METHODS FOR L^1 REGULARIZED REGRESSION

PATRIK R. GERBER

Abstract. We review two popular state-of-the-art algorithms to solve the LASSO (L^1 regularized regression) problem: the Fast Iterative Shrinkage-Thresholding Algorithm (FISTA) and Coordinate Descent. We validate our implementation on an image deblurring task and regularized regression on synthetic data.

1. Introduction. Suppose that we have a matrix $A \in \mathbb{R}^{n\times d}$ and a vector $b \in \mathbb{R}^n$ where b is generated by the procedure

$$b = Ax^* + \epsilon$$

for an unknown vector $x^* \in \mathbb{R}^d$ and noise vector $\epsilon \in \mathbb{R}^n$. The problem of recovering x^* from the noisy and transformed observation b is a classical example of a linear inverse problem. A ubiquitous number of practical problems of interest can be phrased in this general framework from fields such as astrophysics, signal and image processing, optics or statistical inference. Linear regression - one of the most widely used statistical method - is obtained if one takes A to be the design matrix, b to be the observations and ϵ to be standard Gaussian noise. The classical solution to the problem is to consider the least-squares estimator

$$\hat{x}_{\text{LS}} \in \arg\min_{x\in\mathbb{R}^d} \|Ax - b\|_2^2 .$$

The above can be solved in closed form using what are known as the normal equations. Often regularizing the objective leads to better conditioned problems or more interpretable solutions. The two most common forms of regularization are Tikhonov regularization

$$\hat{x}_{\text{TIK}} \in \arg\min_{x\in\mathbb{R}^d} \|Ax - b\|_2^2 + \lambda \|x\|_2^2$$

and L^1 regularization

$$\hat{x}_{L^1} \in \arg\min_{x\in\mathbb{R}^d} \|Ax - b\|_2^2 + \lambda \|x\|_1 \tag{1.1}$$

where $\lambda \geq 0$ is the regularization parameter. The latter problem is known as the LASSO in the statistics literature and has the desirable property that its solutions $\hat{x}_{L^1}(\lambda)$ are sparse (aiding interpretability of statistical models). While solving for $\hat{x}_{\text{TIK}}$ is as easy as finding $\hat{x}_{\text{LS}}$, computing $\hat{x}_{L^1}$ is a more difficult, non-differentiable optimization problem. There are multiple methods to compute the LASSO [1] which include coordinate descent, least-angle-regression and proximal methods. Proximal methods are subject to ongoing research and have state-of the art performance for approximating $\hat{x}_{L^1}$. In Section 2 we introduce the reader to the basics of proximal methods for minimizing non-differentiable functions. In Section 3 we derive the ISTA algorithm and introduce its accelerated version FISTA. In Section 4 we describe and derive the Coordinate Descent algorithm and finally, in Section 5 we perform experiments validating our implementation of ISTA, FISTA and Coordinate Descent. The appendix includes a brief overview of the necessary signal processing knowledge required to fully understand Section 5.1.

2. Proximal Gradient Descent. In this section we follow [4] to give the reader an introduction to proximal methods for optimizing non-differentiable functions.

2.1. Gradient descent. Suppose that we wish to minimize a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$. The gradient descent scheme — which dates back to Cauchy [5] — is the simplest strategy to do so. It is defined by the iterative update rule

$$x_{k+1} = x_k - \eta_k \nabla f(x_k). \tag{2.1}$$

Choosing the stepsize $\eta_k > 0$ is a delicate task. It depends on factors like assumptions placed on f such as (strong) convexity, Lipschitz continuity or smoothness and on how long we want to run the algorithm for. The obvious interpretation of (2.1) is to first (a) find direction of fastest descrease; (b) take step of size $\eta_k \|\nabla f\|$ in that direction.

Another form of (2.1) is obtained when noting that it can be rewritten as

$$x_{k+1} \in \arg\min_{x\in\mathbb{R}^d} \left\{ \nabla f(x_k)^\intercal x + \frac{1}{2\eta_k} \|x - x_k\|^2 \right\}, \tag{2.2}$$

which is easily verified using calculus. The following Lemma is at the heart of proximal gradient methods.

LEMMA 2.1. *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a continuously differentiable function. Then f's gradient is $L(f)$-Lipschitz continuous if and only if*

$$f(x) \leq f(y) + \nabla f(y)^\intercal (x - y) + \frac{L}{2} \|x - y\|^2 \tag{2.3}$$

for all $x, y \in \mathbb{R}^d$ and $L \geq L(f)$.

Remark 2.2. Functions f satisfying (2.3) are referred to as $L(f)$-smooth in the optimization literature.

Notice that now the gradient descent update (2.2) is readily interpreted as minimizing the upper bound (2.3) of f with $y = x_k$ and constant stepsize $\eta_k = 1/L$.

Remark 2.3. What happens if f is not differentiable? Naive gradient descent may fail to converge, or can converge at a suboptimal rate. Indeed, consider the simple example of $f(x) = |x|$ for $x \in \mathbb{R}$ and suppose we want to minimize f with constant learning rate $\eta_k \equiv \eta$. Then

$$|x_k| \geq \alpha \wedge (\eta - \alpha)$$

for all $k \in \mathbb{N}$ where $\alpha := \left| x_0 - \eta \lfloor \frac{x_0}{\eta} \rfloor \right|$. This example shows the lack of robustness of naive gradient descent in the absence of differentiability. Subgradient methods provide a solution to this problem, however their convergence rate can be as slow as $1/\sqrt{k}$ in the case where f is Lipschitz continuous, non-strongly convex and non-smooth [4]. In what follows we will develop methods that allow us to solve problems including LASSO that fall in this category at the much fast rate $1/k^2$.

2.2. The proximal operator. Take $f : \mathbb{R}^d \to \mathbb{R}$ continuously differentiable and L-smooth as before and let $g : \mathbb{R}^d \to \mathbb{R}$ be an arbitrary function, not necessarily differentiable. In this section we describe a method to solve

$$\min_{x\in\mathbb{R}^d} f(x) + g(x). \tag{2.4}$$

Once again, we take the quadratic upper bound

$$f(x) + g(x) \leq f(x_k) + \nabla f(x_k)^\intercal (x - x_k) + \frac{L}{2} \|x - x_k\|^2 + g(x).$$

The above yields the following update rule:

$$x_{k+1} := \texttt{prox}_L(x_k) = \arg\min_{x\in\mathbb{R}^d} \left\{ \nabla f(x_k)^\intercal (x - x_k) + g(x) + \frac{L}{2} \|x - x_k\|^2 \right\} \tag{2.5}$$

where `prox` is called the proximal operator. It is not immediately clear that we've made any progress with (2.5) — we now have to perform a difficult looking minimization for each step of our algorithm. This is where the structure of g comes into play. If g has structure that makes (2.5) easy to solve, we have a working algorithm. A common example of such structure is when g is separable in the sense that

$$g(x) = \sum_{i=1}^{d} g_i(x_i).$$

In this case the update rule (**??**) reduces to d separate 1-dimensional optimization problems. Oftentimes the g_i are convex or allow for a closed form minimizer. It can be shown that proximal gradient descent on $f + g$ has the same convergence rate as gradient descent on f (see [4] and references therein).

3. ISTA and FISTA. In this section we specialize the proximal gradient method to the LASSO problem. To this end set $f(x) = \|Ax - b\|_2^2$ where $x \in \mathbb{R}^d$, $b \in \mathbb{R}^n$ and A is an $n \times d$ matrix and $g(x) = \lambda \|x\|_1$ for a regularization parameter $\lambda \geq 0$. Notice that the Lipschitz-constant of the gradient of f is $2\|A^\intercal A\|_{\text{op}}$. The following Lemma is the reason why proximal gradient descent is a viable method to find the LASSO estimator $\hat{x} := \hat{x}_{L^1}$ given in (1.1).

LEMMA 3.1. *The proximal operator (2.5) for minimizing $\|Ax - b\|_2^2 + \lambda \|x\|_1$ takes the form*

$$\texttt{prox}_L(x) = \mathcal{T}_{\lambda/L}\Big(x - 2A^\intercal(Ax - b)/L\Big)$$

where the shrinkage operator $\mathcal{T}_\alpha$ given by

$$\mathcal{T}_\alpha(x) = (|x| - \alpha)_+ sign(x)$$

is applied coordinatewise.

Proof. By definition of the proximal operator

$$\texttt{prox}_\eta(x_k) \in \underset{x \in \mathbb{R}^d}{\arg\min} \sum_{i=1}^{d} \left\{ \nabla f^{(i)}(x_k) x^{(i)} + \lambda |x^{(i)}| + \frac{L}{2}(x^{(i)} - x_k^{(i)})^2 \right\}$$

where we write $v^{(i)}$ for the i'th coordinate of the vector v. The minimization above decouples into d one-dimensional problems which can be solved in closed form. Each of the summands is (strictly) convex because it is a sum of convex functions. Taking subgradients, first order optimality tells us that the minimizer must satisfy

$$x^{(i)} = \begin{cases} x_k^{(i)} - \nabla f^{(i)}(x_k)/L - \lambda/L & \text{if } x_k^{(i)} - \nabla f^{(i)}(x_k)/L - \lambda/L > 0 \\ x_k^{(i)} - \nabla f^{(i)}(x_k)/L + \lambda/L & \text{if } x_k^{(i)} - \nabla f^{(i)}(x_k)/L + \lambda/L < 0 \\ 0 & \text{otherwise.} \end{cases}$$

The above can succinctly be written as

$$x^{(i)} = \mathcal{T}_{\lambda/L}\Big(x_k^{(i)} - \nabla f^{(i)}(x_k)/L\Big).$$

A simple calculation shows that $\nabla f(x) = 2A^\intercal(Ax - b)$ and the result follows. ∎

Let $L := \|2A^\intercal A\|_{\text{op}}$ denote the Lipschitz-constant of ∇f. Lemma 3.1 leads us to the celebrated and remarkably simple ISTA algorithm:

Algorithm 3.1 *ISTA*

Result: An approximate minimizer of $\|Ax - b\|_2^2 + \lambda \|x\|_1$.
Take $x_0 \in \mathbb{R}^d$.
for k = *1,2, ..., t* **do**
  $x_k = \texttt{prox}_L(x_{k-1})$.
end
return x_t.

Recall that by taking $g(x) \equiv 0$ the problem reduces to least-squares and proximal gradient descent reduces to regular gradient descent. In his seminal work [7] Nesterov introduced an improvement on regular gradient descent, which achieved the minimax optimal $\mathcal{O}(1/k^2)$ rate for smooth and convex optimization. This idea of acceleration was successfully applied to ISTA in [3] resulting in the Fast Iterative Shrinkage Thresholding Algorithm or FISTA that we describe below.

Algorithm 3.2 *FISTA*

Result: An approximate minimizer of $\|Ax-b\|_2^2+\lambda\|x\|_1$.
Take $y_1, x_0 \in \mathbb{R}^d$.
for k = *1,2, ...,* t **do**
  $\mathrm{x}_k = \mathtt{prox}_L(y_k)$.
  $\mathrm{t}_{k+1} = \frac{1+\sqrt{4t_k^2+1}}{2}$.
  $\mathrm{y}_{k+1} = x_k + \frac{t_k-1}{t_{k+1}}(x_k - x_{k-1})$.
end
return x_t.

FISTA and ISTA enjoy the following performance guarantees:

THEOREM 3.2 ([3, Theorems 3.1 and 4.4]). *Write* $F(x) := \|Ax-b\|_2^2+\lambda\|x\|_1$ *and let* $\{x_k\}_{k\geq 0}$ *be generated by Algorithm 3.1 or 4.1. In the previous case*

$$F(x_k) - F(x_0) \leq \frac{L\|x_0 - x_*\|_2^2}{2k}$$

while in the latter case

$$F(x_k) - F(x_0) \leq \frac{2L\|x_0 - x_*\|_2^2}{k^2}.$$

We glossed over one detail in our description of Algorithms 3.1 and 4.1 — we assumed that we have access to $L = 2\|A^\intercal A\|_{\mathrm{op}}$. In many settings in practice A is a very large matrix so that approximating L is very costly. There is a modification of both algorithms called (F)ISTA with backtracking (also called adaptive stepsize), that starts with an initial guess L_0 updating it at each step eventually stabilizing at an upper bound for L just a constant factor away.

4. Coordinate Descent. In this Section we review Coordinate Descent, the main competing algorithm of FISTA that is used in practice today for large scale problems. Suppose that $F = f+g$ where f is convex differentiable and $g(x) = \sum_{i=1}^d g_i(x_i)$ is separable with each term g_i convex. A simple observation can be made: Suppose that a point x minimizes F along each coordinate axis. Then for any other point $y \in \mathbb{R}^d$ we have

$$\begin{aligned} F(y) - F(x) &\geq \nabla f(x)^\intercal (y-x) + \sum_{j=1}^d (g_j(y_j) - g_j(x_j)) \\ &= \sum_{j=1}^d \Big\{ \underbrace{\nabla_j f(x)(y_j - x_j) + g_j(y_j) - g_j(x_j)}_{(*)} \Big\}. \end{aligned}$$

We want to show that each $(*)$ is positive. As we are looking at each coordinate direction separately, to ease notation we assume that F, f, g are functions of one variable only. As x is the minimum, first order optimality gives us

$$0 \in \partial\left(f(x) + g(x)\right) = f'(x) + \partial g(x).$$

The above shows in particular that $-f'(x) \in \partial g(x)$. By definition of subgradients for any y we have

$$g(y) \geq g(x) + (-f'(x))(y-x).$$

Rearranging yields the claim. What we've shown is taht if each coordinate of x is optimal then x itself is optimal. This suggests the following algorithm:

Algorithm 4.1 Coordinate Descent

Result: An approximate minimizer of F
Take $x \in \mathbb{R}^d$.
while $F(x)$ *not converged* **do**
  Pick an index $j \in [d]$.
  Replace x_j by $\arg\min_{x' \in \mathbb{R}} F(x_1, ..., x_{j-1}, x', x_{j+1}, ..., x_d)$.
end
return x.

As is the case for proximal gradient descent, for LASSO this one-dimensional minimization can be solved exactly. For reference, see [6] and the `glmnet` package in `R`. Continuing with notation of previous sections, we want to solve

$$\hat{x} \in \arg\min_{\in \mathbb{R}^d} F(x) = \arg\min_{x \in \mathbb{R}^d} \left\{ \|Ax - b\|^2 + \lambda \|x\|_1 \right\}$$

where $\lambda \geq 0$.

LEMMA 4.1. *For the LASSO the minimization step in Algorithm 4 takes the form*

$$x_j = -\frac{\mathcal{T}_\lambda\Big(2A_j^\intercal(A_{-j}x_{-j} - b)\Big)}{A_j^\intercal A_j}$$

where we write A_{-j} for A with its j'th column removed, A_j for the j'th column and x_{-j} for x without its j'th element.

Proof. We have

$$\|Ax - b\|_2^2 + \lambda \|x\|_1 = \sum_{i=1}^n \left(\sum_{j=1}^d A_{i,j}x_j - b_i \right)^2 + \lambda \sum_{j=1}^d |x_j|.$$

Taking the subdifferential with respect to x_j gives

$$\implies 2\sum_{i=1}^n \left(\sum_{k=1}^d A_{i,k}x_k - b_i \right) A_{i,j} + \lambda \partial |x_j|.$$

First order optimality requires

$$0 \in 2A_j^\intercal(A_{-j}x_{-j} - b) + 2A_j^\intercal A_j x_j + \lambda \partial |x_j|.$$

Looking at the each of the cases $x_j < 0, x_j = 0$ and $x_j > 0$ we get

$$x_j = \begin{cases} 0 & \text{if } 2\left|A_j^\intercal(A_{-j}x_{-j} - b)\right| \leq \lambda \\ -\frac{2A_j^\intercal(A_{-j}x_{-j} - b) - \lambda}{2A_j^\intercal A_j} & \text{if } 2A_j^\intercal(A_{-j}x_{-j} - b) > \lambda \\ -\frac{2A_j^\intercal(A_{-j}x_{-j} - b) + \lambda}{2A_j^\intercal A_j} & \text{if } 2A_j^\intercal(A_{-j}x_{-j} - b) < -\lambda \end{cases}.$$

Once again, this is written succinctly using the thresholding operator:

$$x_j = \frac{\mathcal{T}_{\lambda/2}\Big(A_j^\intercal(b - A_{-j}x_{-j})\Big)}{A_j^\intercal A_j}.$$

□

Coordinate descent for LASSO has similar performance guarantees as FISTA.

5. Experiments. I implemented the following algorithms specialized to LASSO:
1. ISTA with adaptive stepsize (matrix-free)
2. FISTA with adaptive stepsize (matrix-free)
3. Coordinate Descent
4. Naive Gradient Descent.

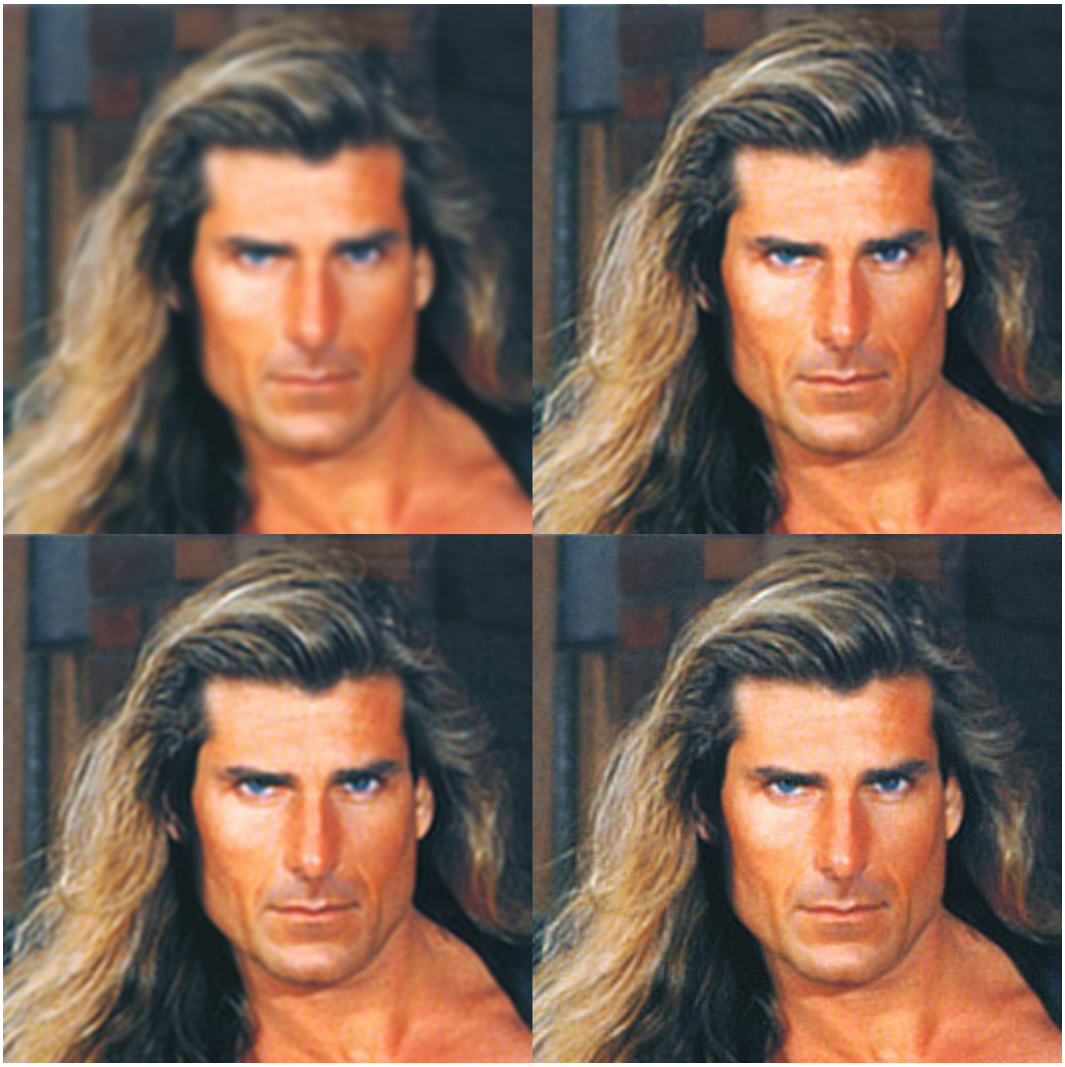

FIGURE 1. *UL: input image. UR: Gradient descent. LL: ISTA, LR: FISTA. (200 iterations each)*

5.1. Image deblurring. I set out to reproduce the results of the seminal paper [3] on image deblurring. The model is as follows. Suppose that we observe a blurry and noisy image $b \in [0,1]^{512\times 512}$ that we assume was generated by the procedure

$$b = \mathcal{R}W^{-1}x_* + \epsilon$$

where $\mathcal{R} : \mathbb{R}^{512\times 512}$ is the Gaussian Blur operator (see the Appendix) with window size k and variance σ^2, W is the Discrete Wavelet Transform with respect to a known wavelet basis and $\epsilon \in \mathbb{R}^{512\times 512}$ is isotropic Gaussian noise with variance τ^2. Our goal is to recover x_*, the DWT of the true/deblurred and denoised image. One possible avenue to recover x_* is to consider the regularized least-squares objective

$$\min_{x\in\mathbb{R}^{512\times 512}} \left\{ \left\|\mathcal{R}W^{-1}x - b\right\|_2^2 + \lambda \left\|x\right\|_1 \right\}.$$

We tested our implementations of ISTA, FISTA and Gradient Descent. We didn't apply Coordinate Descent as our implementation is not matrix-free and thus would require direct access to the DWT, inverse DWT and blurring matrices.

5.1.1. Implementation details. To run (F)ISTA we have to be able to apply the linear maps

$$\mathcal{R}W^{-1} \qquad \text{and} \qquad \left(\mathcal{R}W^{-1}\right)^{\intercal}.$$

By our work in Section 5.2 we know that

$$\left(\mathcal{R}W^{-1}\right)^{\intercal} = W\mathcal{R}.$$

For implementations of $\mathcal{R}$ and W, W^{-1} we use standard packages: `ImageFiltering.jl` for the former and `Wavelets.jl` for the latter. A further point to note is that the testimages are RGB images i.e. elements of the space $[256]^{3\times 512\times 512}$. We separate the red, green and blue channels of the image, normalize them and run the algorithm on each channel independently. Finally, we combine the denoised channels again to get back an RGB image. Our parameters are the same as those used by [3] and are as follows: $k = 4, \sigma^2 = 3, \lambda = 10^{-5}, \tau^2 = 10^{-3}, L_0 = 1, \eta = 1.1$ (η is a parameter of the backtracking step) and the Daubechies Discrete Wavelet Transform was used with periodic boundary conditions.

FIGURE 2. *UL: input image. UR: Gradient descent. LL: ISTA, LR: FISTA. (200 iterations each)*

5.1.2. Results. We performed two sets of experiments. For the first we ran all three algorithms on the same blurry image, started from the point $x_0 = Wb$ i.e. the DWT of the input image. The results can be seen in Figures 1 and 2. On the left of Figure 3 we see the evolution of

$$F(x) := \left\|\mathcal{R}W^{-1}x - b\right\|_2^2 + \lambda \left\|x\right\|_1$$

on log-scale, for 200 iterations. We see from both the pictures and the diagram that gradient descent and ISTA perform comparably with FISTA having a slight edge visually and a more

substantial gap numerically. In the second set of experiments we started from $x_0 = 0$ thereby making the problem harder. This time ISTA and FISTA performed similarly as before while gradient descent failed to get out of the 'valley' of pictures that are mostly black (see Figure 4).

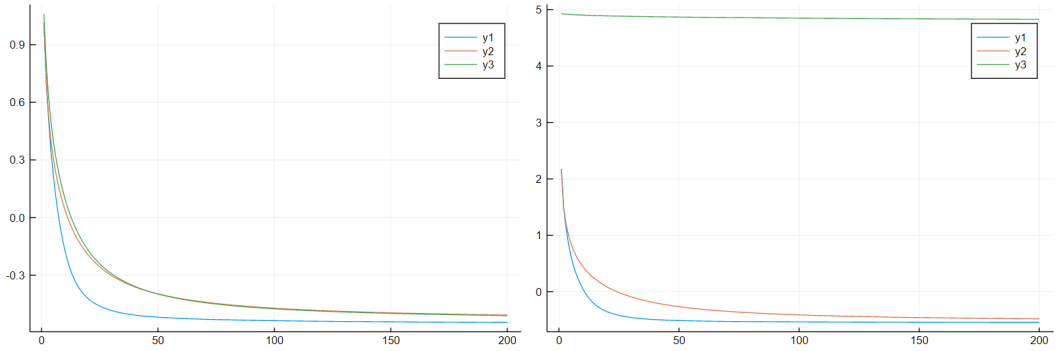

FIGURE 3. *y-axis* $= \log_{10} F$, *x-axis* $=$ *iterations,* $[y1, y2, y3] = [FISTA, ISTA, Gradient\ Descent]$. *LHS:* $x_0 = Wb$, *RHS:* $x_0 = 0$.

FIGURE 4. *Output of* 200 *gradient descent steps started from the all black image* $x_0 = 0$.

5.2. Regression on synthetic dataset. First we draw a matrix $U \in \mathbb{R}^{d\times d}$ with i.i.d. standard Gaussian entries. Then we populate a matrix $A \in \mathbb{R}^{n\times d}$ whose i'th row is given by Ur_i where $r_i \in \mathbb{R}^{d\times 1}$ is a standard Gaussian vector. This way, given U, the rows of A are independent Gaussian with covariance matrix $UU^\intercal$. Finally, we draw a vector $x_* \in \mathbb{R}^d$ at random. We create the synthetic data

$$b = Ax_* + \epsilon$$

where ϵ is isotropic Gaussian noise with variance 0.01. Then, we solve

$$\hat{x} \in \underset{x\in\mathbb{R}^d}{\arg\min} \|Ax - b\|_2^2 + \lambda \|x\|_1 .$$

With the estimate $\hat{x}$ in hand we evaluate its performance by looking at the value of

$$F(\hat{x}) = \|A\hat{x} - b\|_2^2 + \lambda \|\hat{x}\|_1 .$$

We compare our results to those achieved by coordinate descent, ISTA and FISTA in Figure 5. We see that coordinate descent seems to perform well expecially in the high-dimensional setting ($d > n$) and is comparable to FISTA in most settings.

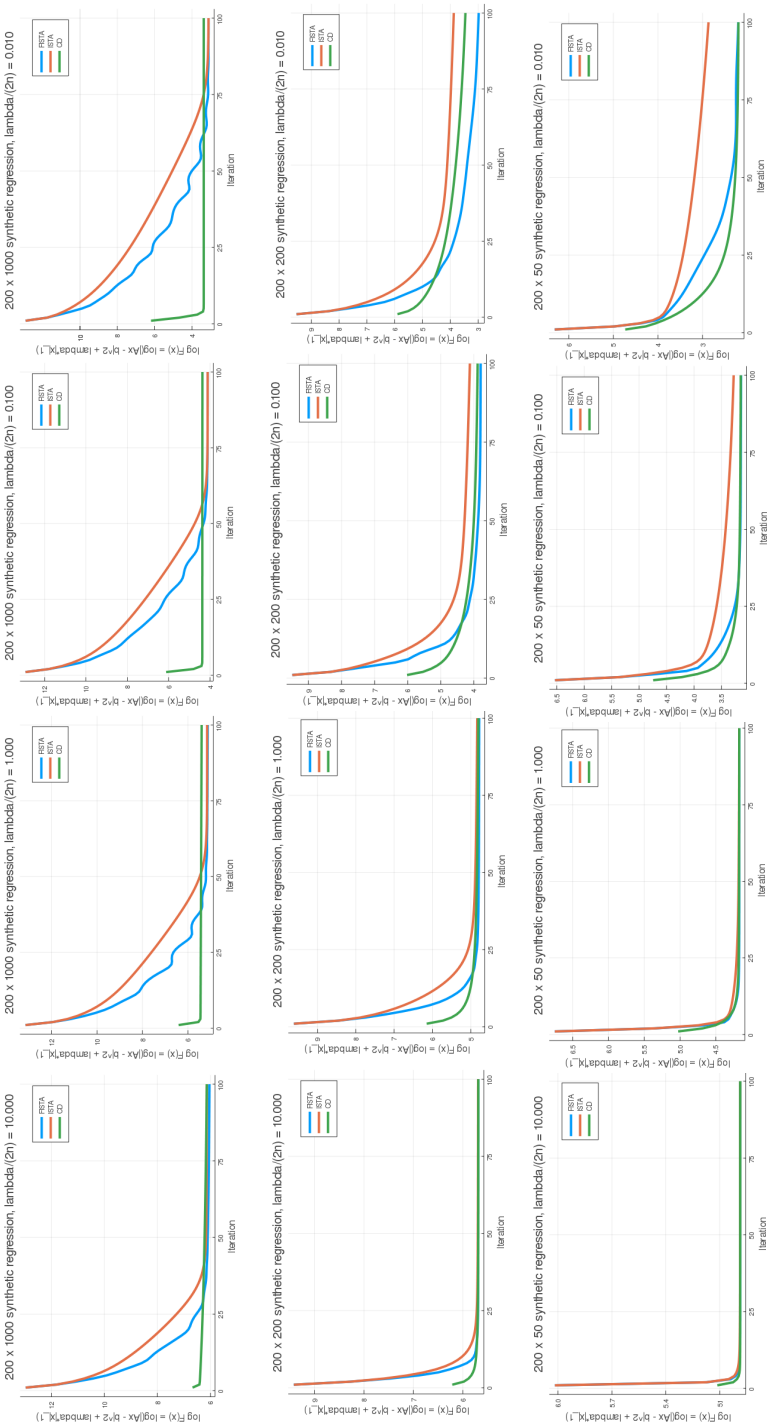

FIGURE 5. *Results of* 12 *experiments on synthetic dataset of varying size.*

REFERENCES

[1] *Computing lasso estimates — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/wiki/Lasso_(statistics)#Computing_lasso_solutions.

[2] *Optimization-based data analysis*. https://cims.nyu.edu/~cfgranda/pages/OBDA_fall17. Accessed: 2020.

[3] A. BECK AND M. TEBOULLE, *A fast iterative shrinkage-thresholding algorithm for linear inverse problems*,

SIAM journal on imaging sciences, 2 (2009), pp. 183–202.
[4] S. Bubeck et al., *Convex optimization: Algorithms and complexity*, Foundations and Trends® in Machine Learning, 8 (2015), pp. 231–357.
[5] A. Cauchy, *Méthode générale pour la résolution des systemes d'équations simultanées*, Comp. Rend. Sci. Paris, 25 (1847), pp. 536–538.
[6] J. Friedman, T. Hastie, and R. Tibshirani, *Regularization paths for generalized linear models via coordinate descent*, Journal of statistical software, 33 (2010), p. 1.
[7] Y. E. Nesterov, *A method of solving a convex programming problem with convergence rate* $o(1/k^2)$, $Dokl.Akad.NaukSSSR$, (1983).

Appendix.

Signal processing. Here I give a brief introduction to the Discrete Wavelet Transform and Gaussian Blurring partly relying on the lecture notes [2]. We do so in order for the reader to understand our experiments in Section 5.

Fourier Series and the Discrete Fourier Transform. Let us recall the theory of Fourier Series and the Discrete Fourier Transform for motivation. It is well known that a function $f \in L^2[0,1]$ can be uniquely represented by its Fourier Series

$$f(\cdot) = \sum_{n\in\mathbb{Z}} \langle f, e^{in\cdot}\rangle_{L^2} e^{in\cdot} \tag{5.1}$$

where equality holds in the L^2 sense. This is due to the fact that $\{e^{in\cdot}\}_{n\in\mathbb{Z}}$ forms an orthonormal basis of $L^2[0,1]$. The values

$$F[n] := \langle f, e^{in\cdot}\rangle_{L^2} = \int_0^1 f(x)e^{inx}\,\mathrm{d}x$$

are called f's Fourier coefficients. Thus we may regard 'taking the Fourier Series' as a linear operation $\Psi_{\mathrm{F}} : L^2[0,1] \to \mathbb{C}^{\mathbb{Z}}$ that maps f to its Fourier coefficients $\{F[n]\}_{n\in\mathbb{Z}}$. The inverse transformation Ψ_{F}^{-1} is given implicitly by (5.1).

Suppose that instead of a function $f \in L^2[0,1]$ we have a discrete signal $\{f[k]\}_{0\le k\le n-1}$ of length n. Analogously to (5.1) we can define the Discrete Fourier Transform

$$F[k] = \langle f, h_k^n\rangle_{\mathbb{C}^n} = \sum_{j=0}^{n-1} f[j]h_k^n[i]$$

where the discrete complex exponentials h_k^n are given by

$$h_k^n[j] := \exp\left(\frac{2\pi ijk}{n}\right), \qquad 0 \le j \le n-1.$$

One can check that $\{\frac{1}{\sqrt{n}}h_0^n, ..., \frac{1}{\sqrt{n}}h_{n-1}^n\}$ is an orthonormal basis of $\mathbb{C}^n$. Once again, we can view 'taking the Discrete Fourier Transform' as a linear map $\Psi_{\mathrm{DF}} : \mathbb{C}^n \to \mathbb{C}^n$. Since dimensions are finite we can express Ψ_{DF} as a matrix $W \in \mathbb{C}^{n\times n}$ given by

$$W_{\mathrm{F}} = \begin{pmatrix} h_0^n[0] & h_0^n[1] & \dots & h_0^n[n-1] \\ \vdots & \vdots & & \vdots \\ h_{n-1}^n[0] & h_{n-1}^n[1] & \dots & h_{n-1}^n[n-1] \end{pmatrix}.$$

By our previous remark $\frac{1}{\sqrt{n}}W_{\mathrm{F}}$ is orthogonal so that the Discrete Fourier Transform's inverse has matrix $W_{\mathrm{F}}^{-1} = \frac{1}{n}W_{\mathrm{F}}^*$. Note that W_{F} might as well be defined as having the normalized $h_k^n/\sqrt{n}$ as its rows, however convention says otherwise (see e.g. Julia's 'fft' function). The similarities between Fourier Series and the Discrete Fourier Transform are easy to see, and the connection is formalized by the Nyquist-Shannon-Kotelnikov Sampling Theorem

THEOREM 5.1 ([2, Theorem 1.13]). *Suppose that $f \in L^2[0,1]$ is bandlimited in the sense that $F[k] = 0$ for $|k| > N$. Then f can be fully recovered from $f(0), f(1/n), ..., f((n-1)/n)$ provided that $n \ge 2N+1$.*

Discrete Wavelet Transform. We develop the ideas for 1-dimensional wavelet analysis with the Haar basis on $[0,1]$, noting that for different applications different bases are appropriate (e.g. the Daubechies basis). Define the mother wavelet $\psi \in L^2[0,1]$ as

$$\psi = \mathbb{1}_{[0,1/2)} - \mathbb{1}_{[1/2,1]}$$

and for $n \ge 0$ and $0 \le k < 2^n$ set

$$\psi_{n,k}(\cdot) := 2^{n/2}\psi(2^n \cdot - k)$$

be its scaled and translated version. One can check that along with the father wavelet $\varphi \equiv \mathbb{1}$ the set $\{\varphi, \{\psi_{n,k}\}_{0\le n, 0\le k<2^n}\}$ forms an orthonormal basis for $L^2[0,1]$. Just as for Fourier Series, we obtain the discrete Haar basis by sampling at the points $0, 1/N, ..., (N-1)/N$. For simplicity one usually takes $N = 2^{K+1}$ so that

$$N = \#\left\{\varphi/\sqrt{N}, \{2^{-n}\psi_{n,k}\}_{0\le n\le K, 0\le k<2^n}\right\}$$

where we rescaled so that the vectors have unit norm. Let us write $\mathcal{H}^N$ for this collection of vectors in $\mathbb{R}^N$ of size N and $\varphi^N, \psi^N_{n,k}$ for the corresponding elements. Just as for the Discrete Fourier Transform, the set $\mathcal{H}^N$ forms an orthonormal basis for $\mathbb{R}^N$. For a signal $\{f[i]\}_{0\le i<N}$ we define the Discrete Haar Transform of f as the collection of numbers

$$H^N_{n,k} = \langle f, \psi^N_{n,k}\rangle_{\mathbb{R}^N}, \qquad 0 \le n \le K,\ 0 \le k < 2^n \tag{5.2}$$

along with the number $H^N_\varphi = \langle f, \varphi^H\rangle_{\mathbb{R}^N}$. Once again, this transformation is represented by the DWT matrix $W_{\mathrm{H}} \in \mathbb{R}^{N\times N}$ given by

$$W_{\mathrm{H}} = \begin{pmatrix} \varphi^N \\ \psi^N_{0,0} \\ \psi^N_{1,0} \\ \psi^N_{1,1} \\ \vdots \\ \psi^N_{K,2^K-1} \end{pmatrix}. \tag{5.3}$$

Note that since $\mathcal{H}^N$ is orthonormal, the matrix W_{H} is orthogonal. In particular $W_{\mathrm{H}}^{-1} = W_{\mathrm{H}}^\intercal$. For illustration we include the DWT matrix for $N = 4$ below.

$$W^{(4)}_{\mathrm{H}} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ -0.5 & -0.5 & 0.5 & 0.5 \\ -0.707107 & 0.707107 & 0 & 0 \\ 0 & 0 & -0.707107 & 0.707107 \end{pmatrix}. \tag{5.4}$$

The reason for the great popularity of wavelets is that as opposed to say the discrete Fourier transform, that besides frequency information the wavelet transform also captures location information. For example, the Fourier Series representation of the function $\mathbb{1}_{[0,1/2]}$ depends heavily on cancellation and suffers from the Gibbs phenomenon, while in the Haar basis its representation is trivial. By varying the mother and father wavelet and imposing various boundary conditions one obtains methods suited better for some applications than others. Wavelet bases have natural extensions to higher dimensions, which is crucial to us as we will be applying the DWT to images, which are represented by 2-d arrays.

Gaussian Blur. Gaussian blur is a basic technique to blur or 'smooth' images. It is a linear transformation $\mathcal{R}$ operating on images (matrices in $\mathbb{R}^{n\times m}$) that replaces each pixel's value by a weighted average of nearby pixels. The weight function is the heat kernel

$$\kappa(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{\|x-y\|^2}{2\sigma^2}} \tag{5.5}$$

where x, y are the positions of two pixels and σ^2 is the variance of the blur. If σ^2 is large the result is very blurry, while as $\sigma^2 \to 0$ the linear map $\mathcal{R}$ converges to the identity. As the kernel (5.5) decays very quickly, it is not necessary to take the average over all pixels. In practice the averaging is done over a $(2k+1)\times(2k+1)$ window centered at the given pixel, where the window size k is to be chosen. For our experiments it is worth noting that $\mathcal{R}$ is symmetric in that $\mathcal{R} = \mathcal{R}^\intercal$.