac{2\beta_L}{\mu}} \sum_{j=1}^{k} \left(1 - \frac{1}{\kappa_P}\right)^{j/2} \sqrt{L(\tilde{w}_0)} \\
&\leq \rho + \sqrt{\frac{(1+\varepsilon)\beta_L}{2\mu}} \|w_{\text{loc}} - w_\star\|_{H_L(w_\star)+\mu I} \sum_{j=1}^{k} \left(1 - \frac{1}{\kappa_P}\right)^{j/2} \\
&\leq \left(1 + \sqrt{\frac{\beta_L}{\mu}} \sum_{j=0}^{\infty} \left(1 - \frac{1}{\kappa_P}\right)^{j/2}\right) \rho \\
&= \left(1 + \frac{\sqrt{\beta_L/\mu}}{1 - \sqrt{1 - \frac{1}{\kappa_P}}}\right) \rho \leq \varepsilon_{\text{loc}}.
\end{aligned}$$

Here, in the second inequality we have used $L(\tilde{w}_0) \leq 2(1+\varepsilon)\|w_{\text{loc}} - w_\star\|^2_{H_L(w_\star)+\mu I}$, which is an immediate consequence of Theorem B.17. Hence, $\tilde{w}_{k+1} \in \mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star)$, and the desired claim follows by induction. □

Theorem B.21 (Fast-local convergence of Damped Newton)**.** *Let w_{loc} be as in Theorem B.13. Consider the iteration*

$$\tilde{w}_{k+1} = \tilde{w}_k - \frac{1}{1+\varepsilon}(H_L(\tilde{w}_k) + \mu I)^{-1} \nabla L(\tilde{w}_k), \quad \text{where } \tilde{w}_0 = w_{\text{loc}}.$$

Then, after k iterations, the loss satisfies

$$L(\tilde{w}_k) \leq \left(1 - \frac{1}{2(1+\varepsilon)^2}\right)^k L(w_{\text{loc}}).$$

Thus after $k = \mathcal{O}\left(\log\left(\frac{1}{\epsilon}\right)\right)$ iterations

$$L(\tilde{w}_k) \leq \epsilon.$$

Proof. Theorem B.20 ensure that $\tilde{w}^k \in \mathcal{N}_{\varepsilon_{\text{loc}}}(w_\star)$ for all k. Thus, we can apply item 1 of Theorem B.17 and the definition of $\tilde{w}^{k+1}$, to reach

$$L(\tilde{w}_{k+1}) \leq L(\tilde{w}_k) - \frac{1}{2(1+\varepsilon)} \|\nabla L(\tilde{w}_k)\|^2_{P^{-1}}.$$

Now, using item 2 of Theorem B.17 and recursing yields

$$L(\tilde{w}_{k+1}) \leq \left(1 - \frac{1}{2(1+\varepsilon)^2}\right) L(\tilde{w}_k) \leq \left(1 - \frac{1}{2(1+\varepsilon)^2}\right)^{k+1} L(w_{\text{loc}}).$$

The remaining portion of the theorem now follows via a routine calculation. □

B.7.4 Formal Convergence of Algorithm 2

Here, we state and prove the formal convergence result for Algorithm 2.

Theorem B.22. *Suppose that Theorem 3.1 and Theorem B.10 hold, and that the loss is μ-PŁ* in $B(w_0, 2R)$, where $R = \frac{2\sqrt{2\beta_L L(w_0)}}{\mu}$. Let ε_{loc} and ρ be as in Theorem B.13, and set $\varepsilon = 1/6$ in the definition of ε_{loc}. Run Algorithm 2 with parameters: $\eta_{\text{GD}} = 1/\beta_L, K_{\text{GD}} = \frac{\beta_L}{\mu} \log\left(\frac{4\max\{2\beta_L,1\}L(w_0)}{\mu\rho^2}\right), \eta_{\text{DN}} = 5/6, \gamma = \mu$ and $K_{\text{DN}} \geq 1$. Then Phase II of Algorithm 2 satisfies*

$$L(\tilde{w}_k) \leq \left(\frac{2}{3}\right)^k L(w_{K_{\text{GD}}}).$$

Hence after $K_{\text{DN}} \geq 3\log\left(\frac{L(w_{K_{\text{GD}}})}{\epsilon}\right)$ iterations, Phase II of Algorithm 2 outputs a point satisfying

$$L(\tilde{w}_{K_{\text{DN}}}) \leq \epsilon.$$

Proof. By assumption the conditions of Theorem B.13 are met, therefore $w_{K_{\text{GD}}}$ satisfies $\|w_{K_{\text{GD}}} - w_\star\|_{H_L(w_\star)+\mu I} \leq \rho$, for some $w_\star \in \mathcal{W}_\star$. Hence, we may invoke Theorem B.21 to conclude the desired result. □

Appendix C

Supplementary Material for Chapter 4

C.1 Solver Configuration and Termination Criteria

This appendix provides a complete reference for the configuration parameters and termination criteria of each solver in `rlaopt`. All configuration classes use sensible defaults, allowing users to get started with minimal tuning.

C.1.1 NyströmPCG Configuration

Solver configuration (`PCGConfig`). Table C.1 lists the parameters of `PCGConfig`.

Stopping criteria (`PCGStoppingCriteria`). Table C.2 lists the termination parameters for NyströmPCG. The solver terminates when the relative residual satisfies

$$\|r_k\|_2 \leq \texttt{tol} \cdot \|b\|_2,$$

where $r_k = b - Ax_k$ is the residual at iteration k, or when the iteration count reaches `max_iters`.

C.1.2 NysADMM Configuration

Solver configuration (`ADMMConfig`). Table C.3 lists the parameters of `ADMMConfig`. The NysADMM solver is accessed via the `ADMM` class in the API.

Stopping criteria (`ADMMStoppingCriteria`). Table C.4 lists the termination parameters for NysADMM. The solver terminates when both the primal and dual residuals at iteration k fall

Parameter	Type	Default	Description
`preconditioner_config`	`PreconditionerConfig`	`IdentityConfig()`	Preconditioner strategy.

Table C.1: `PCGConfig` parameters.

Parameter	Type	Default	Description
`max_iters`	`int`	1000	Maximum number of iterations.
`tol`	`float`	10^{-6}	Relative tolerance for convergence.

Table C.2: `PCGStoppingCriteria` parameters.

below their respective tolerances,

$$\|r_k^{\text{pri}}\| \le \epsilon_k^{\text{pri}} \quad \text{and} \quad \|r_k^{\text{dual}}\| \le \epsilon_k^{\text{dual}},$$

where the tolerances are computed as

$$\epsilon_k^{\text{pri}} = \epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max\big(\|Ax_k\|,\, \|z_k\|,\, \|b\|\big), \qquad \epsilon_k^{\text{dual}} = \epsilon_{\text{abs}} + \epsilon_{\text{rel}} \|A^\top u_k\|,$$

following the standard ADMM convergence criterion [Boyd et al., 2011], or when the iteration count reaches `max_iters`.

C.1.3 Proximal Gradient Configuration

Solver configuration (`ProxGradConfig`). Table C.5 lists the parameters of `ProxGradConfig`.

Stopping criteria (`ProxGradStoppingCriteria`). Table C.6 lists the termination parameters for the proximal gradient solver. The solver terminates when the proximal gradient mapping norm satisfies

$$\texttt{err}_k = \frac{1}{\eta} \|x_k - \mathbf{prox}_{\eta g}(x_k - \eta \nabla f(x_k))\| \le \texttt{tol} \cdot \sqrt{d},$$

where η is the step size, d is the total number of decision variables, and $\mathbf{prox}_{\eta g}$ denotes the proximal operator of ηg. This quantity measures the magnitude of the proximal gradient step, which vanishes at a first-order optimal point. The solver also terminates when the iteration count reaches `max_iters`.

C.1.4 Nyström Preconditioner Configuration

Preconditioner configuration (`NystromConfig`). Table C.7 lists the parameters of `NystromConfig`, which controls the randomized Nyström preconditioner used by NyströmPCG and NysADMM.

Parameter	Type	Default	Description
`rho`	`float`	1.0	Augmented Lagrangian penalty parameter.
`rho_update_factor`	`float`	2.0	Multiplicative factor for updating ρ during primal-dual balancing.
`rho_update_threshold`	`float`	10.0	Threshold ratio of primal to dual residual that triggers a ρ update.
`rho_update_freq`	`int`	25	Frequency (in iterations) for checking and updating ρ.
`alpha`	`float`	1.6	Over-relaxation parameter ($0 < \alpha < 2$).
`sigma`	`float`	10^{-6}	Regularization for the inexact linear system solve.
`gamma`	`float`	1.2	Exponent controlling the decay of the linear system solve tolerance (> 1).
`preconditioner_config`	`PreconditionerConfig`	`NystromConfig`	Preconditioner for the linear system subproblem. Default: Nyström with rank 50.
`preconditioner_update_freq`	`int`	20	Frequency (in iterations) for updating the preconditioner.

Table C.3: `ADMMConfig` parameters.

Parameter	Type	Default	Description
`max_iters`	`int`	1000	Maximum number of iterations.
`eps_abs`	`float`	10^{-4}	Absolute tolerance for primal and dual residuals.
`eps_rel`	`float`	10^{-4}	Relative tolerance for primal and dual residuals.

Table C.4: `ADMMStoppingCriteria` parameters.

Parameter	Type	Default	Description
`eta`	`float`	1.0	Step size for the gradient update.
`use_acceleration`	`bool`	`False`	Whether to use Nesterov acceleration.
`use_linesearch`	`bool`	`True`	Whether to use backtracking line search for step size selection.

Table C.5: `ProxGradConfig` parameters.

Parameter	Type	Default	Description
`max_iters`	`int`	1000	Maximum number of iterations.
`tol`	`float`	10^{-4}	Tolerance for convergence based on the error metric.

Table C.6: `ProxGradStoppingCriteria` parameters.

Parameter	Type	Default	Description
`rank_init`	`int`	(required)	Initial rank of the Nyström approximation.
`rank_max`	`int` or `None`	`None`	Maximum allowable rank. Defaults to `rank_init` if not specified.
`num_power_iters`	`int`	10	Number of power iterations for error estimation in rank adaptation.
`error_tolerance`	`float`	10^{-2}	Error tolerance for rank adaptation.
`base_damping`	`float`	(required)	Base damping parameter ρ for the preconditioner $(\hat{H} + \rho I)^{-1}$.
`damping_mode`	`str`	`"adaptive"`	`"adaptive"`: adjusts damping based on smallest eigenvalue. `"non_adaptive"`: uses `base_damping` only.

Table C.7: `NystromConfig` parameters.

Bibliography

Amirhesam Abedsoltan, Mikhail Belkin, and Parthe Pandit. Toward large kernel models. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J. Zico Kolter. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*, 2019.

Ahmed Alaoui and Michael W Mahoney. Fast randomized kernel ridge regression with statistical guarantees. In *Advances in Neural Information Processing Systems*, 2015.

Zeyuan Allen-Zhu. Katyusha: The first direct acceleration of stochastic gradient methods. *Journal of Machine Learning Research*, 18(221):1–51, 2018.

Zeyuan Allen-Zhu and Yuanzhi Li. What Can ResNet Learn Efficiently, Going Beyond Kernels? In *Advances in Neural Information Processing Systems*, 2019.

Zeyuan Allen-Zhu, Zheng Qu, Peter Richtarik, and Yang Yuan. Even faster accelerated coordinate descent using non-uniform sampling. In *Proceedings of The 33rd International Conference on Machine Learning*, 2016.

Nima Anari and Michał Dereziński. Isotropy and log-concave polynomials: Accelerated sampling and high-precision counting of matroid bases. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, 2020.

Nima Anari, Yang P Liu, and Thuy-Duong Vuong. Optimal sublinear sampling of spanning trees and determinantal point processes via average-case entropic independence. *SIAM Journal on Computing*, pages FOCS22–93, 2024.

Kimon Antonakopoulos, Panayotis Mertikopoulos, Georgios Piliouras, and Xiao Wang. AdaGrad Avoids Saddle Points. In *Proceedings of the 39th International Conference on Machine Learning*, 2022.

David Applegate, Mateo Diaz, Oliver Hinder, Haihao Lu, Miles Lubin, Brendan O' Donoghue, and Warren Schudy. Practical large-scale linear programming using primal-dual hybrid gradient. In *Advances in Neural Information Processing Systems*, 2021.

Yossi Arjevani, Ohad Shamir, and Ron Shiff. Oracle complexity of second-order methods for smooth convex optimization. *Mathematical Programming*, 178:327–360, 2019.

Haim Avron, Kenneth L Clarkson, and David P Woodruff. Faster kernel ridge regression using sketching and preconditioning. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1116–1138, 2017.

Francis Bach. Sharp analysis of low-rank kernel matrix approximations. In *Conference on Learning Theory*, 2013.

Pau Batlle, Matthieu Darcy, Bamdad Hosseini, and Houman Owhadi. Kernel methods are competitive for operator learning. *Journal of Computational Physics*, 496:112549, 2024.

Mikhail Belkin. Approximation beats concentration? An approximation view on inference with smooth radial kernels. In *Conference On Learning Theory*, 2018.

Mikhail Belkin. Fit without fear: remarkable mathematical phenomena of deep learning through the prism of interpolation. *Acta Numerica*, 30:203–248, 2021.

Rajendra Bhatia. *Matrix analysis*, volume 169. Springer Science & Business Media, 2013.

Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-Lopez, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and modular implicit differentiation. In *Advances in Neural Information Processing Systems*, 2022.

Stefan Blücher, Klaus-Robert Müller, and Stefan Chmiela. Reconstructing kernel-based machine learning force fields with superlinear convergence. *Journal of Chemical Theory and Computation*, 19(14):4619–4630, 2023.

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

C. G. Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

Andrea Caponnetto and Ernesto DeVito. Optimal rates for the regularized least-squares algorithm. *Foundations of Computational Mathematics*, 7:331–368, 2007.

Coralia Cartis, IM Nicholas Gould, and Ph L Toint. On the complexity of steepest descent, Newton's and regularized Newton's methods for nonconvex unconstrained optimization problems. *SIAM Journal on Optimization*, 20(6):2833–2852, 2010.

Benjamin Charlier, Jean Feydy, Joan Alexis Glaunes, François-David Collin, and Ghislain Durif. Kernel operations on the GPU, with autodiff, without memory overflows. *Journal of Machine Learning Research*, 22(74):1–6, 2021.

Yifan Chen, Bamdad Hosseini, Houman Owhadi, and Andrew M. Stuart. Solving and learning nonlinear PDEs with Gaussian processes. *Journal of Computational Physics*, 447:110668, 2021.

Yifan Chen, Ethan N. Epperly, Joel A. Tropp, and Robert J. Webber. Randomly pivoted cholesky: Practical approximation of a kernel matrix with few entry evaluations. *Communications on Pure and Applied Mathematics*, 78(5):995–1041, 2025a.

Yuwen Chen, Danny Tse, Parth Nobel, Paul Goulart, and Stephen Boyd. Cuclarabel: Gpu acceleration for a conic optimization solver. *arXiv preprint arXiv:2412.19027*, 2025b.

Li-Fang Cheng, Bianca Dumitrascu, Gregory Darnell, Corey Chivers, Michael Draugelis, Kai Li, and Barbara E. Engelhardt. Sparse multi-output Gaussian processes for online medical time series prediction. *BMC Medical Informatics and Decision Making*, 20(1):152, 2020.

Lénaïc Chizat, Edouard Oyallon, and Francis Bach. On Lazy Training in Differentiable Programming. In *Advances in Neural Information Processing Systems*, 2019.

Michael B Cohen, Cameron Musco, and Christopher Musco. Input sparsity time low-rank approximation via ridge leverage score sampling. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2017.

Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific Machine Learning Through Physics–Informed Neural Networks: Where We Are and What's Next. *J. Sci. Comput.*, 92(3), 2022.

Kurt Cutajar, Michael Osborne, John Cunningham, and Maurizio Filippone. Preconditioning kernel matrices. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems*, 2014.

Tim De Ryck, Samuel Lanthaler, and Siddhartha Mishra. On the approximation of functions by tanh neural networks. *Neural Networks*, 143:732–750, 2021.

Tim De Ryck, Florent Bonnet, Siddhartha Mishra, and Emmanuel de Bézenac. An operator preconditioning perspective on training in physics-informed machine learning. *arXiv preprint arXiv:2310.05801*, 2023.

Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems*, 2014.

Alexandre Défossez, Leon Bottou, Francis Bach, and Nicolas Usunier. A simple convergence proof of Adam and Adagrad. *Transactions on Machine Learning Research*, 2022.

Michał Dereziński and Michael W Mahoney. Determinantal point processes in randomized numerical linear algebra. *Notices of the American Mathematical Society*, 68(1):34–45, 2021.

Michał Dereziński and Jiaming Yang. Solving dense linear systems faster than via preconditioning. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 1118–1129, 2024.

Michał Dereziński, Daniele Calandriello, and Michal Valko. Exact sampling of determinantal point processes with sublinear time preprocessing. In *Advances in Neural Information Processing Systems*, 2019.

Michał Dereziński, Rajiv Khanna, and Michael W Mahoney. Improved guarantees and a multiple-descent curve for column subset selection and the Nyström method. In *Advances in Neural Information Processing Systems*, 2020.

Michał Dereziński, Daniel LeJeune, Deanna Needell, and Elizaveta Rebrova. Fine-grained analysis and faster algorithms for iteratively solving linear systems. *Journal of Machine Learning Research*, 26(144):1–49, 2025a.

Michał Dereziński, Christopher Musco, and Jiaming Yang. Faster linear systems and matrix norm approximation via multi-level sketched preconditioning. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2025b.

Michał Dereziński, Deanna Needell, Elizaveta Rebrova, and Jiaming Yang. Randomized Kaczmarz methods with beyond-Krylov convergence. *arXiv preprint arXiv:2501.11673*, 2025c.

Theo Diamandis, Zachary Frangella, Shipu Zhao, Bartolomeo Stellato, and Madeleine Udell. GeNIOS: an (almost) second-order operator-splitting solver for large-scale convex optimization. *Mathematical Programming Computation*, 2026.

Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

Mateo Díaz, Ethan N Epperly, Zachary Frangella, Joel A Tropp, and Robert J Webber. Robust, randomized preconditioning for kernel ridge regression. *arXiv preprint arXiv:2304.12465*, 2023.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

Weinan E and Bing Yu. The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

Ethan N. Epperly, Joel A. Tropp, and Robert J. Webber. Embrace rejection: Kernel matrix approximation by accelerated randomly pivoted cholesky. *SIAM Journal on Matrix Analysis and Applications*, 46(4):2527–2557, 2025.

R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, 1970.

Zachary Frangella, Joel A. Tropp, and Madeleine Udell. Randomized Nyström preconditioning. *SIAM Journal on Matrix Analysis and Applications*, 44(2):718–752, 2023.

Zachary Frangella, Pratik Rathore, Shipu Zhao, and Madeleine Udell. Promise: Preconditioned stochastic optimization methods by incorporating scalable curvature estimates. *Journal of Machine Learning Research*, 25(346):1–57, 2024a.

Zachary Frangella, Pratik Rathore, Shipu Zhao, and Madeleine Udell. Sketchysgd: Reliable stochastic optimization via randomized curvature estimates. *SIAM Journal on Mathematics of Data Science*, 6(4):1173–1204, 2024b.

Daniel Gabay and Bertrand Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & mathematics with applications*, 2(1): 17–40, 1976.

Jacob Gardner, Geoff Pleiss, Kilian Q Weinberger, David Bindel, and Andrew G Wilson. GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In *Advances in Neural Information Processing Systems*, 2018.

Guillaume Gautier, Guillermo Polito, Rémi Bardenet, and Michal Valko. DPPy: DPP Sampling with Python. *Journal of Machine Learning Research - Machine Learning Open Source Software (JMLR-MLOSS)*, 2019.

Nidham Gazagnadou, Mark Ibrahim, and Robert M. Gower. RidgeSketch: A fast sketching based solver for large scale ridge regression. *SIAM Journal on Matrix Analysis and Applications*, 43(3): 1440–1468, 2022.

Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An Investigation into Neural Net Optimization via Hessian Eigenvalue Density. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Behrooz Ghorbani, Song Mei, Theodor Misiakiewicz, and Andrea Montanari. When Do Neural Networks Outperform Kernel Methods? In *Advances in Neural Information Processing Systems*, 2020.

Behrooz Ghorbani, Song Mei, Theodor Misiakiewicz, and Andrea Montanari. Linearized two-layers neural networks in high dimension. *The Annals of Statistics*, 49(2):1029–1054, 2021.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010.

Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970.

Gene H Golub and Gérard Meurant. *Matrices, moments and quadrature with applications*, volume 30. Princeton University Press, 2009.

Gene H. Golub and Charles F. Van Loan. *Matrix Computations - 4th Edition*. Johns Hopkins University Press, 2013.

Paul Goulart and Yuwen Chen. Clarabel: An interior-point solver for conic programs with quadratic objectives. *arXiv preprint arXiv:2405.12762*, 2024.

Robert Gower, Filip Hanzely, Peter Richtarik, and Sebastian U Stich. Accelerated stochastic matrix inversion: General theory and speeding up BFGS rules for faster second-order optimization. In *Advances in Neural Information Processing Systems*, 2018.

Robert M Gower and Peter Richtárik. Randomized iterative methods for linear systems. *SIAM Journal on Matrix Analysis and Applications*, 36(4):1660–1690, 2015.

Michael Grant, Stephen Boyd, and Yinyu Ye. *Disciplined Convex Programming*, pages 155–210. Springer US, 2006.

Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53 (2):217–288, 2011.

Zhongkai Hao, Jiachen Yao, Chang Su, Hang Su, Ziao Wang, Fanzhi Lu, Zeyu Xia, Yichi Zhang, Songming Liu, Lu Lu, and Jun Zhu. PINNacle: A Comprehensive Benchmark of Physics-Informed Neural Networks for Solving PDEs. *arXiv preprint arXiv:2306.08827*, 2023.

Helmut Harbrecht, Michael Peters, and Reinhold Schneider. On the low-rank approximation by the pivoted Cholesky decomposition. *Applied Numerical Mathematics*, 62(4):428–440, 2012.

Nicholas J Higham. *Accuracy and stability of numerical algorithms*. SIAM, 2002.

Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 2nd edition, 2012.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural networks*, 3(5):551–560, 1990.

M.F. Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, 19(2):433–450, 1990.

Ameya D Jagtap and George Em Karniadakis. Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Communications in Computational Physics*, 28(5):2002–2041, 2020.

Ameya D. Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020a.

Ameya D. Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 2020b.

Ameya D. Jagtap, Ehsan Kharazmi, and George Em Karniadakis. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 365:113028, 2020c.

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, 2013.

Hamed Karimi, Julie Nutini, and Mark Schmidt. Linear convergence of gradient and proximal-gradient methods under the Polyak-łojasiewicz condition. In *Machine Learning and Knowledge Discovery in Databases: European Conference*, pages 795–811, 2016.

George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

Ehsan Kharazmi, Zhongqiang Zhang, and George E.M. Karniadakis. hp-VPINNs: Variational physics-informed neural networks with domain decomposition. *Computer Methods in Applied Mechanics and Engineering*, 374:113547, 2021.

Siavash Khodakarami, Vivek Oommen, Nazanin Ahmadi Daryakenari, Maxim Beekenkamp, and George Em Karniadakis. Spectral bias in physics-informed and operator learning: Analysis and mitigation guidelines. *arXiv preprint arXiv:2602.19265*, 2026.

Reza Khodayi-Mehr and Michael Zavlanos. VarNet: Variational Neural Networks for the Solution of Partial Differential Equations. In *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, pages 298–307, 2020.

George S Kimeldorf and Grace Wahba. A correspondence between Bayesian estimation on stochastic processes and smoothing by splines. *The Annals of Mathematical Statistics*, 41(2):495–502, 1970.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Dmitry Kovalev, Samuel Horváth, and Peter Richtárik. Don't jump through hoops and remove those loops: SVRG and Katyusha are better without the outer loop. In *International Conference on Algorithmic Learning Theory*, 2020.

Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. In *Advances in Neural Information Processing Systems*, 2021.

Jacek Kuczyński and Henryk Woźniakowski. Estimating the largest eigenvalue by the power and lanczos algorithms with a random start. *SIAM Journal on Matrix Analysis and Applications*, 13 (4):1094–1122, 1992.

Alex Kulesza and Ben Taskar. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 5(2–3):123–286, 2012.

Jonathan Lacotte and Mert Pilanci. Effective dimension adaptive sketching methods for faster regularized least-squares optimization. In *Advances in Neural Information Processing Systems*, 2020.

Jonathan Lacotte and Mert Pilanci. Adaptive and oblivious randomized subspace methods for high-dimensional optimization: Sharp analysis and lower bounds. *IEEE Transactions on Information Theory*, 68(5):3281–3303, 2022.

Jonathan Lacotte, Mert Pilanci, and Marco Pavone. High-dimensional optimization in adaptive random subspaces. In *Advances in Neural Information Processing Systems*, 2019.

Jason D. Lee, Ioannis Panageas, Georgios Piliouras, Max Simchowitz, Michael I. Jordan, and Benjamin Recht. First-order methods almost always avoid strict saddle points. *Mathematical Programming*, 176(1):311–337, 2019.

Ke Li, Kejun Tang, Tianfan Wu, and Qifeng Liao. D3M: A Deep Domain Decomposition Method for Partial Differential Equations. *IEEE Access*, 8:5283–5294, 2020.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier Neural Operator for Parametric Partial Differential Equations. In *International Conference on Learning Representations*, 2021.

Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations. *ACM / IMS J. Data Sci.*, 1(3), 2024.

Jihao Andreas Lin, Shreyas Padhy, Javier Antoran, Austin Tripp, Alexander Terenin, Csaba Szepesvari, José Miguel Hernández-Lobato, and David Janz. Stochastic gradient descent for Gaussian processes done right. In *International Conference on Learning Representations*, 2024.

Lin Lin, Yousef Saad, and Chao Yang. Approximating spectral densities of large matrices. *SIAM review*, 58(1):34–65, 2016.

Chaoyue Liu, Libin Zhu, and Misha Belkin. On the linearity of large non-linear models: when and why the tangent kernel is constant. *Advances in Neural Information Processing Systems*, 2020.

Chaoyue Liu, Libin Zhu, and Mikhail Belkin. Loss landscapes and optimization in over-parameterized non-linear systems and neural networks. *Applied and Computational Harmonic Analysis*, 59:85–116, 2022.

Chaoyue Liu, Dmitriy Drusvyatskiy, Mikhail Belkin, Damek Davis, and Yi-An Ma. Aiming towards the minimizers: fast convergence of SGD for overparametrized problems. *arXiv preprint arXiv:2306.02601*, 2023.

Dong C Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.

Ji Liu and Stephen J. Wright. An accelerated randomized Kaczmarz algorithm. *Mathematics of Computation*, 85(297):153–178, 2016.

Songming Liu, Chang Su, Jiachen Yao, Zhongkai Hao, Hang Su, Youjia Wu, and Jun Zhu. Preconditioning for physics-informed neural networks. *arXiv preprint arXiv:2402.00531*, 2024.

Haihao Lu and Jinwen Yang. cuPDLP.jl: A GPU implementation of restarted primal-dual hybrid gradient for linear programming in Julia. *Operations Research*, 73(6):3440–3452, 2025.

Haihao Lu, Zedong Peng, and Jinwen Yang. MPAX: Mathematical programming in JAX. *arXiv preprint arXiv:2412.09734*, 2024.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021a.

Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. DeepXDE: A Deep Learning Library for Solving Differential Equations. *SIAM Review*, 63(1):208–228, 2021b.

Lu Lu, Raphael Pestourie, Wenjie Yao, Zhicheng Wang, Francesc Verdugo, and Steven G Johnson. Physics-informed neural networks with hard constraints for inverse design. *SIAM Journal on Scientific Computing*, 43(6):B1105–B1132, 2021c.

Lu Lu, Raphaël Pestourie, Steven G Johnson, and Giuseppe Romano. Multifidelity deep neural operators for efficient learning of partial differential equations with application to fast inverse design of nanoscale heat transport. *Physical Review Research*, 4(2):023210, 2022.

Siyuan Ma and Mikhail Belkin. Diving into the shallows: A computational perspective on large-scale shallow learning. In *Advances in Neural Information Processing Systems*, 2017.

Siyuan Ma and Mikhail Belkin. Kernel machines that adapt to GPUs for effective large batch training. In *Proceedings of Machine Learning and Systems*, 2019.

Michael W. Mahoney. Randomized algorithms for matrices and data. *Foundations and Trends in Machine Learning*, 3(2):123–224, 11 2011.

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

Giacomo Meanti, Luigi Carratino, Lorenzo Rosasco, and Alessandro Rudi. Kernel methods through the roof: Handling billions of points efficiently. In *Advances in Neural Information Processing Systems*, 2020.

Giacomo Meanti, Antoine Chatalic, Vladimir Kostic, Pietro Novelli, Massimiliano Pontil, and Lorenzo Rosasco. Estimating Koopman operators with sketching to provably learn large scale dynamical systems. In *Advances in Neural Information Processing Systems*, 2024.

Raphael A. Meyer, Cameron Musco, Christopher Musco, and David P. Woodruff. *Hutch++: Optimal Stochastic Trace Estimation*, pages 142–155. 2021.

Siddhartha Mishra and Roberto Molinaro. Estimates on the generalization error of physics-informed neural networks for approximating pdes. *IMA Journal of Numerical Analysis*, 43(1):1–43, 2023.

Ben Moseley, Andrew Markham, and Tarje Nissen-Meyer. Finite basis physics-informed neural networks (FBPINNs): a scalable domain decomposition approach for solving differential equations. *Advances in Computational Mathematics*, 49(4):62, 2023.

Johannes Müller and Marius Zeinhofer. Achieving High Accuracy with PINNs via Energy Natural Gradient Descent. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Cameron Musco and Christopher Musco. Recursive sampling for the Nyström method. In *Advances in Neural Information Processing Systems*, 2017.

Mojmir Mutny, Michał Dereziński, and Andreas Krause. Convergence analysis of block coordinate algorithms with determinantal sampling. In *International Conference on Artificial Intelligence and Statistics*, 2020.

Mohammad Amin Nabian, Rini Jasmine Gladstone, and Hadi Meidani. Efficient training of physics-informed neural networks via importance sampling. *Comput.-Aided Civ. Infrastruct. Eng.*, 36(8): 962–977, 2021.

Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization*, 19(4):1574–1609, 2009.

Yurii Nesterov. *Lectures on convex optimization*, volume 137. Springer, 2018.

Yurii Nesterov and Arkadii Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics, 1994.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.

EJ Nyström. Über die praktische auflösung von integralgleichungen mit anwendungen auf randwertaufgaben. *Acta Mathematica*, 54(1):185–204, 1930.

Brendan O'Donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169:1042–1068, 2016.

Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and trends® in Optimization*, 1(3):127–239, 2014.

Jonathan Parkinson and Wei Wang. Linear-scaling kernels for protein sequences and small molecules outperform deep learning while providing uncertainty quantitation and improved interpretability. *Journal of Chemical Information and Modeling*, 63(15):4589–4601, 2023.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

Mert Pilanci and Martin J. Wainwright. Iterative hessian sketch: Fast and accurate solution approximation for constrained least-squares. *Journal of Machine Learning Research*, 17(53):1–38, 2016.

Boris Teodorovich Polyak. Gradient methods for minimizing functionals. *Zhurnal vychislitel'noi matematiki i matematicheskoi fiziki*, 3(4):643–653, 1963.

M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Machine learning of linear differential equations using Gaussian processes. *Journal of Computational Physics*, 348:683–693, 2017.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. The MIT Press, 11 2005.

Pratik Rathore, Weimu Lei, Zachary Frangella, Lu Lu, and Madeleine Udell. Challenges in training PINNs: A loss landscape perspective. In *Forty-first International Conference on Machine Learning*, 2024.

Pratik Rathore, Zachary Frangella, Sachin Garg, Shaghayegh Fazliani, Michal Derezinski, and Madeleine Udell. Turbocharging gaussian process inference with approximate sketch-and-project. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Pratik Rathore, Zachary Frangella, Jiaming Yang, Michał Dereziński, and Madeleine Udell. Have askotch: A neat solution for large-scale kernel ridge regression. *arXiv preprint arXiv:2407.10070*, 2026.

Peter Richtárik and Martin Takáč. Stochastic reformulations of linear systems: Algorithms and convergence theory. *SIAM Journal on Matrix Analysis and Applications*, 41(2):487–524, 2020.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

Franz M. Rohrhofer, Stefan Posch, Clemens Gößnitzer, and Bernhard C Geiger. On the Role of Fixed Points of Dynamical Systems in Training Physics-Informed Neural Networks. *Transactions on Machine Learning Research*, 2023.

Alessandro Rudi, Raffaello Camoriano, and Lorenzo Rosasco. Less is more: Nyström computational regularization. In *Advances in Neural Information Processing Systems*, 2015.

Alessandro Rudi, Luigi Carratino, and Lorenzo Rosasco. Falkon: An optimal large scale kernel method. In *Advances in Neural Information Processing Systems*, 2017.

Alessandro Rudi, Daniele Calandriello, Luigi Carratino, and Lorenzo Rosasco. On fast leverage score sampling and optimal learning. In *Advances in Neural Information Processing Systems*, 2018.

Ernest K. Ryu and Wotao Yin. *Large-Scale Convex Optimization: Algorithms & Analyses via Monotone Operators*. Cambridge University Press, 2022.

Stefan A. Sauter and Christoph Schwab. *Boundary Element Methods*. Springer, 2011.

Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT Press, 2002.

D. F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656, 1970.

Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12 (4):637–672, 2020.

Annika Stuke, Milica Todorović, Matthias Rupp, Christian Kunkel, Kunal Ghosh, Lauri Himanen, and Patrick Rinke. Chemical diversity in molecular orbital energy predictions with kernel ridge regression. *The Journal of Chemical Physics*, 150(20):204121, 2019.

Jingruo Sun, Zachary Frangella, and Madeleine Udell. SAPPHIRE: Preconditioned stochastic variance reduction for faster large-scale statistical learning. *arXiv preprint arXiv:2501.15941*, 2025.

F. William Townes and Barbara E. Engelhardt. Nonnegative spatial factorization applied to spatial genomics. *Nature Methods*, 20(2):229–238, 2023.

Joel A Tropp. An introduction to matrix concentration inequalities. *Foundations and Trends® in Machine Learning*, 8(1-2):1–230, 2015.

Joel A Tropp, Alp Yurtsever, Madeleine Udell, and Volkan Cevher. Fixed-rank approximation of a positive-semidefinite matrix from streaming data. In *Advances in Neural Information Processing Systems*, 2017.

Stephen Tu, Rebecca Roelofs, Shivaram Venkataraman, and Benjamin Recht. Large scale kernel learning using block coordinate descent. *arXiv preprint arXiv:1602.05310*, 2016.

Stephen Tu, Shivaram Venkataraman, Ashia C. Wilson, Alex Gittens, Michael I. Jordan, and Benjamin Recht. Breaking locality accelerates block Gauss-Seidel. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

Shashanka Ubaru, Jie Chen, and Yousef Saad. Fast estimation of $tr(f(a))$ via stochastic lanczos quadrature. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1075–1099, 2017.

Honghui Wang, Lu Lu, Shiji Song, and Gao Huang. Learning Specialized Activation Functions for Physics-Informed Neural Networks. *Communications in Computational Physics*, 34(4):869–906, 2023.

Ke Wang, Geoff Pleiss, Jacob Gardner, Stephen Tyree, Kilian Q Weinberger, and Andrew Gordon Wilson. Exact Gaussian processes on a million data points. In *Advances in Neural Information Processing Systems*, 2019.

Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and Mitigating Gradient Flow Pathologies in Physics-Informed Neural Networks. *SIAM Journal on Scientific Computing*, 43 (5):A3055–A3081, 2021a.

Sifan Wang, Hanwen Wang, and Paris Perdikaris. On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 384:113938, 2021b.

Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Science Advances*, 7(40):eabi8605, 2021c.

Sifan Wang, Shyam Sankaran, and Paris Perdikaris. Respecting causality is all you need for training physics-informed neural networks. *arXiv preprint arXiv:2203.07404*, 2022a.

Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why PINNs fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022b.

Yongji Wang and Ching-Yao Lai. Multi-stage neural networks: Function approximator of machine precision. *Journal of Computational Physics*, 504:112865, 2024.

Yongji Wang, Mehdi Bennani, James Martens, Sébastien Racanière, Sam Blackwell, Alex Matthews, Stanislav Nikolov, Gonzalo Cao-Labora, Daniel S. Park, Martin Arjovsky, Daniel Worrall, Chongli Qin, Ferran Alet, Borislav Kozlovskii, Nenad Tomašev, Alex Davies, Pushmeet Kohli, Tristan Buckmaster, Bogdan Georgiev, Javier Gómez-Serrano, Ray Jiang, and Ching-Yao Lai. Discovery of unstable singularities. *arXiv preprint arXiv:2509.14185*, 2025.

Christopher Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, 2000.

David P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.

Anqi Wu, Samuel A. Nastase, Christopher A. Baldassano, Nicholas B. Turk-Browne, Kenneth A. Norman, Barbara E. Engelhardt, and Jonathan W. Pillow. Brain kernel: A new spatial covariance function for fMRI data. *NeuroImage*, 245:118580, 2021.

Chenxi Wu, Min Zhu, Qinyang Tan, Yadhu Kartha, and Lu Lu. A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 403:115671, 2023a.

Wensi Wu, Mitchell Daneker, Matthew A Jolley, Kevin T Turner, and Lu Lu. Effective data sampling strategies and boundary condition constraints of physics-informed neural networks for identifying material properties in solid mechanics. *Applied mathematics and mechanics*, 44(7):1039–1068, 2023b.

Yun Yang, Mert Pilanci, and Martin J. Wainwright. Randomized sketches for kernels: Fast and optimal nonparametric regression. *The Annals of Statistics*, 45(3):991 – 1023, 2017.

Jiachen Yao, Chang Su, Zhongkai Hao, Songming Liu, Hang Su, and Jun Zhu. MultiAdam: Parameter-wise Scale-invariant Optimizer for Multiscale Training of Physics-informed Neural Networks. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W. Mahoney. PyHessian: Neural Networks Through the Lens of the Hessian. In *2020 IEEE International Conference on Big Data (Big Data)*, 2020.

Haishan Ye, Luo Luo, and Zhihua Zhang. Nesterov's acceleration for approximate Newton. *Journal of Machine Learning Research*, 21(142):1–37, 2020.

Jeremy Yu, Lu Lu, Xuhui Meng, and George Em Karniadakis. Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems. *Computer Methods in Applied Mechanics and Engineering*, 393:114823, 2022.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

Yushun Zhang, Congliang Chen, Naichen Shi, Ruoyu Sun, and Zhi-Quan Luo. Adam Can Converge Without Any Modification On Update Rules. In *Advances in Neural Information Processing Systems*, 2022.

Shipu Zhao, Zachary Frangella, and Madeleine Udell. NysADMM: Faster composite convex optimization via low-rank approximation. In *Proceedings of the 39th International Conference on Machine Learning*, pages 26824–26840, 2022.