# Tractable Evaluation of Stein's Unbiased Risk Estimate With Convex Regularizers

Parth Nobel, Emmanuel Candès, *Fellow, IEEE*, and Stephen Boyd, *Life Fellow, IEEE*

***Abstract*—Stein's unbiased risk estimate (SURE) gives an unbiased estimate of the $\ell_2$ risk of any estimator of the mean of a Gaussian random vector. We focus here on the case when the estimator minimizes a quadratic loss term plus a convex regularizer. For these estimators SURE can be evaluated analytically for a few special cases, and generically using recently developed general purpose methods for differentiating through convex optimization problems; these generic methods however do not scale to large problems. In this article we describe methods for evaluating SURE that handle a wide class of estimators, and also scale to large problem sizes.**

***Index Terms*—Stein's unbiased risk estimate, SURE, regularized least squares, hyper-parameter selection, trace estimation, Hutch++, unrolling, matrix completion, robust PCA.**

## I. Introduction and Background

### A. Stein's Unbiased Risk Estimate (SURE)

We consider $y \sim \mathcal{N}(\mu, \sigma^2 I)$ where $\mu \in \mathbf{R}^d$ and $I$ is the $d \times d$ identity matrix. We assume $\sigma$ is known and that we are estimating $\mu$. We are analyzing estimators $\hat{\mu} : \mathbf{R}^d \to \mathbf{R}^d$ which estimate $\mu$ given a single sample $y$. The $\ell_2$ risk of an estimator $\hat{\mu}$ is $R(\hat{\mu}) = \mathbf{E}\,\|\hat{\mu}(y) - \mu\|_2^2$.

In 1981, Charles Stein introduced in [1] what is now called Stein's unbiased risk estimate,

$$\mathrm{SURE}(\hat{\mu}, y) = -d\sigma^2 + \|\hat{\mu}(y) - y\|_2^2 + 2\sigma^2 \nabla \cdot \hat{\mu}(y), \qquad (1)$$

where $\nabla \cdot \hat{\mu}(y) = \sum_{i=1}^{d} \frac{\partial \hat{\mu}_i}{\partial y_i}(y)$ is the divergence of $\hat{\mu}$ at $y$. The divergence can also be expressed as $\nabla \cdot \hat{\mu}(y) = \mathbf{Tr}(D\hat{\mu}(y))$, where $D\hat{\mu}(y)$ is the $d \times d$ Jacobian or derivative, evaluated at $y$, and $\mathbf{Tr}$ denotes the trace of a matrix. Stein showed that the SURE statistic is an unbiased estimate of the risk in the sense that $\mathbf{E}\,\mathrm{SURE}(\hat{\mu}, y) = R(\hat{\mu})$. The challenge in evaluating $\mathrm{SURE}(\hat{\mu}, y)$ is evaluating the divergence $\nabla \cdot \hat{\mu}(y)$.

In (1), it is assumed that the estimator $\hat{\mu}$ is weakly differentiable and satisfies some integrability conditions. If this is not the case, SURE is not defined; we discuss this in more detail in Section I-G.

### B. Convex Regularized Regression

In this article we consider the setting where $\mu$ is a known linear function of unknown parameters $\beta \in B$, where $\beta$ can be a vector, a matrix, or tuples of vectors and matrices, and $B$ is the vector space of all such parameters, with dimension $p$. We will identify $B$ with $\mathbf{R}^p$, using some fixed ordering of the entries of the vectors and matrices that comprise $b \in B$. For $b \in B$, we define $\|b\|_2^2$ as the sum of the squares of the entries of $b$. In other words, we use $\|b\|_2^2$ to mean the square of the $\ell_2$ norm of $b$, interpreted as an element of $\mathbf{R}^p$. For example, if $b$ is a matrix, $\|b\|_2^2$ denotes its Frobenius norm, and not its induced $\ell_2$ norm/maximum singular value. When $b$ is a matrix and we wish to refer to its induced $\ell_2$ norm, we use the notation $\sigma_{\max}(b)$.

We take $\mu = \mathcal{A}\beta$, where $\mathcal{A} : B \to \mathbf{R}^d$ is linear. Using our identification of $B$ and $\mathbf{R}^p$, we can represent $\mathcal{A}$ explicitly as a $d \times p$ matrix. But for purposes of computing, it is more convenient to keep it abstract. In the sequel we will denote the adjoint of the mapping as $\mathcal{A}^*$.

We consider estimators given by convex regularized regression, *i.e.*, of the form

$$\hat{\mu}(y) = \mathcal{A} \operatorname*{argmin}_b \left( \frac{1}{2}\|\mathcal{A}b - y\|_2^2 + r(b) \right), \qquad (2)$$

where $r : B \to \mathbf{R} \cup \{\infty\}$ is a convex regularizer. The data in this problem are the linear mapping $\mathcal{A}$, the regularizer $r$, and the observed sample $y$. We will denote the argmin in (2) as $\hat{\beta}(y)$ so that $\hat{\mu}(y) = \mathcal{A}\hat{\beta}(y)$. Many common estimators have this form. For some of these, there are closed form expressions for either $\hat{\mu}(y)$ or SURE.

### C. This Article

In this article we introduce an algorithm to tractably compute SURE for convex regularized regression. Our algorithm, which we call SURE-CR, requires no direct access to the regularizer, only the ability to evaluate and differentiate its proximal operator, *i.e.*, a proximal operator oracle. SURE-CR requires no knowledge of $\mathcal{A}$ beyond the ability to evaluate $b \mapsto \mathcal{A}b$ and

Manuscript received 29 November 2022; revised 11 May 2023; accepted 28 September 2023. Date of publication 12 October 2023; date of current version 21 November 2023. The work of Parth Nobel was supported by the National Science Foundation Graduate Research Fellowship Program under Grant DGE-1656518. The work of Emmanuel Candès was supported in part by the Office of Naval Research under Grant N00014-20-1-2157, in part by the National Science Foundation under Grant DMS-2032014, in part by the Simons Foundation under Award 814641, and in part by the ARO under Grant 2003514594. The work of Stephen Boyd was supported in part by ACCESS (AI Chip Center for Emerging Smart Systems), sponsored by InnoHKfunding, Hong Kong SAR, and in part by the Office of Naval Research under Grant N00014-22-1-2121. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Yao Xie. *(Corresponding author: Parth Nobel.)*

Parth Nobel and Stephen Boyd are with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305 USA (e-mail: ptnobel@stanford.edu).

Emmanuel Candès is with the Department of Statistics, Stanford University, Stanford, CA 94305 USA.

Digital Object Identifier 10.1109/TSP.2023.3323046

1053-587X © 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Stanford University. Downloaded on September 01,2024 at 23:40:18 UTC from IEEE Xplore. Restrictions apply.

$v \mapsto \mathcal{A}^* v$, *i.e.*, a forward-adjoint oracle for $\mathcal{A}$ and $\mathcal{A}^*$. SURE-CR easily scales to problems with numbers of parameters in the millions, where forming or storing the matrix $D\hat{\mu}(y)$ would be impossible.

### D. Classical Examples of Convex Regularized Regression

*a) Ordinary least squares:* In ordinary least squares, $\mathcal{A}$ is a full-rank data matrix $X \in \mathbf{R}^{d \times p}$ and

$$\hat{\mu}(y) = X \operatorname*{argmin}_b \frac{1}{2}\|Xb - y\|_2^2 = X(X^*X)^{-1}X^*y.$$

With the orthogonal projection matrix $H$ defined as $H = X(X^*X)^{-1}X^*$, we have

$$\mathrm{SURE}(\hat{\mu}, y) = (2p - d)\sigma^2 + \|Hy - y\|_2^2.$$

*b) Ridge regression:* In ridge regression, $\mathcal{A}$ is a potentially rank-deficient data matrix, and

$$\begin{aligned}\hat{\mu}(y) &= X \operatorname*{argmin}_b \left(\frac{1}{2}\|Xb - y\|_2^2 + \lambda\|b\|_2^2\right) \\ &= X(X^*X + \lambda I)^{-1}X^*y,\end{aligned}$$

where $\lambda > 0$. With $H = X(X^*X + \lambda I)^{-1}X^*$, we have

$$\mathrm{SURE}(\hat{\mu}, y) = -d\sigma^2 + \|Hy - y\|_2^2 + 2\sigma^2 \,\mathbf{Tr}\, H.$$

*c) LASSO:* In LASSO, $\mathcal{A}$ is again a data matrix, and

$$\hat{\mu}(y) = X \operatorname*{argmin}_b \left(\frac{1}{2}\|Xb - y\|_2^2 + \lambda\|b\|_1\right),$$

where $\lambda > 0$. There is no analytical formula for $\hat{\mu}(y)$, but it is readily evaluated numerically. In the usual case where the LASSO solution is unique, SURE takes the form

$$\mathrm{SURE}(\hat{\mu}, y) = -d\sigma^2 + \|X\hat{\beta}(y) - y\|_2^2 + 2\sigma^2 \,\mathbf{card}\,\hat{\beta}(y),$$

where $\mathbf{card}(\cdot)$ is the number of nonzero entries [2].

The function $\hat{\mu}$ is non-differentiable on a set of Lebesgue measure 0. Therefore, the random data $y$ is almost surely at a differentiable point of $\hat{\mu}$. Specifically, one consequence of [3, Lemma 3] is that $\hat{\mu}$ is non-differentiable only on the set $\bigcup_{i=1}^{p}\{z : |(X^T z)_i| = \lambda\}$.

### E. Matrix Estimators

We now describe a few examples where $\mathcal{A}$ is not a data matrix, and except for the first example, there are no known expressions for SURE.

*a) Singular value thresholding:* The first example is singular value thresholding, where $y$ and $\beta$ are matrices in $B = \mathbf{R}^{m \times n}$ and

$$\hat{\mu}(y) = \operatorname*{argmin}_b \left(\frac{1}{2}\|b - y\|_F^2 + \lambda\|b\|_*\right),$$

where $\lambda > 0$ and $\|\cdot\|_*$ is the nuclear norm, *i.e.*, the dual of the spectral norm, the sum of the singular values of $b$. Here we take $\mathcal{A}$ to be the identity operator in our generic formulation. The estimator $\hat{\mu}$ can be expressed analytically as singular value thresholding, *i.e.*, $\hat{\mu}(y) = UF(\Sigma)V^*$, where $y = U\Sigma V^*$ is the singular value decomposition of $y$, and $F(\Sigma)$ is the diagonal matrix with $F(\Sigma)_{ii} = \max\{\Sigma_{ii} - \lambda, 0\}$. A closed form expression for SURE in this case is given in [4].

*b) Matrix completion:* Our second example is matrix completion, which extends singular value thresholding to the setting where only some entries of a matrix are observed. As in singular value thresholding, we have $\beta \in B = \mathbf{R}^{m \times n}$. In matrix completion, $\mathcal{A} : B \to \mathbf{R}^d$ is a selection operator, with $d$ the number of entries of $\beta$ that are being observed (hence, the observation $\mu$ is a vector containing the observed entries of the matrix). (A selection operator is one where each entry of $\mathcal{A}b$ is an entry of $b$.) The estimator is

$$\hat{\mu}(y) = \mathcal{A} \operatorname*{argmin}_b \left(\frac{1}{2}\|\mathcal{A}b - y\|_2^2 + \lambda\|b\|_*\right),$$

where $\lambda > 0$. Unlike singular value thresholding, there is no known analytical expression for $\hat{\mu}(y)$, but it is readily evaluated. Also, there is no known closed-form expression for SURE for matrix completion which can be tractably evaluated.

For future use we note that $\hat{\beta}(y) = 0$ if and only if

$$\lambda \geq \lambda_{\max} = \sigma_{\max}(\mathcal{A}^* y), \tag{3}$$

where $\mathcal{A}^* y$ is a matrix which satisfies $\mathcal{A}\mathcal{A}^* y = y$ and which has all entries not uniquely determined by that equation equal to 0.

*c) Robust PCA:* Our final example is robust PCA, where

$$b = (L, S) \in \mathbf{R}^{m \times n} \times \mathbf{R}^{m \times n}$$

and $\mathcal{A}(L, S) = L + S$. For completeness, we note that $\mathcal{A}^* V = (V, V)$ where $V$ is any matrix. The estimator is given by

$$\hat{\mu}(y) = \mathcal{A} \operatorname*{argmin}_{L,\, S} \left(\frac{1}{2}\|\mathcal{A}(L, S) - y\|_F^2 + \lambda\|L\|_* + \gamma\|S\|_1\right),$$

where $\lambda > 0$ and $\gamma > 0$. There is no known closed-form expression for $\hat{\mu}(y)$, but it is readily evaluated. There is no known closed-form expression for SURE.

Here too we can determine the values of $\lambda$ and $\gamma$ for which the optimal solution obeys $\hat{\beta}(y) = 0$. We have $\hat{\beta}(y) = 0$ if and only if

$$\lambda \geq \lambda_{\max} = \sigma_{\max}(y) \quad \text{and} \quad \gamma \geq \gamma_{\max} = \|y\|_\infty, \tag{4}$$

where $\|y\|_\infty = \max_{i,j} |y_{ij}|$. We are not aware of this result appearing in the literature, so we give a short derivation here. The necessary and sufficient optimality condition for $L$ and $S$ is

$$L + S - y + \lambda\partial\|L\|_* \ni 0, \qquad L + S - y + \gamma\partial\|S\|_1 \ni 0,$$

where $\partial$ denotes the subdifferential. Applying this to $L = S = 0$ we have that $L = S = 0$ is optimal if and only if

$$y \in \lambda\partial\|0\|_*, \qquad y \in \gamma\partial\|0\|_1.$$

Using the fact that the subdifferential of a norm at zero is the unit ball of the dual norm, we can write this as (4).

### F. Algorithms for Convex Regularized Regression

Several algorithms for evaluating the estimator (2) access the data $\mathcal{A}$ and $r$ in the following restricted way: the linear operator $\mathcal{A}$ is accessed only through its forward and adjoint oracle. This means we can evaluate $\mathcal{A}b$ for any $b \in B$, and $\mathcal{A}^* z$ for any $z \in \mathbf{R}^d$, where $\mathcal{A}^* : \mathbf{R}^d \to B$ is the adjoint of $\mathcal{A}$. This allows us to

Authorized licensed use limited to: Stanford University. Downloaded on September 01,2024 at 23:40:18 UTC from IEEE Xplore. Restrictions apply.

handle problems without forming or storing an explicit matrix representation of $\mathcal{A}$.

The regularizer is accessed only via its proximal operator $\mathbf{prox}_{\eta r} : B \to B$, given by

$$\mathbf{prox}_{\eta r}(v) = \underset{b}{\operatorname{argmin}} \left( \eta r(b) + \frac{1}{2}\|b - v\|_2^2 \right),$$

where $v, b \in B$ and $\eta$ is a positive scalar that can be interpreted (in the context of algorithms) as a step length. Thus our access to the regularizer is via its proximal operator, *i.e.*, we can evaluate $\mathbf{prox}_{\eta r}(v)$ for any $v$. The proximal operators of many common regularizers are known and readily computed [5], [6], [7], [8].

As examples, in LASSO, $r(b) = \lambda\|b\|_1$, and its proximal operator is given elementwise by

$$(\mathbf{prox}_{\eta r}(v))_i = \begin{cases} v_i - \eta\lambda & \text{if } v_i > \eta\lambda \\ -v_i + \eta\lambda & \text{if } v_i < \eta\lambda \\ 0 & \text{else} \end{cases}.$$

This function is known as soft-thresholding and we denote it $\mathcal{T}_{\eta\lambda}$.

In matrix completion, $r(b) = \lambda\|b\|_*$ and $\mathbf{prox}_{\eta r}(v)$ is given by singular value thresholding with regularization parameter $\eta\lambda$. In robust PCA, $r((L,S)) = \lambda\|L\|_* + \gamma\|S\|_1$ is separable with respect to $L$ and $S$. Therefore,

$$\begin{aligned} \mathbf{prox}_{\eta r}((L,S)) &= \underset{L',S'}{\operatorname{argmin}} \Big( \eta(\lambda\|L'\|_* + \gamma\|S'\|_1) \\ &\qquad + \frac{1}{2}\|(L',S') - (L,S)\|_2^2 \Big) \\ &= \underset{L',S'}{\operatorname{argmin}} \Big( \eta\lambda\|L'\|_* + \eta\gamma\|S'\|_1 \\ &\qquad + \frac{1}{2}\|L' - L\|_2^2 + \frac{1}{2}\|S' - S\|_2^2 \Big) \\ &= \Big( \underset{L'}{\operatorname{argmin}} \Big( \eta\lambda\|L'\|_* + \frac{1}{2}\|L' - L\|_2^2 \Big), \\ &\quad \underset{S'}{\operatorname{argmin}} \Big( \eta\gamma\|S'\|_1 + \frac{1}{2}\|S' - S\|_2^2 \Big) \Big) \\ &= \left( \mathbf{prox}_{\eta\lambda\|\cdot\|_*}(L), \mathbf{prox}_{\eta\gamma\|\cdot\|_1}(S) \right). \end{aligned}$$

These two proximal operators are exactly those in LASSO and matrix completion.

We now mention three algorithms that only require oracle access to $\mathcal{A}$, $\mathcal{A}^*$, and $\mathbf{prox}_{\eta r}(\cdot)$.

*a) ISTA:* The proximal gradient method (also known as ISTA) [5], [8], [9] consists of the iterations

$$b^{k+1} = \mathbf{prox}_{\eta r}\left(b^k - \eta\mathcal{A}^*\left(\mathcal{A}b^k - y\right)\right).$$

The algorithm itself requires only multiplication by $\mathcal{A}$ and $\mathcal{A}^*$. The step length $\eta$ must satisfy $\eta \le 2/\sigma_{\max}(\mathcal{A})$ to guarantee convergence [8, Section 4.2]; here, $\sigma_{\max}(\mathcal{A})$ is the induced $\ell_2$ norm, which can be computed by a power algorithm that only uses multiplication by $\mathcal{A}$ and $\mathcal{A}^*$. For our purposes, ISTA can be initialized with any vector which is selected independently of $y$, and we shall require that the Jacobian of $b^1$ with respect to $y$ always be the zero matrix.

*b) FISTA:* The accelerated proximal gradient method (also known as FISTA) [5], [8], [9] consists of adding a momentum term to the proximal gradient method to obtain the iterations

$$\begin{aligned} \tau^{k+1} &= \frac{1 + \sqrt{1 + 4\left(\tau^k\right)^2}}{2} \\ b^{k+1/2} &= b^k + \frac{\tau^k - 1}{\tau^{k+1}}\left(b^k - b^{k-1}\right) \\ b^{k+1} &= \mathbf{prox}_{\eta r}\left(b^{k+1/2} - \eta\mathcal{A}^*\left(\mathcal{A}b^{k+1/2} - y\right)\right), \end{aligned}$$

where $k$ is the iteration counter and $\tau_1 = 1$. The algorithm itself requires only multiplication by $\mathcal{A}$ and $\mathcal{A}^*$. The step length $\eta$ must satisfy $\eta \le 1/\sigma_{\max}(\mathcal{A})$ to guarantee convergence [8, Section 4.3]. It is also possible to use $\tau^k = \frac{k+2}{2}$ [5, Remark 10.35], as we do in the sequel. For our purposes, FISTA can be initialized with any vector $b^1$ which is selected independent of $y$, *i.e.*, we require that the Jacobian of $b^1$ with respect to $y$ always be the zero matrix. FISTA is almost always preferable to ISTA.

*c) ADMM:* The third algorithm we mention is the alternating direction method of multipliers (ADMM) [10], with iterations

$$\begin{aligned} b^{k+1} &= \mathbf{prox}_{\eta r}(z^k - u^k) \\ z^{k+1} &= (\eta\mathcal{A}^*\mathcal{A} + I)^{-1}(b^{k+1} + u^k + \eta\mathcal{A}^*y) \\ u^{k+1} &= u^k + b^{k+1} - z^{k+1}, \end{aligned}$$

where $u^k, z^k \in B$. For ADMM, the parameter $\eta$ can take any positive value. To compute the update step for $z^{k+1}$ we need to solve a positive-definite system of equations by only accessing $\mathcal{A}^*$ and $\mathcal{A}$. There are many methods to do this, for example, conjugate-gradient (CG) type methods [11], [12], [13].

For all of these algorithms, $b^k$ converges to a solution of (2). There are many other algorithms for evaluating these estimators; see, *e.g.*, [5], [14], [15], [16], [17]. The methods for computing SURE we describe below will work with most of these as well.

### G. Weak Differentiability of Convex Regularized Regression

For SURE to be an unbiased estimate of risk, the estimator $\hat{\mu}$ must be weakly differentiable [18, Section 6] and obey some integrability conditions [1]. For our purpose, it is sufficient to show that $\hat{\mu}$ is Lipschitz continuous [4, Lemma III.2].

We will now show that $\hat{\mu}$ is Lipschitz if $r$ is a closed convex proper function. The coefficient estimate $\hat{\beta}(y)$ minimizes $r(b) + \frac{1}{2}\|\mathcal{A}b - y\|_2^2$, and so by [19, Theorem 3.1.23], we have for all $w$,

$$\begin{aligned} r(w) &\ge r(\hat{\beta}(y)) + \langle \mathcal{A}^*y - \mathcal{A}^*\mathcal{A}\hat{\beta}(y) \mid w - \hat{\beta}(y) \rangle \\ &= r(\hat{\beta}(y)) + \langle y - \mathcal{A}\hat{\beta}(y) \mid \mathcal{A}w - \mathcal{A}\hat{\beta}(y) \rangle. \end{aligned}$$

Evaluating this at $w = \hat{\beta}(\tilde{y})$ gives

$$r(\hat{\beta}(\tilde{y})) \ge r(\hat{\beta}(y)) + \langle y - \mathcal{A}\hat{\beta}(y) \mid \mathcal{A}\hat{\beta}(\tilde{y}) - \mathcal{A}\hat{\beta}(y) \rangle,$$

and switching the roles of $y$ and $\tilde{y}$, we obtain

$$r(\hat{\beta}(y)) \ge r(\hat{\beta}(\tilde{y})) + \langle \tilde{y} - \mathcal{A}\hat{\beta}(\tilde{y}) \mid \mathcal{A}\hat{\beta}(y) - \mathcal{A}\hat{\beta}(\tilde{y}) \rangle.$$

Authorized licensed use limited to: Stanford University. Downloaded on September 01,2024 at 23:40:18 UTC from IEEE Xplore. Restrictions apply.

Adding these two inequalities yields

$$\begin{aligned}0 &\geq \langle y - \mathcal{A}\hat{\beta}(y) \mid \mathcal{A}\hat{\beta}(\tilde{y}) - \mathcal{A}\hat{\beta}(y)\rangle \\ &\quad + \langle \tilde{y} - \mathcal{A}\hat{\beta}(\tilde{y}) \mid \mathcal{A}\hat{\beta}(y) - \mathcal{A}\hat{\beta}(\tilde{y})\rangle \\ &= \langle y - \hat{\mu}(y) - \tilde{y} + \hat{\mu}(\tilde{y}) \mid \hat{\mu}(\tilde{y}) - \hat{\mu}(y)\rangle \\ &= \langle y - \tilde{y} \mid \hat{\mu}(\tilde{y}) - \hat{\mu}(y)\rangle + \|\hat{\mu}(\tilde{y}) - \hat{\mu}(y)\|_2^2.\end{aligned}$$

Re-arranging and using the Cauchy-Schwartz inequality gives

$$\begin{aligned}\|\hat{\mu}(\tilde{y}) - \hat{\mu}(y)\|_2^2 &\leq \langle \tilde{y} - y \mid \hat{\mu}(\tilde{y}) - \hat{\mu}(y)\rangle \\ &\leq \|\tilde{y} - y\|_2 \|\hat{\mu}(\tilde{y}) - \hat{\mu}(y)\|_2,\end{aligned}$$

eliminating a factor of $\|\hat{\mu}(\tilde{y}) - \hat{\mu}(y)\|_2$ shows that $\hat{\mu}$ is 1-Lipschitz.

## II. SURE-CR

### A. Randomized Trace Estimation

In this section we describe methods for estimating the trace of a $d \times d$ matrix $M$, that access $M$ only via an oracle that evaluates its adjoint, $v \mapsto M^* v$. We refer to this oracle as vector-matrix oracle, since it evaluates (the transpose of) $v^* M$. We will apply this to the specific matrix $M = D\hat{\mu}(y)$ to evaluate the divergence term in SURE.

The naïve approach is to use the oracle to evaluate $M^* e_i$, where $e_i$ is the $i$th unit vector, for $i = 1, \ldots, d$, whereupon we can readily evaluate

$$\mathbf{Tr}\, M = \sum_{i=1}^{d} e_i^* (M^* e_i).$$

When $d$ is very large, this is slow. It also evidently involves much wasted computation, since we end up computing all $d^2$ entries of $M$, only to sum the $d$ diagonal ones.

Randomized methods can be used to estimate $\mathbf{Tr}\, M$ using far fewer than $d$ evaluations of the adjoint mapping. These methods are based on the simple observation that if the random variable $Z \in \mathbf{R}^d$ satisfies $\mathbf{E}\, Z = 0$ and $\mathbf{E}\, ZZ^* = I$, then we have $\mathbf{E}\, Z^* M Z = \mathbf{Tr}\, M$. To approximate this we compute $m$ independent samples of $Z$, $z_1, \ldots, z_m$, and take the empirical mean as our estimate,

$$\mathbf{Tr}\, M \approx \frac{1}{m} \sum_{i=1}^{m} z_i^* (M^* z_i),$$

which is unbiased. In [20], Hutchinson showed that the variance of the error in this approximation is minimized if the $Z_i$'s are i.i.d. random variables taking values $\pm 1$, each with probability $1/2$, which is known as the Rademacher distribution.

Improvements on this basic randomized method were recently suggested by Meyer, Musco, Musco, and Woodruff in [21]. They proposed Hutch++, which uses a low-rank approximation of the matrix to project some queries away from large singular values of the matrix. Hutch++ is also an unbiased estimator of the trace, and consistently produces a good estimate of the trace using fewer queries to the vector-matrix oracle than the basic randomized method. Hutch++'s computation takes part in three phases, each of which requires an equal number of calls to the vector-matrix oracle, so the total number of queries is a multiple of 3. In the first phase, Hutch++ sketches the matrix; *i.e.*, it multiplies $M$ with a tall rectangular matrix whose entries are i.i.d. Rademacher random variables, and computes an orthogonalization of that matrix product, which is an estimate of the dominant dimensions of the matrix. In the second phase, it computes the exact trace of $M$ projected onto the dominant dimensions found via the sketch. In the third phase, it runs the Hutchinson estimator on $M$ projected away from those dominant dimensions.

In our method for evaluating SURE, we found that 34 queries per Hutch++ phase, for a total of 102 vector-matrix oracle calls, consistently produced high quality estimates of the trace. For small problems, *i.e.*, those of size less than or equal to 102, we exactly compute the trace without any randomization.

Subsequent works have developed alternative trace estimation algorithms [22], [23].

### B. Vector-Jacobian Oracles

In this section we describe methods for computing the adjoint oracle $v \mapsto (D\hat{\mu}(y))^* v$. Using $\hat{\mu}(y) = \mathcal{A}\hat{\beta}(y)$, we have $D\hat{\mu}(y) = \mathcal{A} D\hat{\beta}(y)$ and, therefore,

$$(D\hat{\mu}(y))^* v = \left(D\hat{\beta}(y)\right)^* (\mathcal{A}^* v).$$

So it suffices to evaluate the mapping $u \mapsto \left(D\hat{\beta}(y)\right)^* u$. Roughly speaking, we need to differentiate through the solution of the optimization problem (2), *i.e.*, the mapping from the data $y$ to the parameter estimate $\hat{\beta}(y)$.

*a) Differentiability:* In many cases $\hat{\mu}$ is not differentiable. However in Section I-G we showed that $\hat{\mu}$ is Lipschitz; by applying Rademacher's theorem, we know that $\hat{\mu}$ is a.e.-differentiable under the Lebesgue measure, and since $y$ has a Gaussian distribution $\hat{\mu}$ is almost surely differentiable at $y$ [18, Section 3.1.2].

*b) Generic methods:* Some recent work shows how to differentiate through the solution of some convex optimization problems (when the mapping is differentiable), for example [24] for quadratic programs (QPs) and [25] for cone programs. These methods in turn have been integrated into software frameworks for automatic differentiation such as PyTorch [26] and TensorFlow [27], [28]. Such libraries include CVXPYlayers, diffcp, and OPTNET [24], [25], [29]. All of these give methods for evaluating $u \mapsto D\hat{\beta}(y)^* u$, without forming the matrix $D\hat{\beta}(y)$. These generic methods work well for small problems and some medium-sized problems, but they do not scale to large scale problems. At non-differentiable points, these methods compute a heuristic quantity [30, Section 14].

*c) Differentiating through an iterative solver:* Another approach to differentiating through a convex problem relies on a solver or iterative solution algorithm, such as those described in Section I-F. Existing work differentiates through proximal operators to use them as non-linear activations in neural networks [31], [32], in this work, we differentiate through iterative optimization algorithms to approximate differentiating the solution map. Here we view the iterative algorithm as a sequence

Authorized licensed use limited to: Stanford University. Downloaded on September 01,2024 at 23:40:18 UTC from IEEE Xplore. Restrictions apply.

of mappings, *i.e.*, we view our iterative algorithm as applying an operator $F^k$ at each iteration such that

$$b^{k+1}, S^{k+1} = F^k(b^k, S^k, y)$$

where $S^k$ is any ancillary state in the algorithm (*e.g.*, in FISTA $S^k = b^{k-1}$ and in ADMM $S^k = (z^k, u^k)$). Suppose it takes $\ell$ iterations to converge to a reasonable tolerance, so $\hat{\mu}(y) \approx F^\ell(F^{\ell-1}(\ldots, y), y)$. By implicitly differentiating this recurrence and applying the chain rule, we obtain a series of equations that we use to compute $(D\hat{\mu}(y))^* v$, given a vector of output sensitivities $v$. Our approximation of $\hat{\mu}$ may be non-differentiable on a set of positive Lebesgue measure. In this situation, we need to compute a quantity that can serve as a surrogate for the true vector-Jacobian product. In neural network training, it is common to discuss the vector-Jacobian of a scalar loss function—which is simply the gradient—even when the loss function is non-differentiable. Many choices of surrogates for when differentiability fails have been proposed and seem to work well here [33], [34]. In Section III-B, our empirical results show that a continuous extension of the true derivative yields sufficiently accurate estimates at non-differentiable points of the derivative of $\hat{\mu}$ so that we still have a good estimate of the risk of $\hat{\mu}$.

As an example of differentiating our approximation of $\hat{\mu}$, we work through the derivative of ISTA. ISTA is straightforward to analyze because there is no ancillary state in the algorithm, but this method easily generalizes to the other algorithms from Section I-F. To simplify our equations, we let $b^{k+1/2} = b^k - \eta \mathcal{A}^* \left(\mathcal{A} b^k - y\right)$. By differentiating the ISTA iterations we obtain

$$\begin{aligned} Db^{k+1} &= D\mathbf{prox}_{\eta r}\left(b^{k+1/2}\right) Db^k - \eta \mathcal{A}^* \left(\mathcal{A} Db^k - I\right) \\ &= \left(D\mathbf{prox}_{\eta r}\left(b^{k+1/2}\right) - \eta \mathcal{A}^* \mathcal{A}\right) Db^k + \eta \mathcal{A}^*. \end{aligned}$$

In a forward pass, we can evaluate $b^{k+1/2}$ for $k = 1, \ldots, \ell - 1$ and cache them to enable the vector-Jacobian oracle evaluations. Evaluating $(Db^\ell)^* v$ then becomes a recursive problem, which can be computed using two of the oracles we needed for the forward pass—$\mathcal{A}$ and $\mathcal{A}^*$—and one new oracle: the vector-Jacobian oracle for the proximal operator. The base case for our recursion comes from our requirement that $b^1$ is chosen independently of $y$ *i.e.* that $(Db^1)^* v = 0$. The difficulty in this method relies in evaluating the vector-Jacobian oracle of the proximal operator.

*d) Evaluating proximal operator vector-Jacobian oracles:* For many proximal operators known in closed-form, the Jacobians are trivial to find in closed-form. For example, the $\ell_1$ norm has proximal operator given by soft-thresholding, $\mathcal{T}_\eta$. Since soft-thresholding occurs component-wise, this means that the Jacobian is a diagonal matrix, whose non-zero entries are 1 if $b_i^{k+1/2}$ is above the threshold, $-1$ if it is below the negative of the threshold, and 0 otherwise. In this case, it is possible to efficiently compute the vector-Jacobian oracle without forming the whole Jacobian to find

$$\left(D\mathbf{prox}_{\eta\|\cdot\|_1}\left(b^{k+1/2}\right)\right)^* u = \mathbf{diag}\left(J_{\mathcal{T}_\eta}\left(b^{k+1/2}\right)\right) \circ u,$$

where $a \circ b$ denotes Hadamard or component-wise multiplication. Here we handle the points of non-differentiability by using the value of the derivative at a point in a very small neighborhood of the non-differentiable point. In particular, since non-differentiability occurs only when entries of $X^T y$ are exactly equal to $\eta$, we can instead interpret our choice of a point in the neighborhood as evaluating the derivative at a point within the floating point uncertainty of our vector.

Other closed-form proximal operators have non-trivial Jacobians. For example, the proximal operator of the nuclear norm is given by

$$\mathbf{prox}_{\eta\|\cdot\|_*}(b^{k+1/2}) = U\mathcal{T}_\eta(\Sigma)V^*,$$

where $b^{k+1/2} = U\Sigma V^*$ is the singular-value decomposition of $b^{k+1/2}$ and $\mathcal{T}_\eta$ is soft-thresholding on $\Sigma$. This has a non-trivial Jacobian because of the multi-valued nature of the SVD in the presence of repeated singular values. However, since all proximal operators are Lipschitz, we know that it is a.e.-differentiable. [4, Lemma IV.2] gives closed-form expressions for the Jacobian of this proximal operator that hold for simple and full-rank matrices. However, it is common that later iterations will involve low-rank matrices, which requires us to select an approximation of the vector-Jacobian products. We use the continuous extension of the closed-form vector-Jacobian product, which exists for all matrices which do not have any singular values exactly equal to $\eta$. We derive an expression for this extension in Section C. For matrices with singular values exactly equal to $\eta$ we just evaluate at a point within the neighborhood of the matrix similar to how we handle the $\ell_1$ norm.

In general, when trying to apply SURE-CR to a new proximal operator, it is necessary to be able to evaluate the vector-Jacobian product for that proximal operator. If the proximal operator has points of non-differentiability which are reached by the iterative algorithm, then it is necessary to choose a surrogate for the vector-Jacobian product. The accuracy of SURE-CR is limited by the accuracy of the vector-Jacobian product oracle.

Since $(D\hat{\mu}(y))^* v = \nabla_y \langle \hat{\mu}(y) \mid v \rangle$, it is possible to apply well-known strategies to compute the gradient of a scalar-valued function. Most notably, reverse-mode automatic differentiation automates much of this section's work [35]. For many proximal operators with closed-form expressions, reverse-mode automatic differentiation can differentiate the proximal operator without an analytic derivation of a closed-form for the vector-Jacobian oracle.

As an example we work out how to construct the oracle for $r(b) = \|b\|_1 + \|b\|_2^2$ (a weighted sum of these norms is the regularizer in the elastic net [36]). The proximal operator can be evaluated by applying separability to find that it is given by a scaled form of soft-thresholding,

$$\begin{aligned} \mathbf{prox}_{\eta r}(v) &= \operatorname*{argmin}_b \left(\eta\|b\|_1 + \eta\|b\|_2^2 + \frac{1}{2}\|b - v\|_2^2\right) \\ &= \frac{1}{1+2\eta}\mathcal{T}_\eta(v). \end{aligned}$$

By rewriting soft-thresholding as

$$\mathcal{T}_\eta(v) = (v - \eta\mathbf{1})_+ - (-v - \eta\mathbf{1})_+,$$

Authorized licensed use limited to: Stanford University. Downloaded on September 01,2024 at 23:40:18 UTC from IEEE Xplore. Restrictions apply.

we can express this function in terms of elementary operations that are commonly supported by automatic differentiation libraries, meaning no work is required to construct the vector-Jacobian oracle.

### C. Implementation

We have implemented the methods described above in SURE-CR, an open-source package available at https://github.com/cvxgrp/SURE-CR. It supports divergence computation via CVXPYlayers as well as via differentiation through FISTA and ADMM, and uses Hutch++ to estimate the divergence.

SURE-CR relies on an existing computational graph library, pyTorch [26], to enable GPU-acceleration in our solvers and to enable reverse-mode automatic differentiation. We have implemented a library to encode the linear operator $\mathcal{A}$ as a computational flow graph. It is available at https://github.com/cvxgrp/torch_linops. This library adapts Barratt's preconditioned conjugate gradient implementation [37] and implements randomized preconditioners including Nyström preconditioning [38].

By differentiating through FISTA and ADMM iterations, SURE-CR is able to scale to large problems. For example it can evaluate SURE for a matrix completion problem with $b \in \mathbf{R}^{2000\times 1000}$ and $10\%$ of entries revealed, for which $D\hat{\mu}(y)$ is a $10^5 \times 10^5$ matrix (which of course is never formed) in 120 seconds on the server described in Section III.

To apply SURE-CR to novel problems and regularizers, the user should adapt an example from Section A by implementing their linear operator $\mathcal{A}$ and $\mathcal{A}^*$ as shown in Sections A-C and implementing the proximal operator as a differentiable torch function. This can be done most easily by expressing it as the composition of built-in torch functions as shown in Sections A-B. In the event that a heuristic is used for the derivative of the proximal operator, it may be valuable to test that the heuristic and the true vector-Jacobian products found by CVXPYlayers agree.

SURE-CR currently uses at most one GPU; however, in hyperparameter sweep problems, users can run different experiments on different GPUs in parallel.

## III. Numerical Examples

In this section we report results of numerical examples of SURE-CR. We consider three problems, LASSO, matrix completion, and robust PCA, and for each one, problem instances ranging from small to large. For each instance we evaluate various estimates of SURE, as well as an estimate of the $\ell_2$ risk obtained via a Monte Carlo method described below.

We carry out a few additional experiments that analyze the variance contributed by SURE itself in high-dimensions, and also, the variance contributed by our use of a randomized trace estimator. We will see that the latter is substantially smaller than the former.

Finally, in our last example, we show how SURE-CR can be used to carry out hyperparameter selection.

*a) Hyperparameter selection:* When selecting regularization parameters, we swept over the parameters—equally spaced on a logarithmic scale—on the largest problem size we planned to run. We then selected a value which had risk less than half the risk of the maximum likelihood estimator of $\mu$ and had a high iteration count relative to the other runs in the sweep. We require the risk to be small in order to demonstrate SURE-CR in problem settings where the estimator is useful. The higher the iteration count, the longer SURE-CR takes to run since we have to differentiate through more iterations of the solver algorithm; accordingly, to give a better sense of worst-case runtime when using SURE-CR we prefer problem instances that gave higher iteration counts.

*b) SURE estimates:* In our first example, LASSO, we report the value of the analytical expression for SURE. In all examples we evaluate SURE using CVXPYlayers, where it was possible, *i.e.*, for the smaller problem instances. For each problem we use either ADMM or FISTA, depending on which was faster on small test problems.

*c) Monte Carlo $\ell_2$ risk estimate:* Since we are using synthetic data and know $\mu = \mathcal{A}\beta$, we are able to use a Monte Carlo method to approximate the risk as

$$R(\hat{\mu}) \approx \frac{1}{m}\sum_{i=1}^{m} \|\hat{\mu}(y_i) - \mu\|_2^2,$$

where $y_i \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu, \sigma^2 I)$. (In practical problem settings, this Monte Carlo estimation of $\ell_2$ risk is not possible.)

*d) Computational platform:* We report timings for running SURE-CR on the Stanford University Institute for Computational and Mathematical Engineering's DGX-1, with 8 Nvidia Tesla V100-SXM2-32GB-LS GPUs, an Intel Xeon E5-2698 v4 with 80 cores, 540GiB of memory, and 32GiB of GPU memory per GPU. (However, we were limited to only one GPU during our tests.)

*e) Overview of results:* The results are summarized in the tables below. Comparing the values of the various estimates of SURE and $\ell_2$ risk across each row, we see that there is good agreement, except for the smallest problem instances. In Section B, we show that recent works by Bellec and Zhang [39], [40] enables bounding the variance of SURE to be less than $4\sigma^4 d + 2\sigma^2 R(\hat{\mu})$. For our estimators, the risk scales about affinely with $d$, and therefore the standard deviation of SURE grows slower than its expectation, so we see asymptotic convergence to the true value in relative error.

For the largest instances of matrix completion and robust PCA, each of which have 2 million parameters, we are able to compute SURE in under two minutes. To our knowledge, there was no previously known method for computing SURE for such large instances.

### A. LASSO

We compute SURE for LASSO problems, described in Section I-D. We consider under-determined problems with $p = 2d$, for $d = 250, 500, 2500, 5000, 25000$.

Authorized licensed use limited to: Stanford University. Downloaded on September 01,2024 at 23:40:18 UTC from IEEE Xplore. Restrictions apply.

TABLE I
VALUES OF COORDINATE-WISE SURE ESTIMATES AND COMPUTATION TIMES FOR FIVE LASSO PROBLEM INSTANCES. COORDINATE-WISE SURE IS GIVEN BY $(1/d)\text{SURE}(\hat{\mu}, y)$ AND IS USED TO IMPROVE READABILITY. TIMES GIVEN IN SECONDS

| Dimensions | | CVXPYlayers | | FISTA | | Analytic | MC risk |
|---|---|---|---|---|---|---|---|
| $d$ | $p$ | Value | Time | Value | Time | | |
| 250 | 500 | 0.51 | 252 | 0.51 | 2.54 | 0.52 | 0.48(<0.005) |
| 500 | 1000 | 0.43 | 1929 | 0.37 | 2.89 | 0.40 | 0.50(<0.005) |
| 2500 | 5000 | * | * | 0.58 | 3.10 | 0.57 | 0.51(<0.005) |
| 5000 | 10000 | * | * | 0.47 | 9.19 | 0.46 | 0.53(<0.005) |
| 25000 | 50000 | * | * | 0.58 | 287 | 0.58 | 0.54(<0.005) |

*a) Data generation:* We draw the entries of the data matrix i.i.d. from a standard normal distribution. We pick $\beta$ with $d/20$ nonzero entries equal to a constant and use $\sigma^2 = 2$. We pick the value of the nonzero coefficients so that $\frac{\|\mu\|_2^2}{\|\mu\|_2^2 + d\sigma^2} = 0.8$. We sample one $y$ independently from the rest of the data and select $\lambda = 0.1\lambda_{\max}$ (defined in (3)).

For each instance we use SURE-CR with CVXPYlayers, SURE-CR with FISTA, the analytic SURE value computed against CVXPY's solution, and the Monte Carlo estimate of the risk using CVXPY to solve the optimization problem. In its default configuration, CVXPYlayers has very low accuracy in moderate dimensions and does not raise warnings about the errors. To correct for this, we switched CVXPYlayer's implicit linear system solver for its direct linear system solver; this did not significantly impact runtime on problems for which it was giving accurate results. We present both the risk and time values for each. When using CVXPYlayers, we report the value as $*$ when CVXPYlayers has a non-standard return status warning, raises an error, or takes more than 12 hours. The seed used to generate the Hutch++ queries and the sample point at which to compute SURE are the same for all problems of a given size. The results are given in Table I.

## B. Matrix Completion

We compute SURE for matrix completion problems, described in Section I-E. For all problems, we use $d = 0.1mn$, $\sigma^2 = 2$, and $\lambda = 0.25\lambda_{\max}$. For the large problems used to generate Fig. 1, we use $m = 2000$, $n = 1000$, $d = 0.1mn = 200000$. We use SURE-CR with CVXPYlayers and SURE-CR with ADMM to compute SURE in Table II. We describe how we formed $\mu$ and $\beta$ below.

Since $\mathcal{A}^*\mathcal{A} + \lambda I$ is a diagonal matrix, we replace the preconditioned conjugate gradient step of the ADMM updates with an exact inverse. This has no significant impact on the numerical accuracy of our algorithm, but does improve its runtime.

*a) Data generation:* We first generate $\beta = U\Sigma V^* \in \mathbf{R}^{m\times n}$ with $\max(5, 0.02n)$ non-zero singular values, which are uniformly distributed over $[0, n]$. The matrices $U$ and $V^*$ are generated by computing the SVD of a matrix where each entry is independent and identically distributed as uniform over $[0, 1]$. For the selection operator $\mathcal{A}$, we selected $10\%$ of the entries at random without replacement. We then sampled $y \sim \mathcal{N}(\mathcal{A}\beta, \sigma^2 I)$.

*b) Quantifying Hutch++ uncertainty:* We verify that the uncertainty from using Hutch++ to estimate the divergence is dominated by the uncertainty inherent in SURE. For 20 sample points of $y$, we ran SURE-CR on each point 100 times. In Fig. 1, we show in blue the distribution of the relative error between the SURE-CR values and the sample mean of the SURE-CR runs on that point: let $\text{SURE-CR}(y, i)$ denote the random variable of the output of running SURE-CR on a point $y$ with seed $i$. Then for samples $y_1, y_2, \ldots, y_{10\,020}$, we plot the histogram of

$$\frac{\text{SURE-CR}(\hat{\mu}, y_i, 100i + j) - 100^{-1}\sum_{k=1}^{100}\text{SURE-CR}(\hat{\mu}, y_i, 100i + k)}{10\,000^{-1}\sum_{k=21}^{10\,020}\|\hat{\mu}(y_k) - \mu\|_2^2}$$

for $i = 1, 2, \ldots, 20$ and $j = 1, 2, \ldots, 100$. We also plot the histogram of the relative error between 2000 evaluations of

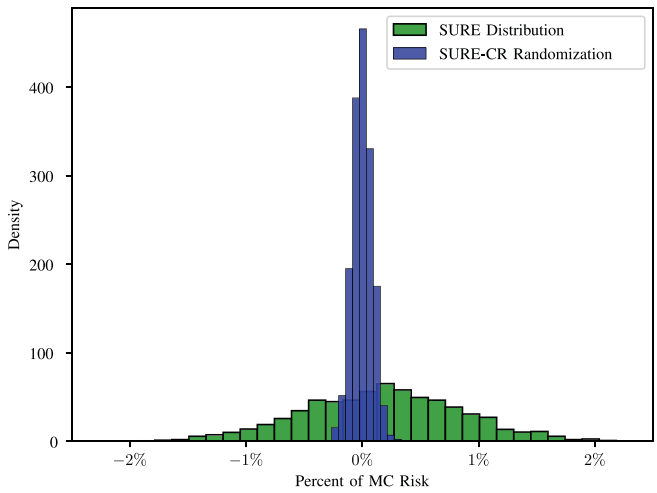


Fig. 1. The green histogram is the relative error between SURE at various sample points against the Monte Carlo risk. The blue histogram shows the relative error between SURE-CR at a sample point and the mean of 100 runs of SURE-CR at that point.

TABLE II
VALUES OF COORDINATE-WISE SURE ESTIMATES AND COMPUTATION TIMES FOR FIVE MATRIX COMPLETION PROBLEM INSTANCES. COORDINATE-WISE SURE IS GIVEN BY $(1/d)\text{SURE}(\hat{\mu}, y)$ AND IS USED TO IMPROVE READABILITY. TIMES GIVEN IN SECONDS

| Dimensions | | CVXPYlayers | | ADMM | | MC risk |
|---|---|---|---|---|---|---|
| $d$ | $p$ | Value | Time | Value | Time | |
| 20 | 200 | 1.15 | 1.51 | 1.16 | 5.20 | 1.33(0.01) |
| 500 | 5000 | 0.86 | 2246 | 0.88 | 49.9 | 0.96(<0.005) |
| 2000 | $2 \times 10^4$ | 0.84 | 15866 | 0.84 | 45.1 | 0.90(<0.005) |
| $5 \times 10^4$ | $5 \times 10^5$ | * | * | 1.69 | 41.2 | 1.70(<0.005) |
| $2 \times 10^5$ | $2 \times 10^6$ | * | * | 0.74 | 114 | 0.74(<0.005) |

Authorized licensed use limited to: Stanford University. Downloaded on September 01,2024 at 23:40:18 UTC from IEEE Xplore. Restrictions apply.

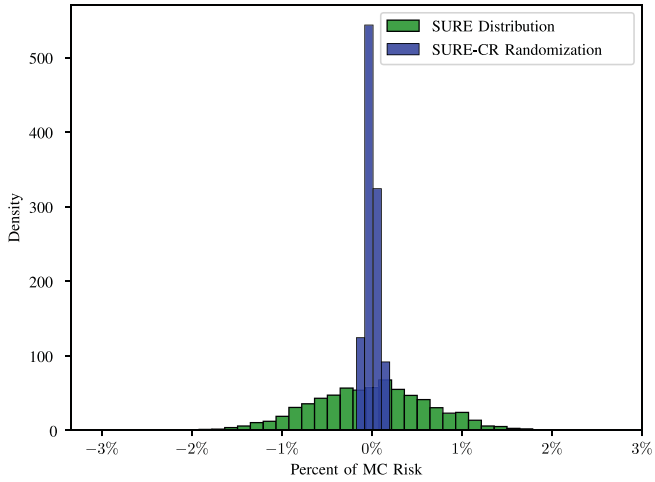


Fig. 2. The green histogram is the relative error between SURE at various sample points against the Monte Carlo risk. The blue histogram shows the relative error between SURE-CR at a sample point and the mean of 100 runs of SURE-CR at that point.

SURE-CR and the Monte Carlo estimation of the risk. The uncertainty from the algorithm's randomization is small compared to SURE's uncertainty.

*c) SURE as estimate of risk:* The green histogram in Fig. 1 shows that SURE-CR is within $2.5\%$ of the Monte Carlo risk at 2000 independent sample points. Precisely, the green histogram shows the histogram of the quantity

$$\frac{\text{SURE-CR}(\hat{\mu}, y_i, i) - 10\,000^{-1} \sum_{j=2001}^{12\,000} \|\hat{\mu}(y_j) - \mu\|_2^2}{10\,000^{-1} \sum_{j=2001}^{12\,000} \|\hat{\mu}(y_j) - \mu\|_2^2}$$

for $i = 1, 2, \ldots, 2000$ and independent samples $y_1$, $y_2, \ldots, y_{12\,000}$. This shows SURE is a good estimate of the true risk.

*d) Non-differentiability:* In around $5\%$ of the 2000 samples used to generate the green histogram in Fig. 1, we observed that our approximation of $\hat{\mu}$ was non-differentiable. We detected this by running our algorithm without using the extension of the derivative and seeing what percentage of runs encountered numerical issues caused by repeated or zero singular values. We then ran the experiment using the extension of the derivative, and report those values here. Notably, those samples are indistinguishable from the other samples in the histogram, showing that our heuristic is effective at approximating the vector-Jacobian products for $\hat{\mu}$ and still providing a good estimate of risk.

## C. Robust PCA

We also tested SURE on robust PCA problems, described in Section I-E. For all problems, we use $m = n$, $\sigma^2 = 2$, $\lambda = 0.16\lambda_{\max}$, and $\gamma = 0.057\gamma_{\max}$. For the large problems used to generate Fig. 2, we used $m = n = 1000$. We use SURE-CR with CVXPYlayers and SURE-CR with ADMM to compute SURE in Table III.

*a) Data generation:* We select $S$ with $\max(10, 10^{-4}n^2)$ non-zero entries drawn from a uniform distribution over $[0, 100]$. We select $L$ with rank $\max(5, 0.02n)$ and singular values distributed uniformly over $[0, n]$. We sampled $y \sim \mathcal{N}(L + S, \sigma^2 I)$.

TABLE III
VALUES OF COORDINATE-WISE SURE ESTIMATES AND COMPUTATION TIMES FOR FIVE ROBUST PCA PROBLEM INSTANCES. COORDINATE-WISE SURE IS GIVEN BY $(1/d)\text{SURE}(\hat{\mu}, y)$ AND IS USED TO IMPROVE READABILITY. TIMES GIVEN IN SECONDS

| Dimensions | | CVXPYlayers | | ADMM | | MC risk |
|---|---|---|---|---|---|---|
| $d$ | $p$ | Value | Time | Value | Time | |
| 100 | 200 | 5.14 | 1.15 | 5.13 | 16.0 | 5.01(0.006) |
| 2500 | 5000 | 0.53 | 115 | 0.53 | 19.9 | 0.59(<0.005) |
| 10000 | $2 \times 10^4$ | 0.31 | 1116 | 0.31 | 21.5 | 0.34(<0.005) |
| $2.5 \times 10^5$ | $5 \times 10^5$ | * | * | 0.27 | 22.1 | 0.27(<0.005) |
| $1 \times 10^6$ | $2 \times 10^6$ | * | * | 0.44 | 31.4 | 0.44(<0.005) |

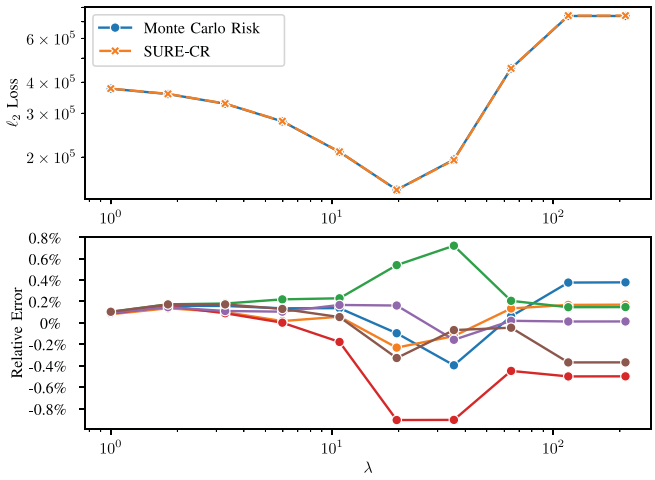


Fig. 3. *Top*. SURE-CR and Monte Carlo estimate of $\ell_2$ risk as a function of the hyperparameter. A single sample of $y$ was used for all of the SURE-CR runs. The two lines are visually indistinguishable. *Bottom*. Relative error plots for the SURE-CR sweep run on 6 independent samples of $y$. The Monte Carlo estimate and the computed SURE value differ by less than $1\%$.

*b) SURE as estimate of risk:* Fig. 2 shows the histogram of the relative error compared to the Monte Carlo estimate of the risk for $m = n = 1000$ and the histogram of the variance from the randomization in SURE-CR for $m = n = 1000$. We ran SURE-CR on 2000 sample points and use $10\,000$ samples for the Monte Carlo estimate. We observed only one sample for which SURE-CR diverged from the Monte Carlo risk by more than $3\%$.

## D. SURE for Hyperparameter Selection

In this experiment, we aim to select an optimal hyperparameter for matrix completion. We use the same setup as in Section III-B with $m = 2000$ and $n = 1000$, except we now draw a single sample $y$.

We then ran a grid search over $\lambda$, varying it exponentially over $[1, 2\lambda_{\max}]$, where $\lambda_{\max}$ is the smallest $\lambda$ for which $\hat{\beta}(y) = 0$. We drew a single sample of $y$, and then for each $\lambda$ we ran SURE-CR with ADMM. We then computed a Monte Carlo estimation of the risk for each $\lambda$. Fig. 3, shows that the risk versus $\lambda$ curves are visually indistinguishable. We also show

Authorized licensed use limited to: Stanford University. Downloaded on September 01,2024 at 23:40:18 UTC from IEEE Xplore. Restrictions apply.

that for 6 independent samples of $y$, the relative error was consistently below $0.9\%$.

## APPENDIX A
## CODE EXAMPLES

### A. CVXPYlayers — LASSO

This code sample demonstrates how to use SURE-CR with CVXPYlayers. It is based on the code used in Section III-A.

```
import cvxpy as cp

import surecr

X, y, variance, lambda_val = ...  # Generate data

beta_cvx = cp.Variable(X.shape[1])
y_cvx = cp.Parameter(y.shape[0])
prob = cp.Problem(
        cp.Minimize(
            1/ 2 * cp.sum_squares(X @ beta_cvx - y_cvx)
            + lambda_val * cp.pnorm(beta_cvx, 1)
        ))
solver = surecr.CVXPYSolver(prob, y_cvx, [beta_cvx], lambda b: X @ b)
sure = surecr.SURE(variance, solver)
cvx_sure_val = sure.compute(y)
```

### B. LASSO

This code sample demonstrates how to use SURE-CR with FISTA and how to define a custom proximal operator. It is based on the code used in Section III-A.

```
import torch
import surecr
import linops as lo

X, y, variance, lambda_val = ... # Generate data

d, p = X.shape
A = lo.aslinearoperator(X.cuda())
y_cuda = y.cuda()
def prox(v, t):
    return torch.relu(v - lambda_val * t) - torch.relu(-v - lambda_val * t)

solver = surecr.FISTASolver(
        A, prox, torch.zeros(p).cuda(),
        device=y_cuda.device)
sure = surecr.SURE(variance, solver)

sure_val = sure.compute(y_cuda)
```

### C. Matrix Completion

This code sample demonstrates how to use SURE-CR with ADMM and how to define a custom linear operator. It is based on the code used in Section III-B.

```
import torch
import surecr
import surecr.prox_lib as pl
import linops as lo

revealed_indices, y, variance, lambda_val, m, n = ... # Generate data


class SelectionOperator(lo.LinearOperator):
    def __init__(self, shape, idxs):
        self._shape = shape
        self._adjoint = _AdjointSelectionOperator(idxs,
                (self._shape[1], self._shape[0]), self)
        self._idxs = idxs

    def _matmul_impl(self, X):
        return X[self._idxs]

    def solve_I_p_lambda_AT_A_x_eq_b(self, lambda_, b):
        LHS = torch.ones_like(b)
        LHS[self._idxs] += lambda_
        return b / LHS

class _AdjointSelectionOperator(lo.LinearOperator):
    def __init__(self,  idxs, shape, adjoint):
        self._shape = shape
        self._adjoint = adjoint
        self._idxs = idxs

    def _matmul_impl(self, y):
        z = torch.zeros(self.shape[0], dtype=y.dtype, device=y.device)
        z[self._idxs] = y
        return z.reshape(-1)

d = len(revealed_indices)
p = m * n
A = SelectionOperator((d, p), revealed_indices)
y_cuda = y.cuda()

# Generates a function that applies singular value thresholding, which uses a
# continous extension of the derivative for the .backward method.
prox = pl.make_scaled_prox_nuc_norm((m, n), lambda_val)

solver = surecr.ADMMSolver(A, prox, torch.zeros(p).cuda(), device=y_cuda.device)
sure = surecr.SURE(variance, solver)

sure_val = sure.compute(y_cuda)
```

### D. Robust PCA

This code sample demonstrates how to use SURE-CR with ADMM and how to use advanced features of torch_linops to generate the linear operator. It is based on the code used in Section III-C.

```
import torch
import surecr
import surecr.prox_lib as pl
import linops as lo

y, variance, lambda_val, gamma_val, m, n = ... # Generate data

d = m * n
p = 2 * d
A = lo.hstack([lo.IdentityOperator(d), lo.IdentityOperator(d)])
y_cuda = y.cuda()

# Generates a function that applies singular value thresholding, which uses a
# continous extension of the derivative for the .backward method.
prox_L = pl.make_scaled_prox_nuc_norm((m, n), lambda_val)

def prox_S(v, t):
    return torch.relu(v - gamma_val * t) - torch.relu(-v - gamma_val * t)

def prox(v, t):
    return torch.hstack([
            prox_L(v[:d], t), prox_S(v[d:], t)
    ])

solver = surecr.ADMMSolver(A, prox, torch.zeros(p).cuda(), device=y_cuda.device)
sure = surecr.SURE(variance, solver)

sure_val = sure.compute(y_cuda)
```

## APPENDIX B
## BOUND ON THE VARIANCE OF SURE

In [40, Theorem 3.2], it is shown that for convex regularized regression

$$\mathbf{var}(\mathrm{SURE}(\hat{\mu}, y)) \leq \mathbf{E}[(\mathrm{SURE}\ (\hat{\mu}, y) - \|\hat{\mu}(y) - \mu\|_2^2)^2] + \sigma^4 d$$

and

$$(\mathrm{SURE}(\hat{\mu}, y) - \|\hat{\mu}(y) - \mu\|_2^2)^2 \leq 2\sigma^2(\|y - \hat{\mu}(y)\|_2^2 + \mathrm{SURE}(\hat{\mu}, y))$$

almost surely. By applying algebraic manipulation and SURE's unbiasedness, we can find that

$$\mathbf{var}(\mathrm{SURE}(\hat{\mu}, y)) \leq 3\sigma^4 d - 4\sigma^4\, \mathbf{E}[\nabla \cdot \hat{\mu}(y)] + 4\sigma^2 R(\hat{\mu}).$$

In [39, Proposition 5.3], it is shown that $D\hat{\mu}(y)$ is almost surely positive semi-definite. This suggests that $\nabla \cdot \hat{\mu}(y) = \mathbf{Tr}(D\hat{\mu}(y)) \geq 0$ almost surely and lets us conclude that

$$\mathbf{var}(\mathrm{SURE}(\hat{\mu}, y)) \leq 3\sigma^4 d + 4\sigma^2 R(\hat{\mu}).$$

## APPENDIX C
## DIFFERENTIATING THE PROXIMAL OPERATOR OF THE NUCLEAR NORM

The proximal operator of the nuclear norm is given by a spectral function $F(X)$ such that $F(X) = UF(\Sigma)V^T$ where

Authorized licensed use limited to: Stanford University. Downloaded on September 01,2024 at 23:40:18 UTC from IEEE Xplore. Restrictions apply.

$U, \Sigma, V^T$ are the full SVD of $X$ and where $F(\Sigma)$ applies the function $\mathcal{T}_\eta(\sigma) = (\sigma - \eta)_+$ elementwise to all entries of $\Sigma$.

The function is non-differentiable when $X$ has repeated singular values, any singular values equal to 0, or any singular values equal to $\eta$. Formally, the mapping $X \mapsto (DF(X))^* Z$ for a fixed matrix $Z$, is only defined when $X$ has all distinct singular values and no singular values equal to 0 or $\eta$. However, it turns out there exists a function continuous on the set of matrices with no singular values equal to $\eta$, which is equal to the mapping $X \mapsto (DF(X))^* Z$, wherever that mapping is defined. We refer to this function as a continuous extension. In this section, we find the continuous extension of $X \mapsto (DF(X))^* Z$ for all fixed $Z$.

We assume that $X, Z, \Sigma, \zeta, \Gamma, \Delta \in \mathbf{R}^{m\times n}$, $U \in \mathbf{R}^{m\times m}$, $V \in \mathbf{R}^{n\times n}$, and that $\Omega_U, \Omega_V, \Omega_\Sigma$ are linear operators from $\mathbf{R}^{m\times n}$ to $\mathbf{R}^{m\times n}$. Without loss of generality, we assume $m \geq n$. A simple matrix is one without repeated singular values.

### A. Gradient for Full-Rank and Simple Matrices

[4] gives that for simple and full-rank $X$:

$$(DF(X))\,\Delta = U\left((\Omega_U\Delta)F(\Sigma) + (\Omega_\Sigma\Delta) + F(\Sigma)(\Omega_V\Delta)\right)V^T$$

where

$$(\Omega_U\Delta)_{ij} = \begin{cases} 0 & \text{if } i = j \\ -\frac{1}{\sigma_i^2-\sigma_j^2}\big(\sigma_j(U^T\Delta V)_{ij} & \\ \qquad +\sigma_i(U^T\Delta V)_{ji}\big) & \text{if } i\neq j \wedge i\leq n \\ \frac{1}{\sigma_j}(U^T\Delta V)_{ij} & \text{else} \end{cases},$$

$$(\Omega_V\Delta)_{ij} = \begin{cases} 0 & \text{if } i = j \\ \frac{1}{\sigma_i^2-\sigma_j^2}\big(\sigma_i(U^T\Delta V)_{ij} & \\ \qquad +\sigma_j(U^T\Delta V)_{ji}\big) & \text{else} \end{cases},$$

and

$$(\Omega_\Sigma\Delta)_{ij} = \begin{cases} \mathcal{T}'_\eta(\sigma_i)(U^T\Delta V)_{ii} & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}.$$

In order to find the adjoint of this mapping we begin by constructing a convenient orthonormal basis of $\mathbf{R}^{m\times n}$. We then project the desired quantity $(DF(X))^*Z$ onto the basis vectors. We can then weight and sum the basis elements to form $(DF(X))^*Z$.

Let $\{E^{ij}\}_{i,j\in[m]\times[n]}$ be the standard basis of $\mathbf{R}^{m\times n}$, *i.e.*, $E^{ij}_{k\ell} = 1$ iff $i = k$ and $j = \ell$ and is otherwise 0. Let $\Delta^{ij} = u_i v_j^T$. Critically, $U^T\Delta^{ij}V = E^{ij}$ which will greatly simplify the mappings given above. For notational simplicity, let $\zeta = U^T Z V$.

Evaluating the projection yields

$$\langle (DF(X))^* Z \mid \Delta^{ij}\rangle = \begin{cases} \mathcal{T}'_\eta(\sigma_i)\zeta_{ii} & \text{if } i = j \\ \frac{\mathcal{T}_\eta(\sigma_j)}{\sigma_j}\zeta_{ij} & \text{if } i > n \\ \frac{\sigma_i\mathcal{T}_\eta(\sigma_i)-\sigma_j\mathcal{T}_\eta(\sigma_j)}{\sigma_i^2-\sigma_j^2}\zeta_{ij} & \\ \quad +\frac{\sigma_j\mathcal{T}_\eta(\sigma_i)-\sigma_i\mathcal{T}_\eta(\sigma_j)}{\sigma_i^2-\sigma_j^2}\zeta_{ji} & \text{else} \end{cases}.$$

This projection is not defined for some basis elements whenever there exists $i \neq j$ such that $\sigma_i = \sigma_j$ or $i$ such that $\sigma_i = 0$.

### B. Extension by Continuity to All Matrices

Following [4], we seek to extend the projection of $(DF(X))^*Z$ by continuity to the situation where there exists $i \neq j$, such that $\sigma_i = \sigma_j$ or there exists $\sigma_i = 0$. Note that the projection is only ill-defined for basis elements $\Delta^{ij}$ such that $i \leq n$ and $i \neq j$. Since simple and full-rank matrices are dense in $\mathbf{R}^{m\times n}$, we will consider a sequence of matrices $X^{(k)}$ such that each $X^{(k)}$ is simple and full-rank and $\lim_{k\to\infty} X^{(k)} = X$.

From [4], we have that for $i \neq j$ such that $\sigma_i = \sigma_j > 0$,

$$\frac{\sigma_i^{(k)}\mathcal{T}_\eta(\sigma_i^{(k)}) - \sigma_j^{(k)}\mathcal{T}_\eta(\sigma_j)}{\left(\sigma_i^{(k)}\right)^2 - \left(\sigma_j^{(k)}\right)^2}\zeta_{ij} \to \left(\frac{1}{2}\mathcal{T}'_\eta(\sigma_i) + \frac{1}{2}\frac{\mathcal{T}_\eta(\sigma_i)}{\sigma_i}\right)\zeta_{ij},$$

and that for $i \neq j$ such that $\sigma_i = \sigma_j = 0$,

$$\frac{\sigma_i^{(k)}\mathcal{T}_\eta(\sigma_i^{(k)}) - \sigma_j^{(k)}\mathcal{T}_\eta(\sigma_j)}{\left(\sigma_i^{(k)}\right)^2 - \left(\sigma_j^{(k)}\right)^2}\zeta_{ij} \to \mathcal{T}'_\eta(0)\zeta_{ij}.$$

A symmetric version of the argument from [4] gives that for $i \neq j$ such that $\sigma_i = \sigma_j > 0$,

$$\frac{\sigma_j^{(k)}\mathcal{T}_\eta(\sigma_i^{(k)}) - \sigma_i^{(k)}\mathcal{T}_\eta(\sigma_j)}{\left(\sigma_i^{(k)}\right)^2 - \left(\sigma_j^{(k)}\right)^2}\zeta_{ji} \to \left(\frac{1}{2}\mathcal{T}'_\eta(\sigma_i) - \frac{1}{2}\frac{\mathcal{T}_\eta(\sigma_i)}{\sigma_i}\right)\zeta_{ji},$$

and for $i \neq j$ such that $\sigma_i = \sigma_j = 0$,

$$\frac{\sigma_j^{(k)}\mathcal{T}_\eta(\sigma_i^{(k)}) - \sigma_i^{(k)}\mathcal{T}_\eta(\sigma_j)}{\left(\sigma_i^{(k)}\right)^2 - \left(\sigma_j^{(k)}\right)^2}\zeta_{ji} \to 0.$$

Lastly, note that when $\sigma_j = 0$,

$$\lim_{\sigma_j^{(k)}\to 0}\frac{\mathcal{T}_\eta(\sigma_j^{(k)})}{\sigma_j^{(k)}} = \mathcal{T}'_\eta(0).$$

In summary, the continuous extension of $\langle (DF(X))^*Z, \Delta^{ij}$ for all $X$ is given by

$$\Gamma_{ij} = \begin{cases} \mathcal{T}'_\eta(\sigma_i)\zeta_{ii} & \text{if } i = j \\ R(\sigma_j)\zeta_{ij} & \text{if } i > n \\ Q(\sigma_i,\sigma_j)\zeta_{ij} + T(\sigma_i,\sigma_j)\zeta_{ji} & \text{else} \end{cases}$$

where

$$R(\sigma) = \begin{cases} \frac{\mathcal{T}_\eta(\sigma)}{\sigma} & \text{if } \sigma > 0 \\ \mathcal{T}'_\eta(\sigma) & \text{if } \sigma = 0 \end{cases},$$

$$Q(\sigma_i,\sigma_j) = \begin{cases} \frac{1}{2}\mathcal{T}'_\eta(\sigma_i) + \frac{1}{2}\frac{\mathcal{T}_\eta(\sigma_i)}{\sigma_i} & \text{if } \sigma_i = \sigma_j > 0 \\ \mathcal{T}'_\eta(0) & \text{if } \sigma_i = \sigma_j = 0 \\ \frac{\sigma_i\mathcal{T}_\eta(\sigma_i)-\sigma_j\mathcal{T}_\eta(\sigma_j)}{\sigma_i^2-\sigma_j^2} & \text{else} \end{cases},$$

and

$$T(\sigma_i,\sigma_j) = \begin{cases} \frac{1}{2}\mathcal{T}'_\eta(\sigma_i) - \frac{1}{2}\frac{\mathcal{T}_\eta(\sigma_i)}{\sigma_i} & \text{if } \sigma_i = \sigma_j > 0 \\ 0 & \text{if } \sigma_i = \sigma_j = 0 \\ \frac{\sigma_j\mathcal{T}_\eta(\sigma_i)-\sigma_i\mathcal{T}_\eta(\sigma_j)}{\sigma_i^2-\sigma_j^2} & \text{else} \end{cases}.$$

Authorized licensed use limited to: Stanford University. Downloaded on September 01,2024 at 23:40:18 UTC from IEEE Xplore. Restrictions apply.

### C. Numerically Stable Computation

Constructing $\Delta^{ij}$ in order to evaluate $\sum_{i=1}^{m}\sum_{j=1}^{n}\Gamma_{ij}\Delta^{ij}$ is numerically unstable in high dimensions.

However, some simple algebra gives that

$$\begin{aligned}\sum_{i=1}^{m}\sum_{j=1}^{n}\Gamma_{ij}\Delta^{ij} &= UU^T\left(\sum_{i=1}^{m}\sum_{j=1}^{n}\Gamma_{ij}\Delta^{ij}\right)VV^T \\ &= U\left(\sum_{i=1}^{m}\sum_{j=1}^{n}\Gamma_{ij}U^T\Delta^{ij}V\right)V^T \\ &= U\left(\sum_{i=1}^{m}\sum_{j=1}^{n}\Gamma_{ij}E^{ij}\right)V^T = U\Gamma V^T.\end{aligned}$$

Experimentally, evaluating $U\Gamma V^T$ is numerically stable.

## ACKNOWLEDGMENT

The authors thank Mert Pilanci for many helpful comments during a talk about this article. The authors thank Raphael Meyer for help with Hutch++. The authors also thank an anonymous reviewer for an unusually thorough and careful review that helped to improve the article. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

## REFERENCES

[1] C. M. Stein, "Estimation of the mean of a multivariate normal distribution," *Ann. Statist.*, vol. 9, no. 6, pp. 1135–1151, 1981. [Online]. Available: http://www.jstor.org/stable/2240405

[2] H. Zou, T. Hastie, and R. Tibshirani, "On the 'degrees of freedom' of the LASSO," *Ann. Statist.*, vol. 35, no. 5, pp. 2173–2192, 2007.

[3] R. J. Tibshirani and J. Taylor, "Degrees of freedom in lasso problems," *Ann. Statist.*, vol. 40, no. 2, pp. 1198–1232, 2012, doi: 10.1214/12-AOS1003.

[4] E. J. Candès, C. A. Sing-Long, and J. D. Trzasko, "Unbiased risk estimates for singular value thresholding and spectral estimators," *IEEE Trans. Signal Process.*, vol. 61, no. 19, pp. 4643–4657, Oct. 2013.

[5] A. Beck, *First-Order Methods in Optimization*. Philadelphia, PA, USA: SIAM, 2017.

[6] G. Chierchia, E. Chouzenoux, P. L. Combettes, and J.-C. Pesquet. "The proximity operator repository." User's Guide. Accessed: May 25, 2022. [Online]. Available: http://proximity-operator.net/

[7] J. J. Moreau, "Fonctions convexes duales et points proximaux dans un éspace Hilbertien," *Comptes Rendus l'Académie Sci. Paris*, vol. 255, no. 22, pp. 2897–2899, 1962.

[8] N. Parikh and S. Boyd, "Proximal algorithms," *Found. Trends Optim.*, vol. 1, no. 3, pp. 127–239, 2014.

[9] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM J. Imag. Sci.*, vol. 2, no. 1, pp. 183–202, 2009, doi: 10.1137/080716542.

[10] S. Boyd, N. Parikh, and E. Chu, *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*. Boston, MA, USA: Now, 2011.

[11] M. R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," *J. Res. Nat. Bur. Standards*, vol. 49, no. 6, p. 409, 1952.

[12] A. Krylov, "On the numerical solution of equations which in technical questions determine the frequency of small vibrations of material systems," *Izv. Akad. Nauk SSSR*, vol. 7, no. 4, pp. 491–539, 1931.

[13] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep., 1994. [Online]. Available: https://dl.acm.org/doi/book/10.5555/865018

[14] A. Chambolle and T. Pock, "A first-order primal-dual algorithm for convex problems with applications to imaging," *J. Math. Imag. Vision*, vol. 40, no. 1, pp. 120–145, 2011.

[15] Y. Nesterov, "Gradient methods for minimizing composite functions," *Math. Program.*, vol. 140, no. 1, pp. 125–161, Aug. 2013.

[16] J. Nocedal and S. Wright, *Numerical Optimization*. New York, NY, USA: Springer-Verlag, 2006.

[17] N. Simon, J. Friedman, and T. Hastie, "A blockwise descent algorithm for group-penalized multiresponse and multinomial regression," 2013. [Online]. Available: https://arxiv.org/abs/1311.6529

[18] L. Evans and R. Gariepy, *Measure Theory and Fine Properties of Functions*, Revised ed. New York, NY, USA: CRC Press, 2015.

[19] Y. Nesterov, *Lectures on Convex Optimization* (Springer Optimization and Its Applications). Cham, Switzerland: Springer-Verlag, 2018.

[20] M. F. Hutchinson, "A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines," *Commun. Statist. — Simul. Comput.*, vol. 18, no. 3, pp. 1059–1076, 1989, doi: 10.1080/03610918908812806.

[21] R. A. Meyer, C. Musco, C. Musco, and D. Woodruff, "Hutch++: Optimal stochastic trace estimation," in *Proc. 4th Symp. Simplicity Algorithms (SOSA)*, 2021, pp. 142–155.

[22] D. Persson, A. Cortinovis, and D. Kressner, "Improved variants of the Hutch++ algorithm for trace estimation," *SIAM J. Matrix Anal. Appl.*, vol. 43, no. 3, pp. 1162–1185, 2022, doi: 10.1137/21M1447623.

[23] E. N. Epperly, J. A. Tropp, and R. J. Webber, "Xtrace: Making the most of every sample in stochastic trace estimation," 2023, *arXiv:2301.07825*.

[24] B. Amos and J. Z. Kolter, "OptNet: Differentiable optimization as a layer in neural networks," in *Proc. 34th Int. Conf. Mach. Learn.*, Sydney, Australia: PMLR, 2017, vol. 70, pp. 136–145.

[25] A. Agrawal, S. Barratt, S. Boyd, E. Busseti, and W. Moursi, "Differentiating through a cone program," *J. Appl. Numer. Optim.*, vol. 1, no. 2, pp. 107–115, 2019.

[26] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[27] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* (2015). TensorFlow. [Online]. Available: https://www.tensorflow.org/

[28] A. Agrawal et al., "TensorFlow Eager: A multi-stage, Python-embedded DSL for machine learning," in *Proc. 2nd SysML Conf.*, 2019, pp. 178–189.

[29] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter, "Differentiable convex optimization layers," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 9558–9570.

[30] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Philadelphia, PA, USA: SIAM, 2008.

[31] S. Diamond, V. Sitzmann, F. Heide, and G. Wetzstein, "Unrolled optimization with deep priors," 2018. [Online]. Available: https://arxiv.org/abs/1705.08041v2

[32] S. Wang, S. Fidler, and R. Urtasun, "Proximal deep structured models," in *Proc. Adv. Neural Inf. Process. Syst.*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29, Barcelona, Spain: Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/f4be00279ee2e0a53eafdaa94a151e2c-Paper.pdf

[33] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," 2013. [Online]. Available: https://arxiv.org/abs/1308.3432

[34] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, Apr. 11–13, 2011, pp. 315–323. [Online]. Available: https://proceedings.mlr.press/v15/glorot11a.html

[35] A. Griewank, "On automatic differentiation," *Math. Program., Recent Develop. Appl.*, vol. 6, no. 6, pp. 83–107, 1989.

[36] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *J. Roy. Statistical Soc., Ser. B (Statistical Methodology)*, vol. 67, no. 2, pp. 301–320, 2005.

[37] S. Barratt. "torch_cg." GitHub. [Online]. Available: https://github.com/sbarratt/torch_cg

[38] Z. Frangella, J. A. Tropp, and M. Udell, "Randomized Nyström preconditioning," 2021. [Online]. Available: https://arxiv.org/abs/2110.02820

Authorized licensed use limited to: Stanford University. Downloaded on September 01,2024 at 23:40:18 UTC from IEEE Xplore. Restrictions apply.

[39] P. C. Bellec and C.-H. Zhang, "De-biasing convex regularized estimators and interval estimation in linear models," 2021. [Online]. Available: https://arxiv.org/abs/1912.11943
[40] P. C. Bellec and C.-H. Zhang, "Second-order Stein: SURE for SURE and other applications in high-dimensional inference," *Ann. Statist.*, vol. 49, no. 4, pp. 1864–1903, 2021.

**Parth Nobel** received the B.S. degree in electrical engineering and computer science from the UC Berkeley, in 2021. He is working toward the Ph.D. degree in electrical engineering with Stanford University. Since 2022, he has been a Visiting Scholar with the UC Berkeley in electrical engineering and computer science. His research interests include applying convex optimization and randomized numerical linear algebra to statistics, signal processing, and various other application areas.

**Emmanuel Candès** (Fellow, IEEE) received the Ph.D. degree in statistics from Stanford University, in 1998. He is the Barnum-Simons Chair in mathematics and statistics with Stanford University, and a Professor in electrical engineering (by courtesy). His research interests lie at the interface of statistics, information theory, signal processing, and computational mathematics. He has received several awards, including the Alan T. Waterman Award from the NSF, the MacArthur Fellowship, the 2020 Princess of Asturias Award for Technical and Scientific Research, and the 2021 IEEE Jack S. Kilby Signal Processing Medal. He was elected to the National Academy of Sciences and to the American Academy of Arts and Sciences in 2014.

**Stephen Boyd** (Life Fellow, IEEE) received the A.B. degree in mathematics from Harvard University, Cambridge, MA, USA, in 1980, and the Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley, CA, USA, in 1985. He is currently the Samsung Professor in engineering, and a Professor in electrical engineering with Stanford University, Stanford, CA, USA. He is a member of U.S. National Academy of Engineering (NAE), a foreign member of the Chinese Academy of Engineering (CAE), and a foreign member of the National Academy of Engineering of Korea (NAEK). His current research focuses on convex optimization applications in control, signal processing, machine learning, and finance.

Authorized licensed use limited to: Stanford University. Downloaded on September 01,2024 at 23:40:18 UTC from IEEE Xplore. Restrictions apply.