18.338 Project: Randomization to Speed up Convex Optimization

Theo Diamandis   Email: *tdiamand@mit.edu*

December 9, 2021

1 Introduction

Convex optimization techniques have transformed a huge number of fields [BBV04]; these techniques have enabled everything from SpaceX's vertical (usually non-explosive) rocket landings to modern radiation treatment planning. In fact, convex optimization continues to diffuse through field after field, accelerated by new modeling tools like `CVXPY` [DB16; Agr+18] and `Convex.jl` [Ude+14], and by open source solvers including `SCS` [ODo+16], `Hypatia.jl` [CKV20], and `COSMO.jl` [GCG19].

Although current convex optimization techniques can effectively solve moderately-sized problems with many thousands of variables and constraints, some modern applications require solving much larger problems. An increase in problem size has become especially pronounced in fields like finance, supply chain management, and machine learning, where the amount of data and the complexity of systems have both increased dramatically. The iteration time of almost all optimization methods scales superlinearly in the number of variables. As a result, developing new methods with better algorithmic complexity or parallelization capabilities is an active area of research.

For many convex optimization algorithms, the main[1] costly primitive is a linear system solve at each iteration [BBV04; ODo+16]. Thus, if we speed up the linear system solve, we speed up the optimization procedure. Techniques from randomized linear algebra show a lot of promise in providing this speedup, and they have begun to work their way into the optimization literature (*e.g.,* [PW16; FTU21; Yur+21; Der+21]). In this project, we make another step towards scaling optimization via randomization by developing a fast solver for ℓ_1-regularized regression problems, which appear frequently in machine learning.

We focus on a specific problem class in this project in order to limit engineering complexity and facilitate easy comparisons. However, the developed methods extend to many other convex optimization problems. We leave this extension to future work. Here, we hope to provide sufficient evidence to convince optimization solver developers to explore using randomized linear algebra primitives in their software. All code for this report is implemented in the Julia Language [Bez+17] and can be found at

`https://github.com/tjdiamandis/RandomizedLasso.jl`

2 Problem

Like many good problems, the one I'm working on here was proposed by my friend Andrew over lunch. Specifically, Andrew wants to solve ℓ_1-regularized regression problems (sometimes called Lasso regression) of the form

$$\text{minimize} \quad \tfrac{1}{2}\|Ax-b\|_2^2 + \lambda\|x\|_1, \tag{1}$$

where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $x \in \mathbb{R}^n$, and $m \geq n$. We assume that data fits in memory (on a big cluster machine). In the case of Andrew's problem, m is on the order of millions and n is at least

[1]This is true for the problems I'm looking at in this project but not necessarily true in general.

ten thousand. He also has to solve this problem for several thousand right hand sides b. He tried four solvers on this problem. Three of them failed. The fourth is painfully slow. Fortunately, ideas from randomized linear algebra [MT, §17] can help speed up linear system solves.

Baseline: Sketching the Data Matrix

Perhaps the most straightforward approach is to *sketch and solve.* After choosing an appropriate sketching matrix $\Omega \in \mathbb{R}^{r\times m}$ with $r \ll m$ (see [Tro+17] for a survey), we solve the problem

$$\text{minimize} \quad \tfrac{1}{2}\|\Omega Ax - \Omega b\|_2^2 + \lambda\|x\|_1. \tag{2}$$

The size reduction as a result of sketching, allows us to solve (2) with a standard, off-the-shelf solver. Due to its simplicity, the sketch and solve method is very popular in the literature. However, this method often has poor computational performance in practice, requiring either a large number of solver iterations [FTU21] or a large sketch size r [LP20] for even modest accuracy.

A More Refined Approach

Instead of sketching the entire problem, we can apply random matrix techniques to speed up the slow part of the optimization procedure: the linear system solve. Several methods exist for large-scale convex optimization. In this work, we concentrate on the Alternating Direction Method of Multipliers (ADMM) (see [BPC11] for a survey). Note that a similar idea can be applied to an interior point method (see Appendix E for details). Before discussing the use of randomized linear algebra, we outline the ADMM approach.

3 Alternating Direction Method of Multipliers

The Alternating Direction Method of Multipliers (ADMM) is a highly parallelizable method to solve convex optimization problems. It can scale to efficiently solve very large problems. Following [BPC11], we rewrite (1) as

$$\begin{array}{ll} \text{minimize} & \tfrac{1}{2}\|Ax - b\|_2^2 + \lambda\|z\|_1 \\ \text{subject to} & x - z = 0. \end{array} \tag{3}$$

Note that (3) is equivalent to (1). The iteration updates become (see Appendix C for a derivation)

$$x^{k+1} = (A^TA + \rho I)^{-1}(A^Tb + \rho(z^k - u^k)) \tag{4}$$

$$z^{k+1} = S_{\lambda/\rho}(x^{k+1} + u^k) \tag{5}$$

$$u^{k+1} = u^k + x^{k+1} - z^{k+1}, \tag{6}$$

where u is the dual variable associated with the equality constraint and S is the soft thresholding operator, defined as

$$S_\kappa(a) = \begin{cases} a - \kappa & a > \kappa \\ 0 & |a| \le \kappa \\ a + \kappa & a < -\kappa, \end{cases}$$

and implemented elementwise. We can interpret each ADMM update (4)-(6) as performing a ridge regression in (4), zeroing small weights in (5), and then enforcing consensus in (6). Most of the computational work is in solving for the x updates in (4).

Stopping Criterion

For the Lasso problem (1), we take advantage of the fact that we can construct a dual feasible point explicitly (see Appendix B for a derivation of the dual problem):

$$\nu = \frac{\lambda}{\|A^T(Ax-b)\|_\infty}(Ax-b).$$

Using the dual feasible point, we evaluate the duality gap

$$\eta = \|Ax-b\|_2^2 + \lambda\|x\|_1 + \frac{1}{2}\nu^T\nu + \nu^T b \geq f(x) - f(x^\star).$$

Note that a clever implementation only requires $O(n^2)$ operations to recompute η at each iteration. If this gap is less than some small number ϵ (we expect it to be zero at optimality), we stop. It is common to choose these tolerances as a mix of an absolute and relative criterion, for example,

$$\epsilon = \epsilon^{\text{abs}} + \epsilon^{\text{rel}} \max\{\|x^k\|, \|z^k\|\}.$$

Since the variable z is associated with the ℓ_1 penalty in (3), z^k will be sparse while x^k will only be close to sparse (x^k and z^k converge to each other in the limit). Thus, we return z^k when the algorithm terminates and use z^k to evaluate the stopping criterion above.

4 Nyström Preconditioned Conjugate Gradient Method

A direct method to solve (4) requires $O(n^3)$ flops (assuming A^TA has already been computed), which is prohibitively expensive for large problems. Thus, we use the preconditioned conjugate gradient (PCG) method to solve these liner systems since the matrices are positive definite. CG converges quickly when the linear system to be solved has clustered eigenvalues or a low condition number [TB97, Ch. 38]. Unfortunately, this system is often ill-conditioned, resulting in slow convergence. Thus, the choice of a preconditioner plays an essential role in the performance of the CG algorithm.

A standard (but often very bad) approach is to use the diagonal elements of the matrix as the preconditioner. However Frangella et al. [FTU21] propose a random *Nyström preconditioner*, which can provide benefits over the diagonal approach. In both cases we can leverage the fact that we only have to build this preconditioner once, and we can warmstart CG for x^{k+1} from the previous iteration x^k. Thus, the preconditioner's cost can be amortized over all the optimization algorithm's iterations.

We will focus on solving linear systems of the form

$$M_\mu x = (M + \mu I)x = v \qquad \text{where } M \in \mathbb{S}_+^n \text{ and } \mu \geq 0 \tag{7}$$

with PCG. We outline the results from Frangella et al. [FTU21], which provide a recipe for efficiently constructing a good preconditioner using random matrix theory.

Motivation

We know that CG converges in quickly when the condition number of a matrix is small. Specifically, for the system $Mx = b$, where $M \in \mathbb{S}_+^n$. The error (measured in M-norm) after t iterations will be reduced by a factor of $2\left(\sqrt{\kappa(M)}-1\right)/\left(\sqrt{\kappa(M)}+1\right)$ [TB97, Ch. 38]. Convergence is even faster for a clustered spectrum. These convergence results motivate us to find a preconditioner P such that $P^{-1}v$ is easily evaluated and $P^{-1/2}MP^{-1/2}$ has a tightly clustered spectrum.

We will assume that M has a rapidly decaying spectrum, with r "big" eigenvalues and the rest "small" (as measured by relative difference from λ_n). We would like to find a preconditioner that replaces the big eigenvalues $\lambda_1, \dots, \lambda_r$ with a small value in the spectrum of $P^{-1/2}MP^{-1/2}$.

Suppose we have the eigendecomposition for the best rank-r approximation to M: $M_r = V_r \Lambda_r V_r^T$ where $V_r \in \mathbb{R}^{n\times r}$ and $\Lambda = \mathbf{diag}\,(\lambda_1, \dots, \lambda_r)$. Consider the preconditioner

$$P_\star = \frac{1}{\alpha} V_r \Lambda_r V_r^T + (I - V_r V_r^T). \tag{8}$$

First, note that it is easy to invert $P_\star$:

$$P_\star^{-1} = \alpha V_r \Lambda_r^{-1} V_r^T + (I - V_r V_r^T),$$

and the associated matrix-vector product can be computed with $O(nr)$ flops. Second, the preconditioned matrix $P_\star^{-1/2} M P_\star^{-1/2}$ has the same eigenvalues as M but with the top r eigenvalues replaced by α. Choosing $\alpha \in [\lambda_n, \lambda_{r+1}]$ gives a tightly clustered spectrum. The new condition number is λ_{r+1}/λ_n and the range of the eigenvalues is now $\lambda_{r+1} - \lambda_n$. Thus, we expect the preconditioner to provide a significant reduction in CG runtime when $\lambda_{r+1} \ll \lambda_1$. Unfortunately, it is expensive to compute the best rank-r approximation accurately. For this reason, we turn to randomized linear algebra and compute a rank-r randomized Nyström approximation.

Nyström Sketch

The Nyström sketch of a matrix $M \in \mathbb{S}_+^n$ with rank parameter r is

$$\hat{M}_{\text{nys}} = (M\Omega)(\Omega^T M \Omega)^\dagger (M\Omega)^T, \qquad \text{where } \Omega \in \mathbb{R}^{n\times r} \text{ is standard normal.} \tag{9}$$

This matrix is the best positive semidefinite approximation of M whose range coincides with the range of $M\Omega$. We couldn't find this explicitly outlined anywhere, but the Nyström sketch could be inspired by solving the least squares problem

$$\hat{M}_{\text{nys}} = \underset{X \in \mathbf{range}(M\Omega)}{\operatorname{argmin}} \|M - X\|_F^2.$$

This problem is equivalent to the unconstrained problem

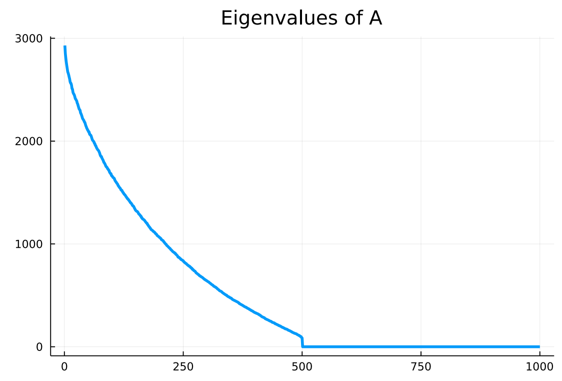
$$\hat{M}_{\text{nys}} = \underset{X}{\operatorname{argmin}} \|M - \Omega X\|_F^2. \tag{10}$$

This problem is convex, and the formula for the Nyström sketch (9) follows directly from the optimally conditions of (10).

Since (9) is not numerically sound, we use Algorithm 1 below to produce a sketch $\hat{M}_{\text{nys}} \approx M$. The randomness in the sketch ensures that the approximation is good with high probability [MT, §14]. Specifically, we have that

$$\mathbb{E}\left[\|M - \hat{M}_{\text{nys}}\|\right] \le \lambda_{k+1} + \frac{k}{r-k-1}\left(\sum_{j>k} \lambda_j\right),$$

where r and k are defined in Algorithm 1 and $\|\cdot\|$ is the spectral norm. This result essentially tells us that if the spectrum decays relatively quickly, then the approximation will be good in expectation. We verify this behavior numerically in Figure 1 for a matrix $M_\mu = BB^T + \mu I$ where $B =$ `randn(1000, 500)` and $\mu = 10^{-3}$. In fact, we see that the bound may be somewhat pessimistic.



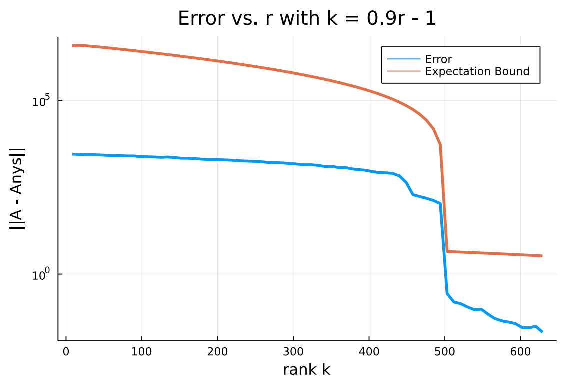


Figure 1: Eigenvalues of a random matrix (left) and the Nystrom sketch error (right)

Algorithm 1: Nystrom Sketch

```
Input: Matrix $M \in \mathbb{S}_+^n$, number of samples $r$, final rank $k$
Result: Factors of the eigenvalue decomposition of the Nyström sketch: $(U_{:,1:k}, \hat{\Lambda}_{1:k,1:k})$.
$\Omega =$ randn(n, r) ;                                  // Gaussian test matrix
$\Omega =$ qr($\Omega$).Q;
$Y = M\Omega$ ;                                           // $r$ matvecs
$\delta =$ eps($\|Y\|_F$) ;
$Y_\delta = Y + \delta\Omega$ ;                           // shift for stability
$C =$ chol($\Omega^T Y_\delta$) ;
$B = Y_\delta / C$ ;
$U, \Sigma, V =$ svd($B$);
$\Lambda = \left(\Sigma^2 - \delta I\right)_+$ ;          // remove shift, compute eigvals
```

Randomized Nyström Preconditioner

The rank-r randomized Nyström sketch of M_μ, $\hat{M}_{\text{nys}} = U\hat{\Lambda}U^T$, can be computed efficiently, and it approximates M_μ nearly as well as the optimal rank-r approximation [FTU21, Cor. 2.3]. The randomized Nyström preconditioner for M_μ and its inverse take the form

$$\begin{aligned} P &= \frac{1}{\hat{\lambda}_r + \mu} U(\hat{\Lambda} + \mu I)U^T + (I - UU^T), \\ P^{-1} &= (\hat{\lambda}_r + \mu)U(\hat{\Lambda} + \mu I)^{-1}U^T + (I - UU^T). \end{aligned} \tag{11}$$

This preconditioner reduces the condition number nearly as much as $P_\star$ (*cf.* (8)) [FTU21, Prop. 5.3]. Furthermore, if r is chosen to be sufficiently large, then PCG procudes an iterate with relative error less than ϵ after $\lceil 3.8 \log(2/\epsilon) \rceil$ iterations. The full Nyström PCG algorithm is outlined in Algorithm 2. In Figure 2, we see that the Nyström preconditioner (11) has performance that approaches the true preconditioner (8) to solve $Mx = v$ as the rank of the sketch is increased. We generated $M = BB^T$ where $B =$ `randn(1000, 500)`. Then, $v = Ax$ where $x =$ `randn(1000)`. In Figure 3, we examine PCG convergence for the ridge regression problem with $\mu = 10^{-3}$ on the Guillermo dataset [Van+13], which has $m = 20,000$ samples and $n = 4,296$ features. Convergence improves as we increase r (and $k = 0.9r$).

A key quantity in determining the quality of the preconditioner is (based on the analysis in

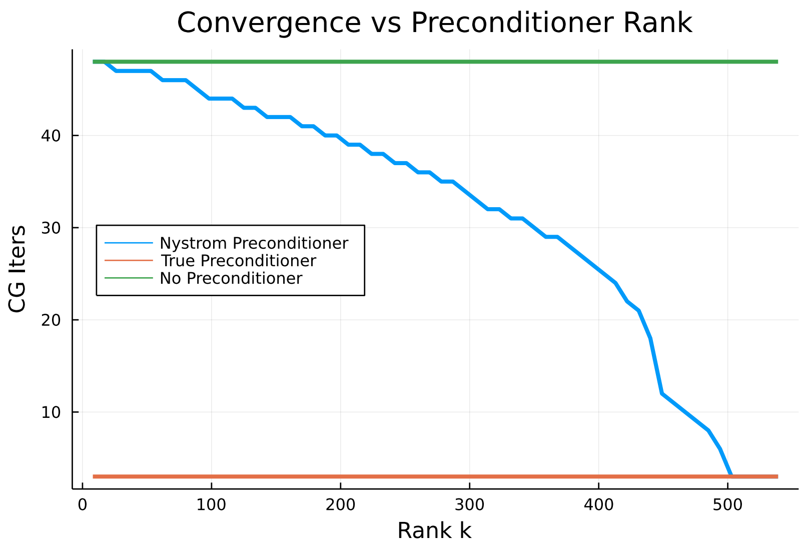


Figure 2: Convergence of PCG on a random example with different preconditioners.

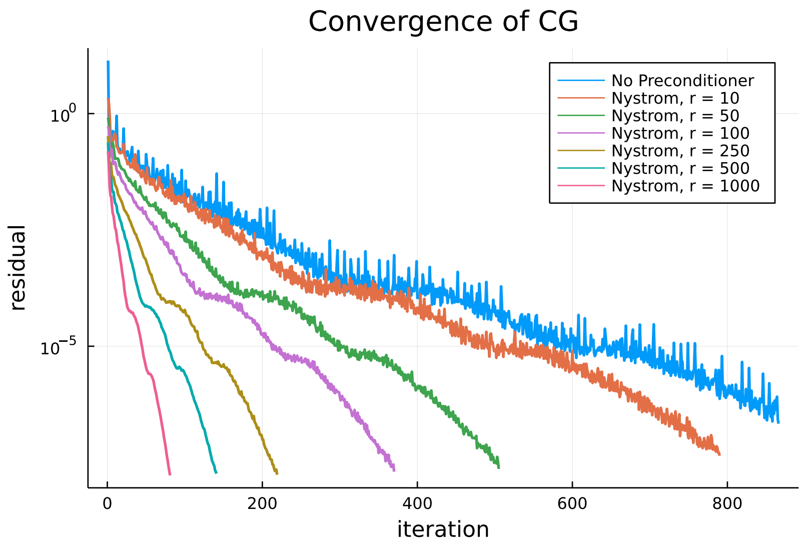


Figure 3: Convergence of PCG for ridge regression on the Guillermo dataset.

[FTU21]) the effective dimension of the regularized matrix $M + \mu I$:

$$d_{\text{eff}}(\mu) = \mathbf{Tr}(M(M+\mu I)^{-1}) = \sum_{i=1}^{n} \frac{\lambda_i(M)}{\lambda_i(M)+\mu}.$$

In particular, if we choose $r = 2\lceil 1.5 d_{\text{eff}}(\mu)\rceil + 1$, then the condition number of the preconditioned system satisfies [FTU21, Theorm 5.1]

$$\mathbb{E}\left[\kappa(P^{-1/2}A_\mu P^{-1/2})\right] < 28.$$

If r is chosen in this way, then the overall runtime of PCG is [FTU21]

$$O(d_{\text{eff}}(\mu)^2 n + n^2(d_{\text{eff}}(\mu) + \log(1/\epsilon)))$$

operations. The true d_{eff} for the dataset in Figure 3 is approximately 1900, so we'd expect convergence to continue to improve with $r > 1000$. Unfortunately, this theoretical strategy to choose r may be intractable in practice (it would require computing the eigenvalues, which destroys the computational savings we're after). We detail a more practical strategy to choose r in Appendix D.

Algorithm 2: Nyström PCG

```
Input: Matrix $A \in \mathbb{S}^n_+$, vector $b \in \mathbb{R}^n$, initial guess $x_0$, regularization parameter $\mu$, rank $r$,
       solution tolerance $\epsilon$
Result: Approximate solution $\hat{x}$ to (7)
$U, \hat{\Lambda} =$ NystromApprox($A$, $r$) ;                       // using Algorithm 1
$r_0 = b - (A + \mu I)x_0$;
$z_0 = P^{-1}r_0$ ;                                                 // using (11)
$p_0 = z_0$;
while $\|r\|_2 > \epsilon$ do
    $v = (A + \mu I)p_0$ ;
    $\alpha = (r_0^T z_0)/(p_0^T v)$ ;                              // compute step size
    $x = x_0 + \alpha p_0$ ;
    $r = r_0 - \alpha v$ ;
    $z = P^{-1}r$ ;                                                 // compute search direction
    $\beta = (r^T z)/(r_0^T z_0)$ ;
    $x_0 \leftarrow x,\ r_0 \leftarrow r,\ p_o \leftarrow z + \beta p_0,\ z_0 \leftarrow z$
end
```

5 A Real Example

Finally, we are ready to solve the ℓ_1 regression problem (1) using the randomized Nyström preconditioner for the PCG iterations in ADMM's x update (4). We solve the ℓ_1 regression problem for the Guillermo dataset [Van+13], which has $m = 20,000$ samples and $n = 4,296$ features. We use regularization parameter $\lambda = 0.01\lambda_{\max}$, where $\lambda_{\max} = \|A^T b\|_\infty$ is the value above which $x = 0$ is the optimal solution. We stop ADMM when the duality gap decreases below 10^{-5}. All results are computed on a Macbook Pro with a 2.3GHz 8-Core Intel i9 processor and 32GB of RAM. We use OpenBLAS [XQC21] with 8 threads for the numerical linear algebra. No other part of the algorithm is multithreaded.

Numerical Results

We run the ADMM algorithm both with and without Nyström preconditioning. The algorithm takes 131 iterations to converge to a solution with a duality gap under 10^{-5}. The final solution's root mean square error is 3.39×10^{-3}. Note that our implementation uses both over-relaxation [BPC11, §3.4.3] and an adaptive penalty parameter ρ [BPC11, §3.4.1]. Table 1 contains a runtime breakdown for both cases examined. We see that the preconditioner, which has a modest upfront setup cost, leads to a 5x improvement in solve time. Furthermore, these problems are often solved for many values of λ (called the "regularization path"). The preconditioner does not depend on λ, so it only needs to be computed once. Figure 4 compares the convergence of ADMM with and without Nyström preconditioning. It's clear that Nyström preconditioning significantly speeds up ADMM for ℓ_1-regularized regression problems solved with ADMM. These results suggest that using the Nyström preconditioner in ADMM is very much worth the additional overhead for modestly sized problems.

	No Preconditioning	Nyström Preconditioning
Setup time (total)	0.023s	2.066s
Preconditioning time	n/a	2.048s
Solve time	72.406s	14.877s
Total time	72.429s	16.944s

Table 1: ADMM runtime breakdown

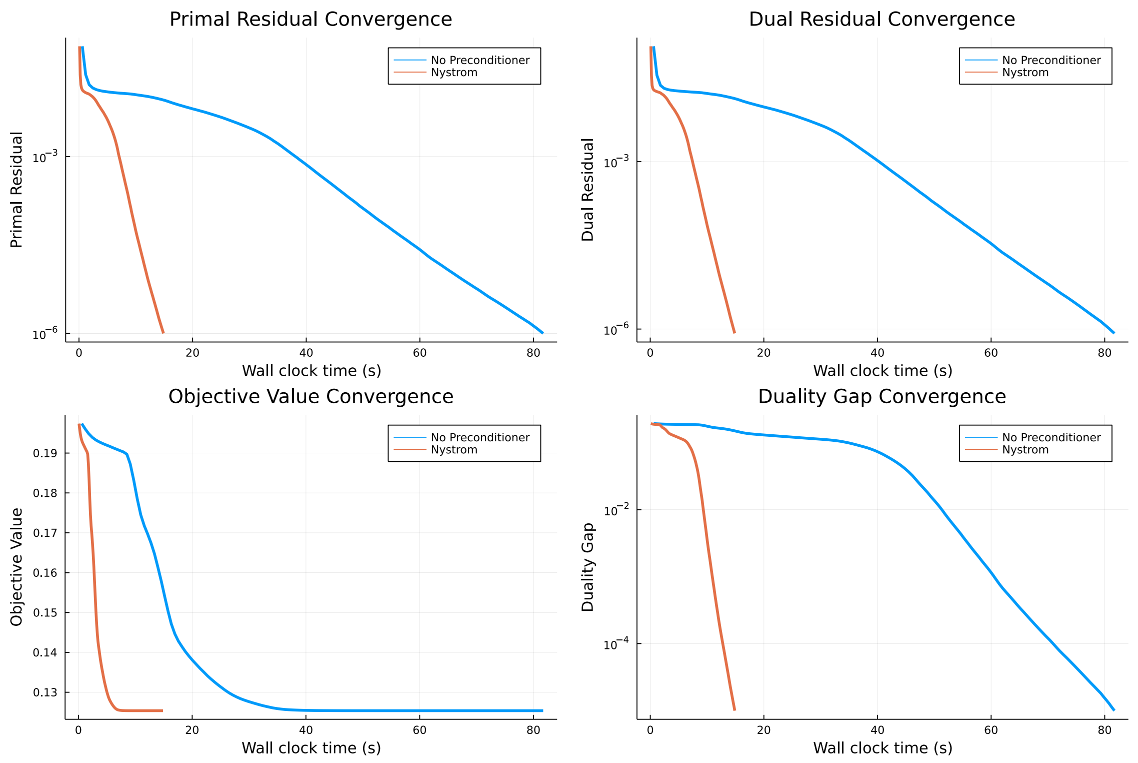


Figure 4: Convergence of ADMM with and without preconditioning on the Guillermo dataset.

6 Conclusion

In this report, we show evidence that randomized numerical linear algebra techniques can significantly speed up optimization algorithms. These results lead to several interesting questions:

1. Which other primitives (*e.g.,* linear system solves) can be sped up using randomized linear algebra techniques? Could we speed up iterative solvers for non-PSD systems, perhaps by using the randomized SVD [MT, Alg. 8] instead of the Nyström sketch?

2. Where in optimization can we get away with randomized approximation? For example, could we build semidefinite programming algorithms around randomized projections to the PSD cone ([RGN21] would suggest that the answer is "yes")?

3. What are the theoretical error guarantees when randomization is used in optimization?

In addition, these results point to several useful lines of future work on the implementation side including

1. Standard libraries for numerical linear algebra primitives that can be easily used in existing optimization algorithms without much code refactoring (a place where Julia's multiple dispatch could shine).
2. High quality optimization libraries that are built on algorithms which directly use randomized linear algebra (*e.g.,* the `SketchyCGAL` semidefinite programming algorithm in [Yur+21][2]).

Finally the package `RandomizedLasso.jl`, which was created for this project, could use some performance improvements before it's truly ready for others to use in their research. These improvements will be completed within the next few weeks so that my friend who inspired this project can use this software in his work (creating another Python to Julia convert in the process).

[2]I implemented this over the summer but need to connect it to JuMP [DHL17]. Code is at `https://github.com/tjdiamandis/SketchyCGAL.jl`

A Alternative Approaches

Newton Update Sketches We can view (4) as the unconstrained convex optimization problem

$$\begin{aligned} x^{k+1} &= \text{argmin } x^T \left(\frac{1}{2}A^TA + \rho I\right) x + \left(A^Tb + \rho(z^k - u^k)\right)^T x \\ &= \text{argmin } \frac{1}{2}\|Ax - b\|^2 + \rho\|x - (z^k - u^k)\|^2 \\ &= \text{argmin } f(x) + g(x). \end{aligned} \tag{12}$$

Both functions are smooth, convex, and twice differentiable. We can apply the `Newton-LESS` method developed by Dereziński et al. [Der+21] to iteratively solve (12):

$$x^{k+1,t+1} = x^{k+1,t} - \mu(A^TS_t^TS_tA + \rho I)^{-1}\left((A^TA + 2\rho I)x + \left(A^Tb + \rho(z^k - u^k)\right)\right), \tag{13}$$

which can be recognized as a sketched Netwon step. There are several choices for the skecthing matrix S_t, including Gaussian [PW16] and sparsity-preserving skecthes [Der+21]. The classic quasi-Netwon method LBFGS [LN89] would provide a good baseline to compare against.

B Dual Problem

In this section, we derive the Lagrange dual of the ℓ_1-regularized regression problem (1) following [Kim+07]. First, introduce a new variable $y \in \mathbb{R}^n$ to write the equivalent problem

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}y^Ty + \lambda\|x\|_1 \\ \text{subject to} & y = Ax - b. \end{array} \tag{14}$$

The Lagrangian is then

$$L(x, y, \nu) = \frac{1}{2}y^Ty + \lambda\|x\|_1 + \nu^T(Ax - b - y).$$

We obtain the dual function by minimizing over x and y for a fixed ν.

$$G(\nu) = \inf_{x,y} L(x, y, \nu) = \begin{cases} -\frac{1}{2}\nu^T\nu - \nu^Tb, & \|(A^T\nu)\|_\infty \le \lambda, \\ -\infty, & \text{otherwise.} \end{cases}$$

Thus, the dual problem is

$$\begin{array}{ll} \text{maximize} & G(\nu) \\ \text{subject to} & \|(A^T\nu)\|_\infty \le \lambda, \end{array} \tag{15}$$

which is also convex. Furthermore, any dual feasible point ν provides a lower bound on the optimal value of (1), and under mild conditions, the optimal values of (15) and (1) are the same [BBV04, Ch. 5].

Optimality Conditions The optimality conditions for (1) provide a means to relate the primal and dual variables directly. Since this problem is unconstrained, the optimality conditions are simply that 0 is contained in the subdifferential, *i.e.,*

$$0 \in A^T(Ax - b) + \lambda\partial\|x\|_1,$$

where

$$(\partial\|x\|_1)_i = \begin{cases} 1 & x > 0 \\ -1 & x < 0 \\ [-1,1] & x = 0 \end{cases}$$

and notation is abused in a sensible way. Rearranging a bit, we have the optimality conditions

$$(A^T(Ax-b))_i \in \begin{cases} \{+\lambda\} & x_i < 0 \\ \{-\lambda\} & x_i > 0 \\ [-\lambda,\lambda] & x_i = 0. \end{cases} \tag{16}$$

Dual Feasible Points From the definition of our Langrangian, we have that $y^\star = \nu^\star = Ax^\star - b$. Given an arbitrary x, we can easily construct dual feasible points for (15) by taking into account the constraint and the optimality conditions (16):

$$\nu = \frac{\lambda}{\|A^T(Ax-b)\|_\infty}(Ax-b). \tag{17}$$

Assuming not all $x_i = 0$, then this dual variable is optimal when constructed with $x = x^\star$ by (16). When $x \neq x^\star$, we can see that (17) satisfies

$$\|A^T\nu\|_\infty = \left\|A^T\left(\frac{\lambda}{\|A^T(Ax-b)\|_\infty}(Ax-b)\right)\right\|_\infty = \frac{\lambda}{\|A^T(Ax-b)\|_\infty}\|A^T(Ax-b)\|_\infty \leq \lambda.$$

Therefore, ν is dual feasible and $G(\nu)$ provides a lower bound on (1).

C ADMM Method

The ADMM provides a powerful means to decompose and parallelize convex optimization problems. It is related to a number of other methods, including Douglas-Rachford splitting, proximal methods, and Dykstra's alternating projections. In general, the algorithm solves problems of the form

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & Ax + Bz = c. \end{array}$$

Define the augmented Lagrangian as

$$L_\rho(x,z,u) = f(x) + g(z) + y^T(Ax+By-c) + (\rho/2)\|Ax+By-c\|_2^2,$$

where y is the dual variable. ADMM consists of the iterations

$$\begin{aligned} x^{k+1} &= \operatorname*{argmin}_x L_\rho(x,z^k,y^k) \\ z^{k+1} &= \operatorname*{argmin}_z L_\rho(x^{k+1},z,y^k) \\ y^{k+1} &= y^k + \rho(Ax^{k+1} + Bz^{k+1} - c). \end{aligned}$$

ADMM is similar to the method of multipliers, where we minimize jointly over x and z. The "alternating direction" refers to the sequential x then z minimization in ADMM. Alternatively, ADMM can be seen as the method of multipliers with a single Gauss-Sidel pass over x and z instead of joint minimization [BPC11].

Sometimes, it is more convenient to scale the dual variable in ADMM to get the iterates

$$\begin{aligned}
x^{k+1} &= \underset{x}{\operatorname{argmin}} \left(f(x) + (\rho/2)\|Ax + Bz6K - c + u^k\|_2^2 \right) \\
z^{k+1} &= \underset{z}{\operatorname{argmin}} \left(g(z) + (\rho/2)\|Ax^{k+1} + Bz - c + u^k\|_2^2 \right) \\
u^{k+1} &= u^k + Ax^{k+1} + Bz^{k+1} - c.
\end{aligned} \tag{18}$$

In this work, we use the scaled version of ADMM (18) to derive (4)-(5).

D Sketch Rank Selection

One crucial part of the algorithm outlined in this work is the selection of the parameters r and k for the Nyström sketch. We take $k = 0.9r$ and use the procedure from [FTU21] which can be summarized as follows:

1. Create a sketch $A_{\text{nys}} = \texttt{nystrom\_sketch}(A, k, r)$ using parameter r (Algorithm 1).
2. Estimate the error $E = \|A_{\text{nys}} - A\|_F^2$ using the randomized power method (Algorithm 3).
3. If $E < \epsilon$, return A_{nys}. Otherwise, set $r \leftarrow 2r$ and repeat.

We apply this procedure to generate A_{nys} before forming the randomized preconditioner.

Algorithm 3: Randomized Power method for estimating $\|A_{\text{nys}} - A\|$

```
Input: $A \in \mathbb{S}_+^n$ and its Nyström sketch $A_{\text{nys}}$
Result: Estimate $E$ of $\|A_{\text{nys}} - A\|$
$g$ = randn(n)
$v$ = normalize($g$)
for $i = 1, 2, \ldots, q$ do
    $v = Av_0 - A_{\text{nys}}v_0$
    $E = v_0^T v$
    $v$ = normalize($v$)
    $v_0 = v$
end
```

E Interior Point Method

As an alternative to ADMM, we could use the same preconditioner technique with an interior point method. Interior point methods formed the basis of the first widely available, high-quality convex optimization solvers in the late 90's and early 00's, built on theoretical work the decade prior. The popularity of these methods stems from their low theoretical and empirical iteration complexity (*i.e.,* the number of iterations required to reach error less than ϵ); however, these methods do not parallelize as nicely as ADMM-based approaches. For datasets that fit on a single machine, this inability to parallelize may not be a concern.

Here, we outline the truncated Newton interior-point method outlined in [Kim+07]. We start by transforming (1) into the equivalent convex problem

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|Ax - b\|_2^2 + \lambda \sum_{i=1}^d u_i \\ \text{subject to} & -u_i \le x_i \le u_i, \quad i \in [n], \end{array} \tag{19}$$

with variables $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^n$. We can then transfer this constraint to the objective function by defining a barrier function, *i.e.,* a function that is (nearly) zero when the constraint is satisfied and infinity when it is violated. For the bound constraints in (19) we use

$$\Phi(x, u) = -\sum_{i=1}^{n} \log(u_i + x_i) - \sum_{i=1}^{n} \log(u_i - x_i).$$

This allows us to turn (19) into the unconstrained optimization problem

$$\operatorname{argmin} \phi_t(x, u) = t\|Ax - b\|_2^2 + t\sum_{i=1}^{n} \lambda u_i + \Phi(x, u), \tag{20}$$

where we vary the parameter t from 0 to ∞.

Now, we essentially just apply Newton's method to the optimality conditions of (20), which consists of three steps (expanded upon in the subsequent section). First, we compute a search direction by solving the Newton system

$$\nabla^2 \phi_t(x, u) \begin{bmatrix} \Delta x \\ \Delta u \end{bmatrix} = \nabla \phi_t(x, u). \tag{21}$$

Next, we compute a step size s via a backtracking line search (see [BBV04, Ch. 9]) and update the iterate

$$(x, u) \leftarrow (x, u) + s(\Delta x, \Delta u).$$

Finally, we construct a dual feasible point (see Appendix B)

$$\nu = \frac{\lambda}{\|A^T(Ax - b)\|_\infty}(Ax - b)$$

to evaluate the duality gap

$$\eta = \|Ax - b\|_2^2 + \lambda\|x\|_1 + \frac{1}{2}\nu^T\nu + \nu^T b.$$

If this gap is small enough (we expect it to be zero at optimality), we stop. Otherwise, we increase t and repeat the procedure. The most expensive step in each iteration is the linear system solve (21), which is where we will focus our attention.

E.1 Implementation Details

Recall that the interior point method consists of three main steps:

1. Compute a **search direction** by solving the Newtown system (21).
2. Compute a **step size** s via a backtracking line search and update the iterate.
3. Compute a dual feasible point and corresponding **duality gap**. If this is small enough, terminate. Otherwise, update the barrier parameter and repeat this procedure.

Initialization Following [Kim+07], we initialize $t = 1/\lambda$, $x = 0$, and $u = \mathbf{1}$. We must initialize (x, u) such that the barrier function $\Phi(x, u) < \infty$. This condition is then guaranteed to hold for all subsequent iterates. In addition, we choose a parameter $\epsilon_{\text{rel}} > 0$ for the stopping criterion

$$\frac{f(x) - f(x^\star)}{f(x^\star)} \leq \epsilon_{\text{rel}}.$$

Search Direction We compute a search direction by solving the Newton system

$$\nabla^2 \phi_t(x,u) \begin{bmatrix} \Delta x \\ \Delta u \end{bmatrix} = \nabla \phi_t(x,u).$$

The Hessian can be written as

$$\nabla^2 \phi_t(x,u) = \begin{bmatrix} 2tA^T A + D_1 & D_2 \\ D_2 & D_1 \end{bmatrix}$$

where

$$D_1 = \mathbf{diag}\left(\frac{2(u_1^2 + x_1^2)}{(u_1^2 - x_1^2)^2}, \dots, \frac{2(u_n^2 + x_n^2)}{(u_n^2 - x_n^2)^2} \right)$$
$$D_2 = \mathbf{diag}\left(\frac{-4u_1 x_1}{(u_1^2 - x_1^2)^2}, \dots, \frac{-4u_n x_n}{(u_n^2 - x_n^2)^2} \right).$$

The gradient can be written as

$$\nabla \phi_t(x,u) = \begin{bmatrix} 2tA^T(Ax-y) \\ t\lambda \mathbf{1} \end{bmatrix} + \begin{bmatrix} 2x_1/(u_1^2 - x_1^2) \\ \vdots \\ 2x_n^2(u_n^2 - x_n^2) \\ 2u_1/(u_1^2 - x_1^2) \\ \vdots \\ 2u_n^2(u_n^2 - x_n^2) \end{bmatrix}.$$

A potential (non-random) preconditioner for CG is

$$P = \begin{bmatrix} 2t\,\mathbf{diag}\left(A^T A\right) + D_1 & D_2 \\ D_2 & D_1 \end{bmatrix} = \begin{bmatrix} D_3 & D_2 \\ D_2 & D_1 \end{bmatrix}.$$

For PCG, we are interested in the solve operation $P^{-1}v$ for $v = (v_1, v_2) \in \mathbb{R}^{2n}$, which can be computed as (see [Kim+07])

$$P^{-1}v = \begin{bmatrix} (D_1 D_3 - D_2^2)^{-1}(D_1 v_1 - D_2 v_2) \\ (D_1 D_3 - D_2^2)^{-1}(-D_2 v_1 + D_3 v_2) \end{bmatrix}.$$

Since all matrices D_i involved are diagonal, the computational cost is $O(n)$ flops.

Step Size Next, we compute a step size s via a backtracking line search via Algorithm 4. Typical values are $\alpha = 0.01$ and $\beta = 0.5$. Note that by convention we will take $\phi_t(x + s\Delta x, u + s\Delta u) = \infty$ if the input is not in the domain. Once we compute the step size, we update the iterate

$$(x,u) \leftarrow (x,u) + s(\Delta x, \Delta u).$$

Algorithm 4: Backtracking Line Search [BBV04, Ch. 9]

```
Input: Decent direction $(\Delta x, \Delta u)$ for $\phi_t(x,u)$, $\alpha \in (0, 0.5)$, $\beta \in (0,1)$.
Result: Step size $s$.
$s = 1$ ;
while $\phi_t(x + s\Delta x, u + s\Delta u) > \phi_t(x,u) + \alpha s \nabla \phi_t(x,u)^T(\Delta x, \Delta u)$ do
    $s = \beta s$
end
```

Duality Gap Following the derivation in Appendix B, we construct a dual feasible point

$$\nu = \frac{\lambda}{\|A^T(Ax-b)\|_\infty}(Ax-b).$$

From weak duality, we know that

$$G(\nu) \leq f(x^\star) \leq f(x) \tag{22}$$

for any dual feasible ν. Thus, we can conclude that the duality gap $\eta = f(x) - G(\nu)$ satisfies

$$f(x) - f(x^\star) \leq f(x) - G(\nu) = \eta. \tag{23}$$

To construct a bound on the relative duality gap

$$\frac{f(x) - f(x^\star)}{f(x^\star)}$$

we use (22) and (23) to get

$$\frac{f(x) - f(x^\star)}{f(x^\star)} \leq \frac{\eta}{G(\nu)}.$$

Thus, we stop the optimization procedure if $\eta / G(\nu) \leq \epsilon_{\text{rel}}$. Otherwise, we update $t = \mu t$ (*e.g.*, taking $\mu \in [2, 50]$, as described in [BBV04, Ch. 11]) and repeat.

References

[Agr+18] Akshay Agrawal et al. “A rewriting system for convex optimization problems.” In: *Journal of Control and Decision* 5.1 (2018), pp. 42–60.

[BBV04] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization.* Cambridge university press, 2004.

[Bez+17] Jeff Bezanson et al. “Julia: A fresh approach to numerical computing.” In: *SIAM review* 59.1 (2017), pp. 65–98.

[BPC11] Stephen Boyd, Neal Parikh, and Eric Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers.* Now Publishers Inc, 2011.

[CKV20] Chris Coey, Lea Kapelevich, and Juan Pablo Vielma. “Towards practical generic conic optimization.” In: *arXiv preprint arXiv:2005.01136* (2020).

[DB16] Steven Diamond and Stephen Boyd. “CVXPY: A Python-embedded modeling language for convex optimization.” In: *Journal of Machine Learning Research* 17.83 (2016), pp. 1–5.

[Der+21] Michał Dereziński et al. “Newton-LESS: Sparsification without Trade-offs for the Sketched Newton Update.” In: *arXiv preprint arXiv:2107.07480* (2021).

[DHL17] Iain Dunning, Joey Huchette, and Miles Lubin. “JuMP: A modeling language for mathematical optimization.” In: *SIAM review* 59.2 (2017), pp. 295–320.

[FTU21] Zachary Frangella, Joel A Tropp, and Madeleine Udell. “Randomized Nyström Preconditioning.” In: *arXiv preprint arXiv:2110.02820* (2021).

[GCG19] Michael Garstka, Mark Cannon, and Paul Goulart. “COSMO: A conic operator splitting method for large convex problems.” In: *2019 18th European Control Conference (ECC).* IEEE. 2019, pp. 1951–1956.

[Kim+07] Seung-Jean Kim et al. “An interior-point method for large-scale *ell*_1-regularized least squares.” In: *IEEE journal of selected topics in signal processing* 1.4 (2007), pp. 606–617.

[LN89] Dong C Liu and Jorge Nocedal. “On the limited memory BFGS method for large scale optimization.” In: *Mathematical programming* 45.1 (1989), pp. 503–528.

[LP20] Jonathan Lacotte and Mert Pilanci. “Effective dimension adaptive sketching methods for faster regularized least-squares optimization.” In: *arXiv preprint arXiv:2006.05874* (2020).

[MT] PG Martinsson and JA Tropp. “Randomized numerical linear algebra: foundations & algorithms (2020).” In: *arXiv preprint arXiv:2002.01387* ().

[ODo+16] Brendan O’Donoghue et al. “Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding.” In: *Journal of Optimization Theory and Applications* 169.3 (June 2016), pp. 1042–1068. URL: `http://stanford.edu/~boyd/papers/scs.html`.

[PW16] Mert Pilanci and Martin J Wainwright. “Iterative Hessian sketch: Fast and accurate solution approximation for constrained least-squares.” In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1842–1879.

[RGN21] Nikitas Rontsis, Paul Goulart, and Yuji Nakatsukasa. “Efficient semidefinite programming with approximate admm.” In: *Journal of Optimization Theory and Applications* (2021), pp. 1–29.

[TB97] Lloyd N Trefethen and David Bau III. *Numerical linear algebra.* Vol. 50. Siam, 1997.

[Tro+17] Joel A Tropp et al. "Practical sketching algorithms for low-rank matrix approximation." In: *SIAM Journal on Matrix Analysis and Applications* 38.4 (2017), pp. 1454–1485.

[Ude+14] Madeleine Udell et al. "Convex Optimization in Julia." In: *SC14 Workshop on High Performance Technical Computing in Dynamic Languages* (2014). arXiv: `1410.4821 [math-oc]`.

[Van+13] Joaquin Vanschoren et al. "OpenML: Networked Science in Machine Learning." In: *SIGKDD Explorations* 15.2 (2013), pp. 49–60. DOI: `10.1145/2641190.2641198`. URL: `http://doi.acm.org/10.1145/2641190.2641198`.

[XQC21] Zhang Xianyi, Wang Qian, and Zaheer Chothia. "Openblas." In: *URL: http://xianyi.github.io/OpenBLA* 88 (2021).

[Yur+21] Alp Yurtsever et al. "Scalable semidefinite programming." In: *SIAM Journal on Mathematics of Data Science* 3.1 (2021), pp. 171–200.