A Julia package for projection and proximal operators

Tao Hu

Contents

1 Introduction

Modern optimization problems often involve non-differentiable functions, constrains and are in large scale. Classical methods (such as Newton's method, Coordinate descent) may not be able to handle such problem. *Proximal algorithm* is a class of optimization algorithms which turn out to be extremely suitable for many nonsmooth, constrained, and large-scale problems[1]. The key step in proximal algorithm is to evaluate the *proximal operator* (or *proximal mapping*) of a function. For a closed convex function g, its proximal operator $\text{prox}_g(\boldsymbol{x})$ exists and is unique for all $\boldsymbol{x}$, and it is defined as

$$\text{prox}_g(\boldsymbol{x}) = \text{argmin}_{\boldsymbol{u}} \left(g(\boldsymbol{u}) + \frac{1}{2}\|\boldsymbol{u} - \boldsymbol{x}\|_2^2 \right)$$

So far, there are many successful applications of proximal algorithm in modern statistics and machine learnings: sparse recovery for signals and images[2], singular value thresholding for matrix completion[3], regularized matrix regression[4] and *etc.* As the wide application of proximal algorithm, it's worthwhile to maintain a well-implemented library for proximal operators, since proximal operators are the building blocks of proximal algorithm.

Julia[5] is a newly born programming language for scientific computing. It has syntax similar as high level language such as MATLAB, but provides nearly C language efficiency. Julia also support low level operations such as memory allocation and pointer. Therefore, Julia is really a promising programing language in modern statistics. As Julia is still very young, many packages are under construct. It is interesting to construct a library for proximal operators in Julia.

In this project, I develop a Julia package for proximal operators. In total, 15 proximal operators are implemented. The following parts of this report are organized as: section 2 will talk about all proximal operators which are implemented and the algorithms, section 3 is for the overall organization of this package, section 4 provides a concrete example for fitting linear regression model with max function penalty to illustrate the usage of this package, section 5 is about the conclusions of this project and section 6 lists several potential directions for future work.

2 Methods

2.1 Projection operators

In optimization theory, an *indicator function* is defined on a set $\mathcal{C}$ as

$$\chi_{\mathcal{C}}(\boldsymbol{x}) = \begin{cases} 0 & \boldsymbol{x} \in \mathcal{C} \\ +\infty & \boldsymbol{x} \notin \mathcal{C} \end{cases}$$

The proximal mapping of an indicator function is a *projection operator*. Let $g(\boldsymbol{x}) = \chi_{\mathcal{C}}(\boldsymbol{x})$, we will notate the projection operator as

$$\text{prox}_g(\boldsymbol{x}) = \text{argmin}_{\boldsymbol{u}} \left(\chi_{\mathcal{C}}(\boldsymbol{u}) + \frac{1}{2}\|\boldsymbol{u} - \boldsymbol{x}\|_2^2 \right) = P_{\mathcal{C}}(\boldsymbol{x})$$

Next list the projection operators that I have implemented. If a problem has no analytical solution, numerical algorithm to solve that problem will be provided.

(1) Box

Let $\mathcal{C} = \{x | l \leq x \leq u\}$ be a *box* or *hyper-rectangle*, then

$$P_{\mathcal{C}}(x) = \begin{cases} l & x \leq l \\ x & l \leq x \leq u \\ u & x \geq u \end{cases}$$

Note that l and u can be $-\infty$ or $+\infty$.

(2) Affine set

Let $\mathcal{C} = \{\boldsymbol{x} \in \mathtt{R}^n | \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}$ be an *affine set*, then

$$P_{\mathcal{C}}(\boldsymbol{x}) = \boldsymbol{x} - \boldsymbol{A}^{+}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$$

(3) Halfspace

Let $\mathcal{C} = \{\boldsymbol{x} \in \mathtt{R}^n | \boldsymbol{a}^T\boldsymbol{x} \leq b\}$ be an *halfspace*, then

$$P_{\mathcal{C}}(\boldsymbol{x}) = \boldsymbol{x} - \frac{(\boldsymbol{a}^T\boldsymbol{x} - b)_+}{\|\boldsymbol{a}\|_2^2}\boldsymbol{a}$$

where $(u)_+ = \max\{u, 0\}$.

(4) Simplex

Let $\mathcal{C} = \{z | z \geq 0, \mathbf{1}^T z = d\}$ be the *simplex*. The corresponding projection operator has no analytical solution. Here, I used the algorithm given by Duchi[6] to solve this projection problem. The algorithm is showed in Algorithm 1.

Data: A vector $\boldsymbol{x} \in \mathtt{R}^n$ and scalar $d > 0$
Result: Vector $\boldsymbol{y}$ which is the projection of $\boldsymbol{x}$ on $\mathcal{C}$
1 Sort $\boldsymbol{x}$ into $\boldsymbol{\mu}$: $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_n$
2 Find $\rho = \max\left\{j \in [n] : \mu_j - \frac{1}{j}\left(\sum_{r=1}^{j}\mu_r - d\right) > 0\right\}$
3 Define $\theta = \frac{1}{\rho}\left(\sum_{i=1}^{\rho}\mu_i - d\right)$
4 Calculate output $\boldsymbol{y}$ as $y_i = \max\{x_i - \theta, 0\}$

Algorithm 1: Algorithm for projection onto the simplex

(5) Second order cone (SOC)

Let $\mathcal{C} = \{(\boldsymbol{x}, t) \in \mathtt{R}^{n+1} | \|\boldsymbol{x}\|_2 \leq t\}$ be a *second order cone*, then

$$P_{\mathcal{C}}(x, s) = \begin{cases} (\mathbf{0}, 0) & \|\boldsymbol{x}\|_2 \leq -s \\ (\boldsymbol{x}, s) & \|\boldsymbol{x}\|_2 \leq s \\ \frac{1}{2}(1 + \frac{s}{\|\boldsymbol{x}\|_2})(\boldsymbol{x}, \|\boldsymbol{x}\|_2) & \|\boldsymbol{x}\|_2 \geq |s| \end{cases}$$

(6) Positive semidefinite cone

Let $\mathcal{C} = \mathbb{S}_+^n$ be an *positive semidefinite cone*, then

$$P_{\mathcal{C}}(\boldsymbol{X}) = \sum_{i=1}^{n} (\lambda_i)_+ \boldsymbol{u}_i \boldsymbol{u}_i^T$$

where $\sum_{i=1}^{n} (\lambda_i)_+ \boldsymbol{u}_i \boldsymbol{u}_i^T$ is the eigen-decomposition of $\boldsymbol{X}$.

2.2 Proximal operators

List the proximal operators that I have implemented. If a problem has no analytical solution, numerical algorithm to solve that problem will be provided.

(1) Constant function

Let $g(\boldsymbol{x}) = c$, then

$$\text{prox}_g(\boldsymbol{x}) = \boldsymbol{x}$$

(2) Quadratic functions

Let $g(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^T \boldsymbol{x} + c$ with $\boldsymbol{A} \in \mathbb{S}_+^n$, then

$$\text{prox}_{\lambda g}(\boldsymbol{x}) = (\boldsymbol{I} + \lambda \boldsymbol{A})^{-1}(\boldsymbol{x} - \lambda \boldsymbol{b})$$

To solve this problem is essentially equal to solve the linear equation system of

$$(\boldsymbol{I} + \lambda \boldsymbol{A})\boldsymbol{y} = \boldsymbol{x} - \lambda \boldsymbol{b}$$

When n is in small size, the numerical linear algebra method can solve it directly, such as QR decomposition. When n is in large size or $\boldsymbol{A}$ is structured, iterative method will be used. In this implementation, *conjugate gradient* (CG) method was used. The CG algorithm is given in Algorithm 2.

Data: $\boldsymbol{A}$, $\boldsymbol{b}$, and initial value $\boldsymbol{x}^{(0)}$
Initialize: $\boldsymbol{r}^{(0)} \leftarrow \boldsymbol{A}\boldsymbol{x}^{(0)} - \boldsymbol{b}$, $\boldsymbol{p}^{(0)} \leftarrow -\boldsymbol{r}^{(0)}$, $t = 0$
while $\boldsymbol{r}^{(t)} \neq \boldsymbol{0}$ **do**
  $\alpha^{(t)} \leftarrow \frac{\boldsymbol{r}^{(t)T}\boldsymbol{r}^{(t)}}{\boldsymbol{p}^{(t)T}\boldsymbol{A}\boldsymbol{p}^{(t)}}$
  $\boldsymbol{x}^{(t+1)} \leftarrow \boldsymbol{x}^{(t)} + \alpha^{(t)}\boldsymbol{p}^{(t)}$
  $\boldsymbol{r}^{(t+1)} \leftarrow \boldsymbol{r}^{(t)} + \alpha^{(t)}\boldsymbol{A}\boldsymbol{p}^{(t)}$
  $\beta^{(t+1)} \leftarrow \frac{\boldsymbol{r}^{(t+1)T}\boldsymbol{r}^{(t+1)}}{\boldsymbol{r}^{(t)T}\boldsymbol{r}^{(t)}}$
  $\boldsymbol{p}^{(t+1)} \leftarrow -\boldsymbol{r}^{(t+1)} + \beta^{(t+1)}\boldsymbol{p}^{(t)}$
  $t \leftarrow t + 1$
end

Algorithm 2: CG Algorithm for solving $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$

Note that $\boldsymbol{A}, \boldsymbol{b}$ and c can be zeros or other special structures, which will lead to some important cases.

- $\boldsymbol{A} = \boldsymbol{0}$, *i.e.*, g is affine, then $\text{prox}_{\lambda g}(\boldsymbol{x}) = \boldsymbol{x} - \lambda \boldsymbol{b}$.
- if $g(\boldsymbol{x}) = (1/2)\|\boldsymbol{x}\|_2^2$, then

$$\text{prox}_{\lambda g}(\boldsymbol{x}) = \left(\frac{1}{1+\lambda}\right)\boldsymbol{x}$$

(3) Log barrier

Let $g(\boldsymbol{x}) = -\sum_{i=1}^{n} \log x_i$ be the *log barrier* function, then

$$\text{prox}_{\lambda g}(\boldsymbol{x})_i = \frac{x_i + \sqrt{x_i^2 + 4\lambda}}{2}$$

(4) ℓ_1 norm

Let $g(\boldsymbol{x}) = \|\boldsymbol{x}\|_1$ (lasso's penalty term), then

$$\text{prox}_{\lambda g}(\boldsymbol{x})_i = \text{sgn}(x_i)(|x_i| - \lambda)_+$$

This is the so-called *soft thresholding* (*shrinkage*) operator.

(5) ℓ_2 norm

Let $g(\boldsymbol{x}) = \|\boldsymbol{x}\|_2$, then

$$\text{prox}_{\lambda g}(\boldsymbol{x}) = \left(1 - \frac{\lambda}{\|\boldsymbol{x}\|_2}\right)_+ \boldsymbol{x}$$

This is the so-called *block* (*group*) *soft thresholding* operator.

(6) Sum of norms

Let $g(\boldsymbol{x}) = \sum_{g \in G} \|\boldsymbol{x}_g\|_2$ (group lasso's penalty term), then

$$\text{prox}_{\lambda g}(\boldsymbol{x})_g = \left(1 - \frac{\lambda}{\|\boldsymbol{x}_g\|_2}\right)_+ \boldsymbol{x}_g$$

(7) Elastic net

Let $g(\boldsymbol{x}) = \|\boldsymbol{x}\|_1 + (\gamma/2)\|\boldsymbol{x}\|_2^2$ be the *elastic net regularization* term, then

$$\text{prox}_{\lambda g}(\boldsymbol{x}) = \left(\frac{1}{1+\lambda\gamma}\right)\text{prox}_{\lambda\|\cdot\|_1}(\boldsymbol{x})$$

(8) Nuclear norm

Let $g(\boldsymbol{x}) = \|\boldsymbol{X}\|_*$ be the *nuclear norm*, then

$$\text{prox}_{\lambda g}(\boldsymbol{X}) = \boldsymbol{U}\text{diag}((\boldsymbol{\sigma} - \lambda)_+)\boldsymbol{V}^T$$

where $\boldsymbol{X} = \boldsymbol{U}\text{diag}(\boldsymbol{\sigma})\boldsymbol{V}^T$ is the SVD of $\boldsymbol{X}$.

(9) Max function

Let $g(\boldsymbol{x}) = \max\{x_i\}$ be the max function, then

$$\text{prox}_{\lambda g}(\boldsymbol{x}) = \boldsymbol{x}^*$$

where

$$\begin{aligned} x_i^* &= \min\{t^*, x_i\} \\ \sum_{i=1}^{n}(x_i - t^*)_+ &= \lambda \end{aligned}$$

As $\sum_{i=1}^n (x_i - t^*)_+ - \lambda$ is piecewise linear and decreasing function of t^*, so bisection strategy can be used to search for the value of t^* with the initial interval of [minx_i - (1/n), maxx_i]. Summarize the algorithm as in Algorithm 3.

Data: vector $\boldsymbol{x} \in \mathtt{R}^n$, scalar λ and tolerance ϵ
Result: t^*
Initialize: $t_l = \min(\boldsymbol{x}) - (1/n)$, $t_u = \max(\boldsymbol{x})$ and $g(t) = \sum_{i=1}^n (x_i - t)_+ - \lambda$
while $t_u - t_l > \epsilon$ **do**
    $t_0 = \frac{t_l + t_u}{2}$
    if $\text{sgn}(g(t_0)) == \text{sgn}(g(t_l))$ **then**
        $t_l = t_0$
    else
        $t_u = t_0$
    end
end
$t^* = \frac{t_l + t_u}{2}$

Algorithm 3: Bisection Algorithm for searching t^* such that $\sum_{i=1}^n (x_i - t^*)_+ = \lambda$

3 Package usage

3.1 Package organization

This Julia package implement 6 projection operators and 9 proximal operators. List the functions available in this package in Table 1. For each projection/proximal operator, an in-place version is also provided. For example, function *prox_l1* perform the proximal operator for ℓ_1 norm, then *prox_l*1! is the in-place version of *prox_l*1. Thus, there are 30 functions available in this package.

Table 1: Function list

Function name	Description
proj_affine	projection operator onto affine set $\mathcal{C} = \{\boldsymbol{x} \in \mathtt{R}^n \vert \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}$
proj_box	projection operator onto box set $\mathcal{C} = \{x \vert l \leq x \leq u\}$
proj_halfspace	projection operator onto halfspace $\mathcal{C} = \{\boldsymbol{x} \in \mathtt{R}^n \vert \boldsymbol{a}^T\boldsymbol{x} \leq b\}$
proj_psdc	projection operator onto positive semidefinite cone $\mathcal{C} = \mathtt{S}_+^n$
proj_simplex	projection operator onto simplex set $\mathcal{C} = \{z \vert z \geq 0, \mathbf{1}^T z = d\}$
proj_soc	projection operator onto second order cone $\mathcal{C} = \{(\boldsymbol{x}, t) \in \mathtt{R}^{n+1} \vert \|\boldsymbol{x}\|_2 \leq t\}$
prox_affine	proximal operator for affine function $g(\boldsymbol{x}) = \lambda(\boldsymbol{b}^T\boldsymbol{x} + c)$
prox_elasticnet	proximal operator for elastic net penalty term $g(\boldsymbol{x}) = \lambda\left(\|\boldsymbol{x}\|_1 + (\gamma/2)\|\boldsymbol{x}\|_2^2\right)$
prox_l1	proximal operator for ℓ_1 norm $g(\boldsymbol{x}) = \lambda\|\boldsymbol{x}\|_1$
prox_l2	proximal operator for ℓ_2 norm $g(\boldsymbol{x}) = \lambda\sum_{g\in G}\|\boldsymbol{x}_g\|_2$
prox_logbarrier	proximal operator for log barrier $g(\boldsymbol{x}) = -\lambda\sum_{i=1}^n \log x_i$
prox_max	proximal operator for max function $g(\boldsymbol{x}) = \lambda\max\{x_i\}$
prox_nuclear	proximal operator for nuclear norm $g(\boldsymbol{x}) = \lambda\|\boldsymbol{X}\|_*$
prox_quad	proximal operator for quadratic function $g(\boldsymbol{x}) = \lambda\left(\frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x} + c\right)$, $\boldsymbol{A} \in \mathtt{S}_+^n$
prox_sumofsquares	proximal operator for sum of squares $g(\boldsymbol{x}) = (\lambda/2)\|\boldsymbol{x}\|_2^2$

3.2 How to use this package

As this package has not yet been published to the public (since more operators need to be added), you can not install the package as other published packages. However, all source codes have already been provided in the *src/* folder. Therefore, you need to manually *include* the source files in order to this package's functionality. For example, if you want to use *prox_l1*!, then type following command in Julia

```
julia> include("./src/prox_l1!.jl")
```

Note that you should specify the right path of the source file. Then you should be able to use *prox_l1*!.

The usage for each function is quite straight forward: in general, you need to pass input $\boldsymbol{x}$, scalar λ and other necessary data to make the function well-defined. For example, the function *prox_quad* need four parameters: $\boldsymbol{x}, \boldsymbol{A}, \boldsymbol{b}$ and λ. Each function has a brief inline explanation for its usage. A more comprehensive documentation is needed to make this package easier to use.

In the future, this package will be published, so you can install the package using *Pkg.add*.

4 Example

To illustrate the usage of our package and verify its implementation, I tried to solve a linear regression problem with max function penalty. Let $\boldsymbol{X} \in \mathtt{R}^{n\times p}$ be the covariates matrix, $\boldsymbol{\beta} \in \mathtt{R}^p$ be the regression coefficients and $\boldsymbol{y} \in \mathtt{R}^n$ be the response vector, we are interested in solving the optimization problem

$$\min_{\boldsymbol{\beta}} \left\{ \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda \max(\boldsymbol{\beta}) \right\}$$

In this example, set $n = 10000, p = 1000$ and set the random seed as 2015790003. Generate random standard normal as each element of $\boldsymbol{X}$, and let $\boldsymbol{\beta} = (1, 2, 3, 4, 5, 0, \ldots, 0)^T$. Then generate response vector $\boldsymbol{y}$ as $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta}$. Set the grid of tunning parameter λ as $\lambda = 0, 500, 1000, \ldots, 10000$.

FISTA algorithm was employed to solve this problem. The t^{th} iterations is

$$\begin{aligned}
\boldsymbol{\beta}^* &\leftarrow \boldsymbol{\beta}^{(t-1)} + \frac{t-2}{t-1}(\boldsymbol{\beta}^{(t-1)} - \boldsymbol{\beta}^{(t-2)}) \\
\boldsymbol{\beta}^{(t)} &\leftarrow \text{prox}_{sg}\left(\boldsymbol{\beta}^* + s\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}^*)\right)
\end{aligned}$$

where s is the step size and $g(\boldsymbol{\beta}) = \lambda \max(\boldsymbol{\beta})$. Thus, the key step is to perform the proximal operator of max function. Here, I used the corresponding function *prox_max!* in our package to do that proximal operator. Note that the in-place version of the function was used.

On a desktop with CPU i7 @ 3.40 GHz and 16G RAM, it took 1.3 seconds to obtain the whole solution path (figure 4). According to the solution path, we can find that FISTA algorithm did recover the true signal pretty well and very fast. However, the max function penalty term can not really regularize very well since this penalty term only penalize the largest element of $\boldsymbol{\beta}$. Therefore, we observed that only the largest element become smaller as λ increase, while other elements were almost unchanged. So this regression can not do variable selection compared to LASSO.

I also tried to use a convex programming solver (*i.e.*, *Convex.jl* package) to solve this problem directly. However, *Convex.jl* will not give the results even after 20 minutes.

The scripts for this example and corresponding results are included in the HTML file *lrg_max.html*.

5 Conclusion

In this project, a Julia package for proximal operators are constructed. 6 projection operators and 9 proximal operators are implemented. For the proximal operators which have analytical solutions, it's trivial to implement. For the proximal operators which have no analytical solutions, I employed several strategies to solve them: (1) If a problem be reduced to solve a linear equation system, conjugate gradient method were used when in large scale. (2) an algorithm proposed by Duchi were employed to solve the problem of projection onto simplex. (3) for evaluating the proximal operators of a scalar function, bisection strategy was used to search for the numerical solution. If the scale function is not differentiable everywhere but its subgradient exist, then an alternative *localization method*[1] can be used.

All implementations are done with careful considerations for efficiency: (1) Use BLAS routine to do linear algebras. (2) Avoid unnecessary memory allocation by using in-place version functions and the pointers. (3) For every function in this package, a corresponding in-place version is provided. According to the result of the example from section 4, we can see that a good-implemented library could lead to a fast algorithm.

However, this package is far away from perfect. Many other proximal operators need to be added, and more importantly, a uniformed framework should be established for the purpose of modularization. More examples are also necessary. In this project, the example may not make

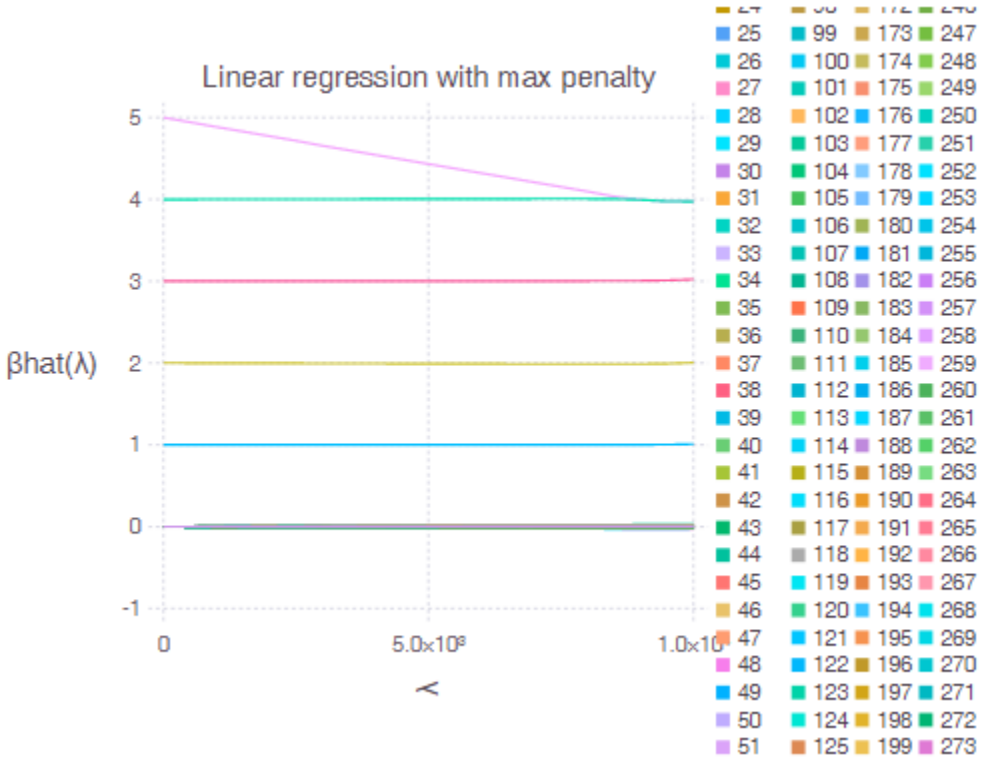

Figure 1: solution path for the example

much sense in terms of the effectiveness of regularization. In one word, there are many further works need to do to complete this package. And in this project, the basic framework has been established.

6 Future work

The possible directions for future work are:

(1) Implement more projection operators and proximal operators, especially for those which have no analytical solutions. We should provide a general framework so that more projection/proximal operators can be obtained.

(2) Give more concrete examples and applications, so that people can get the usage quickly.

(3) Publish the package with a well-explained documentation.

References

[1] Neal Parikh and Stephen Boyd. Proximal Algorithms. *Foundations and Trends in Optimization*, 1(3):123-231, 2014.

[2] A. Bruckstein, D. Donoho, and M. Elad. From sparse solutions of systems of equations to sparse modeling of signals and images. *SIAM Review*, vol. 51, no. 1, pp. 3481, 2009.

[3] J. Cai, E. Cands, and Z. Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization*, vol. 20, no. 4, pp. 19561982, 2010.

[4] H. Zhou and L. Li. Regularized matrix regression. *Journal of Royal Statistical Society Series B*, 76(2):463-483, 2014.

[5] Julia. `http://julialang.org/`

[6] John Duchi, Shai Shalev-Shwartz, Yoram Singer and Tushar Chandra. Efficient Projections onto the ℓ_1-Ball for Learning in High Dimensions. *Proceedings of the* 25^{th} *International Conference on Machine Learning*. ACM New York, NY, 2008.