Scalable First-order Method for Certifying Optimal k-Sparse GLMs

Jiachang Liu
Cornell University
jl4624@cornell.edu

Soroosh Shafiee
Cornell University
shafiee@cornell.edu

Andrea Lodi
Cornell Tech and Technion-IIT
al748@cornell.edu

Abstract

This paper investigates the problem of certifying optimality for sparse generalized linear models (GLMs), where sparsity is enforced through an ℓ_0 cardinality constraint. While branch-and-bound (BnB) frameworks can certify optimality by pruning nodes using dual bounds, existing methods for computing these bounds are either computationally intensive or exhibit slow convergence, limiting their scalability to large-scale problems. To address this challenge, we propose a first-order proximal gradient algorithm designed to solve the perspective relaxation of the problem within a BnB framework. Specifically, we formulate the relaxed problem as a composite optimization problem and demonstrate that the proximal operator of the non-smooth component can be computed exactly in log-linear time complexity, eliminating the need to solve a computationally expensive second-order cone program. Furthermore, we introduce a simple restart strategy that enhances convergence speed while maintaining low per-iteration complexity. Extensive experiments on synthetic and real-world datasets show that our approach significantly accelerates dual bound computations and is highly effective in providing optimality certificates for large-scale problems.

1 Introduction

Sparse generalized linear models (GLMs) are essential tools in machine learning (ML), widely applied in fields like healthcare, finance, engineering, and science. In this paper, we aim to solve

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^p} \quad f(\boldsymbol{X}\boldsymbol{\beta}, \boldsymbol{y}) + \lambda_2\|\boldsymbol{\beta}\|_2^2 \quad \text{s.t.} \quad \|\boldsymbol{\beta}\|_\infty \leq M,\ \|\boldsymbol{\beta}\|_0 \leq k, \tag{1}$$

where $\boldsymbol{X} \in \mathbb{R}^{n\times p}$ and $\boldsymbol{y} \in \mathbb{R}^n$ denote the matrix of features and the vector of labels, respectively, while the parameter $M > 0$ can be either user-defined based on prior knowledge or estimated from the data [14]. The GLM loss function, denoted by $f : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, is assumed to be Lipschitz smooth, the parameter $k \in \mathbb{N}$ controls the number of nonzero coefficients, and $\lambda_2 > 0$ is a small Tikhonov regularization coefficient to address collinearity. Alas, problem (1) is NP-hard [11]. As a result, most existing methods rely on heuristics that deliver high-quality approximations but lack guarantees of optimality. This limitation is particularly problematic in high-stakes applications like healthcare, where ensuring accuracy, reliability, and safety is essential. Therefore, we emphasize the pursuit of certifiably optimal solutions.

A naive approach to solve (1) to optimality is to reformulate it as a mixed-integer programming (MIP) problem and leverage commercial MIP solvers. However, these solvers face significant scalability challenges, particularly with large datasets and nonlinear objectives. A major bottleneck arises from the need to compute tight lower bounds at each node of the branch-and-bound (BnB) tree, a critical component for efficient pruning and solver performance. Existing approaches either generate loose bounds that reduce pruning efficiency or result in high computational costs per iteration. Moreover, they are challenging to parallelize, so they cannot leverage modern hardware accelerators like GPUs.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: MLxOR: Mathematical Foundations and Operational Integration of Machine Learning for Uncertainty-Aware Decision-Making.

To address these challenges, we propose a scalable first-order method to efficiently calculate lower bounds within the Branch and Bound (BnB) framework. By reformulating the problem and deriving its continuous perspective relaxation as an unconstrained convex composite optimization problem, we apply the Fast Iterative Shrinkage-Thresholding Algorithm (FISTA). However, successful FISTA relies on efficient computation of the proximal operator, which requires solving a second order cone program (SOCP) problem. To enable FISTA's efficient use, we introduce a novel customized pooled-adjacent-violation algorithm (PAVA) for exact proximal operator computation with log-linear time complexity, ensuring scalability for large problems. Our approach offers significant computational efficiency by relying on matrix-vector multiplication, which is highly amenable to GPU acceleration, addressing a key limitation of existing methods. To further accelerate FISTA, we implement a restart heuristic, achieving an empirical linear convergence rate — a novel result for this problem type.

Empirically, we demonstrate substantial speedups (1-2 orders of magnitude) in computing dual bounds compared to existing techniques. These improvements significantly enhance the overall efficiency of the BnB process, enabling the certification of large-scale instances previously intractable with commercial MIP solvers.

2 Problem Formulation and Methodology

In order to prune nodes, we first need to calculate a lower bound by solving a convex relaxation of Problem (1). To this end, we employ the perspective function. The perspective function of a quadratic term β_j^2 is β_j^2/z_j. By relaxing the binary variables z_j to be continuous in $[0,1]$, we obtain the following convex relaxation of (1):

$$\begin{aligned} &\min_{\boldsymbol{\beta},\boldsymbol{z}\in\mathbb{R}^p} f(\boldsymbol{X}\boldsymbol{\beta},\boldsymbol{y}) + \lambda_2 \textstyle\sum_{j\in[p]} \beta_j^2/z_j \\ \text{s.t.} \quad &\boldsymbol{z}\in[0,1]^p,\, \mathbf{1}^T\boldsymbol{z}\le k,\; -Mz_j \le \beta_j \le Mz_j \;\forall j\in[p]. \end{aligned} \tag{2}$$

While Problem (2) can be solved using standard conic solvers (e.g., MOSEK, Gurobi), they often rely on interior-point methods that do not scale well with data size. First-order alternatives (e.g., SCS, which is based on ADMM) can be slow to converge and computationally intensive. To overcome these challenges, we begin with reformulating (2) as the following *unconstrained* optimization problem

$$\min_{\boldsymbol{\beta}} f(\boldsymbol{X}\boldsymbol{\beta},\boldsymbol{y}) + 2\lambda_2\, g(\boldsymbol{\beta}), \tag{3}$$

where the implicit function $g:\mathbb{R}^p\to\mathbb{R}\cup\{\infty\}$ is defined as

$$g(\boldsymbol{\beta}) = \left\{ \min_{\boldsymbol{z}\in\mathbb{R}^p} \tfrac{1}{2}\textstyle\sum_{j\in[p]} \beta_j^2/z_j \,\text{s.t.}\boldsymbol{z}\in[0,1]^p,\, \mathbf{1}^\top\boldsymbol{z}\le k, -Mz_j\le\beta_j\le Mz_j\;\forall j\in[p] \right\}. \tag{4}$$

Note that g is convex as convexity is preserved under partial minimization over a convex set [15, Theorem 5,3]. Furthermore, as f is assumed to be Lipschitz smooth and g is non-smooth, Problem (3) is an unconstrained optimization problem with a convex composite objective function. As such, it is amenable to be solved using the FISTA algorithm proposed in [4].

In the following, we show how to efficiently evaluate the proximal operator of g as well as accelerate the FISTA algorithm using a restart strategy.

2.1 Conjugate function g^*

Recall that the conjugate of g is defined as

$$g^*(\boldsymbol{\alpha}) = \sup_{\boldsymbol{\beta}\in\mathbb{R}^p} \boldsymbol{\alpha}^\top\boldsymbol{\beta} - g(\boldsymbol{\beta}).$$

The following lemma gives a closed-form expression for g^*, where $\mathrm{TopSum}_k(\cdot)$ denotes the sum of the top k largest elements, and $H_M:\mathbb{R}\to\mathbb{R}$ is the Huber loss function defined as

$$H_M(\alpha_j) := \begin{cases} \frac{1}{2}\alpha_j^2 & \text{if } |\alpha_j|\le M \\ M|\alpha_j| - \frac{1}{2}M^2 & \text{if } |\alpha_j| > M \end{cases}. \tag{5}$$

We use the shorthand notation $\mathbf{H}_M(\boldsymbol{\alpha})$ to denote $\mathbf{H}_M(\boldsymbol{\alpha}) = (H_M(\alpha_1),\ldots,H_M(\alpha_p))$.

Lemma 2.1. *The conjugate of g is given by*

$$g^*(\boldsymbol{\alpha}) = \mathrm{TopSum}_k(\mathbf{H}_M(\boldsymbol{\alpha})). \tag{6}$$

2.2 Proximal operators of g^* and g

Recall that the proximal operator of g^* with weight parameter $\rho > 0$ is defined as

$$\operatorname{prox}_{\rho g^*}(\boldsymbol{\mu}) = \underset{\boldsymbol{\alpha} \in \mathbb{R}^p}{\operatorname{argmin}} \ \frac{1}{2}\|\boldsymbol{\alpha} - \boldsymbol{\mu}\|_2^2 + \rho g^*(\boldsymbol{\alpha}). \tag{7}$$

The evaluation of $\operatorname{prox}_{\rho g^*}$ involves a minimization problem that can be reformulated as a convex SOCP problem, which is computationally expensive and only returns an approximate solution. Inspired by [6], we design a customized pooled adjacent violators algorithm (PAVA) that provides an exact evaluation of $\operatorname{prox}_{\rho g^*}$ in linear time after performing a simple 1D sorting step (see Algorithm 1 in Appendix A for details).

Theorem 2.2. *For any $\boldsymbol{\mu} \in \mathbb{R}^p$, PAVA (Algorithm 1) evaluates $\operatorname{prox}_{\rho g^*}(\boldsymbol{\mu})$ exactly in $\mathcal{O}(p \log p)$.*

The proof relies on several auxiliary lemmas. We start with the following lemma, which uncovers a close connection between $\operatorname{prox}_{\rho g^*}(\cdot)$ and the generalized isotonic regression problems.

Lemma 2.3. *For any $\boldsymbol{\mu} \in \mathbb{R}^p$, we have*

$$\operatorname{prox}_{\rho g^*}(\boldsymbol{\mu}) = \operatorname{sgn}(\boldsymbol{\mu}) \odot \boldsymbol{\nu}^\star,$$

where $\odot$ denotes the Hadamard (element-wise) product, $\boldsymbol{\nu}^\star$ is the unique solution of the following optimization problem

$$\begin{aligned} \min_{\boldsymbol{\nu} \in \mathbb{R}^p} \quad & \tfrac{1}{2}\textstyle\sum_{j \in [p]} (\nu_j - |\mu_j|)^2 + \rho \sum_{j \in \mathcal{J}} H_M(\nu_j) \\ \text{s.t.} \quad & \nu_j \geq \nu_l \ \text{ if } \ |\mu_j| \geq |\mu_l| \ \ \forall j, l \in [p], \end{aligned} \tag{8}$$

and $\mathcal{J}$ is the set of indices of the top k largest elements of $|\mu_j|, j \in [p]$.

The next lemma shows that the vector generated by PAVA (Algorithm 1) is an exact solution to (8). Intuitively, Algorithm 1 iteratively merges adjacent blocks until no isotonic constraint violations remain, at which point the resulting vector is guaranteed to be the optimal solution to (8).

Lemma 2.4. *The vector $\hat{\boldsymbol{\nu}}$ returned by PAVA (Algorithm 1) solves* (8) *exactly.*

Finally, the merging process in Algorithm 1 can be executed efficiently. Intuitively, each element of $\boldsymbol{\mu}$ is visited at most twice; once during its initial processing and once when it is included in a merged block. This ensures that the process achieves a linear time complexity.

Lemma 2.5. *Aside from sorting, PAVA (Algorithm 1) has linear time complexity $\mathcal{O}(p)$.*

By the extended Moreau decomposition theorem [3, Theorem 6.45], for any weight parameter $\rho > 0$ and any point $\boldsymbol{\mu} \in \mathbb{R}^p$, the proximal operators of g and g^* satisfies

$$\operatorname{prox}_{\rho^{-1} g}(\boldsymbol{\mu}) = \boldsymbol{\mu} - \rho^{-1} \operatorname{prox}_{\rho g^*}(\rho \boldsymbol{\mu}). \tag{9}$$

Hence, together with Theorem 2.2, we can compute exactly $\operatorname{prox}_{\rho^{-1} g}$ using Algorithm 1 with log-linear time complexity. This enables an efficient implementation of the FISTA algorithm.

2.3 FISTA algorithm with restart

The vanilla FISTA algorithm is prone to oscillatory behavior, which results in a sub-linear convergence rate of $\mathcal{O}(1/T^2)$ after T iterations. We further accelerate the empirical convergence performance of the FISTA algorithm by incorporating a simple restart strategy based on the function value, originally proposed in [13].

In simple terms, the restart strategy operates as follows: if the objective function increases during the iterative process, the momentum coefficient is reset to its initial value. The effectiveness of the restart strategy hinges on the efficient computation of the loss function. This task essentially reduces to evaluating the implicit function g defined in (4), which would involve solving a SOCP problem. However, the value of g can be computed efficiently by leveraging the majorization technique [9] (see Algorithm 2 in Appendix A).

Theorem 2.6. *For any $\boldsymbol{\beta} \in \mathbb{R}^p$, majorization (Algorithm 2) computes $g(\boldsymbol{\beta})$ exactly in $\mathcal{O}(p + p \log k)$.*

Theorem 2.6 guarantees that Algorithm 2 can efficiently compute the value of $g(\boldsymbol{\beta})$, which is crucial for our value-based restart strategy to be effective in practice. Empirically, we observe that the function value-based restart strategy can accelerate FISTA from the sub-linear convergence rate of $O(1/T^2)$ to a linear convergence rate in many empirical results. To the best of our knowledge, *this is the first linear convergence result of using a first-order method in the MIP context* when calculating the lower bounds in the BnB tree. The FISTA algorithm is summarized in Algorithm 3 in Appendix A.

3 Experiments and Conclusion

We evaluate on both synthetic and real-world datasets with two key empirical questions: 1) How fast is our FISTA method in calculating the lower bounds compared to existing solvers? 2) How fast is our BnB algorithm compared to existing solvers? For baselines, we compare with the following SOTA commercial and open-source SOCP solvers: Gurobi [8], MOSEK [2], SCS [12], and Clarabel [7].

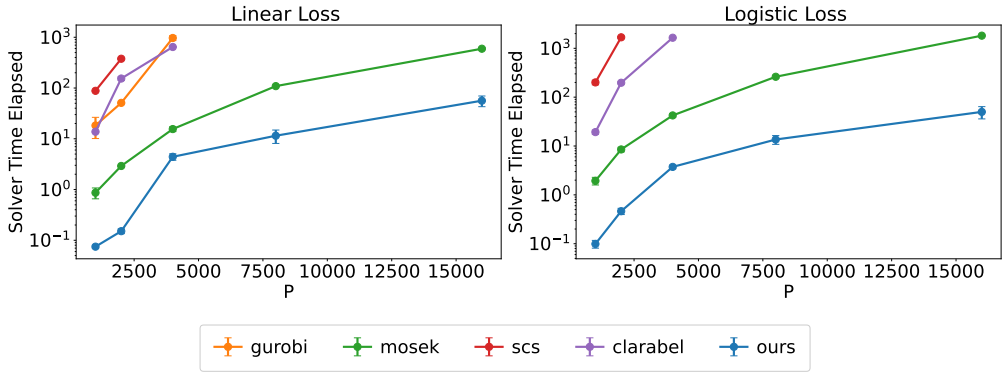

Figure 1: Running time comparison of solving Problem (2). We set $M = 2.0$, $\lambda_2 = 1.0$, and n-to-p ratio to be 1. Gurobi cannot solve the ℓ_0-constrained logistic regression problem.

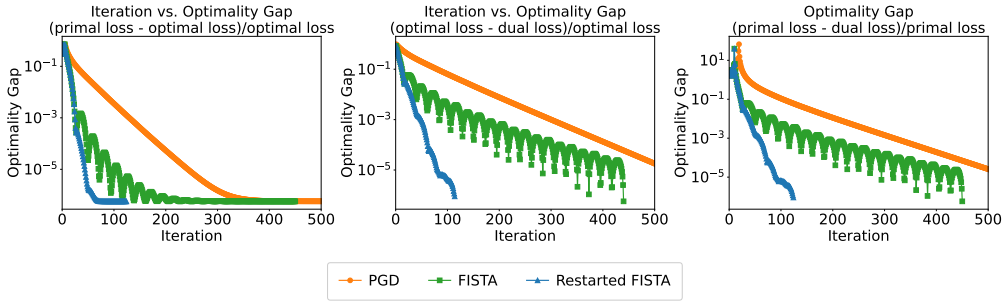

Figure 2: Empirical convergence rate of our restarted FISTA (compared with PGD, the proximal gradient method, and FISTA) on solving the perspective relaxation in Problem (2) with the logistic loss, $n = 16000, p = 16000, k = 10, \rho = 0.5, \lambda_2 = 1.0,$ and $M = 2.0$.

Table 1: Certifying optimality on large-scale and real-world datasets.

		ours		Gurobi		MOSEK	
		time (s)	opt. gap (%)	time (s)	opt. gap (%)	time (s)	opt. gap (%)
Linear Regression	synthetic ($k = 10, M = 2$) (n=16k, p=16k, seed=0)	79	0.0	1800	-	1915	-
	Cancer Drug Response ($k = 5, M = 5$) (n=822, p=2300)	41	0.0	1800	0.89	188	0.0
Logistic Regression	Synthetic ($k = 10, M = 2$) (n=16k, p=16k, seed=0)	626	0.0	N/A	N/A	2446	-
	DOROTHEA ($k = 10, M = 2$) (n=2300, p=89989)	230	0.0	N/A	N/A	1814	0.63

Our experimental results highlight the efficiency and scalability of our proposed method. Figure 1 shows that our FISTA-based approach is over an order of magnitude faster than the best conic solver (MOSEK) for solving the convex relaxation, especially on large-scale instances where most baselines time out. This speedup is driven by our efficient proximal operator evaluation and a restart scheme that achieves a linear convergence rate, as shown in Figure 2. Furthermore, when integrated into a branch-and-bound framework, our method certifies optimality significantly faster than Gurobi and MOSEK, as detailed in Table 1. Our approach solves most instances to optimality in minutes, while commercial solvers struggle to close the gap within the time limit. This demonstrates the practical advantage of our specialized first-order method for this class of problems.

In summary, we introduce a first-order proximal algorithm to solve the perspective relaxation of cardinality-constrained GLM problems. Our key contributions include: 1) a customized PAVA for scalable proximal operator evaluation, 2) a value-based restart strategy that achieves linear convergence, and 3) possible GPU acceleration for further speedup. Extensive empirical results demonstrate that our method outperforms state-of-the-art solvers by 1-2 orders of magnitude, establishing it as a practical, high-performance component for integration into next-generation MIP solvers.

Acknowledgments

This work used the Delta system at the National Center for Supercomputing Applications through allocation CIS250029 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by National Science Foundation grants #2138259, #2138286, #2138307, #2137603, and #2138296.

References

[1] R. K. Ahuja and J. B. Orlin. A fast scaling algorithm for minimizing separable convex functions subject to chain constraints. *Operations Research*, 49(5):784–789, 2001.

[2] M. ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 11.0.4*, 2025.

[3] A. Beck. *First-Order Methods in Optimization*. SIAM, 2017.

[4] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[5] M. J. Best, N. Chakravarti, and V. A. Ubhaya. Minimizing separable convex functions subject to simple chain constraints. *SIAM Journal on Optimization*, 10(3):658–672, 2000.

[6] F. M. Busing. Monotone regression: A simple and fast $O(n)$ PAVA implementation. *Journal of Statistical Software*, 102:1–25, 2022.

[7] P. J. Goulart and Y. Chen. Clarabel: An interior-point solver for conic programs with quadratic objectives, 2024.

[8] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2025.

[9] J. Kim, M. Tawarmalani, and J.-P. P. Richard. Convexification of permutation-invariant sets and an application to sparse principal component analysis. *Mathematics of Operations Research*, 47(4):2547–2584, 2022.

[10] T. Lei, J. Bai, S. Brahma, J. Ainslie, K. Lee, Y. Zhou, N. Du, V. Zhao, Y. Wu, B. Li, et al. Conditional adapters: Parameter-efficient transfer learning with fast inference. *Advances in Neural Information Processing Systems*, 36:8152–8172, 2023.

[11] B. K. Natarajan. Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, 24(2):227–234, 1995.

[12] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. SCS: Splitting conic solver, version 3.2.4. `https://github.com/cvxgrp/scs`, 2023.

[13] B. O'donoghue and E. Candes. Adaptive restart for accelerated gradient schemes. *Foundations of Computational Mathematics*, 15:715–732, 2015.

[14] Y. W. Park and D. Klabjan. Subset selection for multiple linear regression via optimization. *Journal of Global Optimization*, 77(3):543–574, 2020.

[15] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

[16] S. Shafiee and F. Kılınç-Karzan. Constrained optimization of rank-one functions with indicator variables. *Mathematical Programming*, pages 1–47, 2024.

[17] Y. Xie, H. Dai, M. Chen, B. Dai, T. Zhao, H. Zha, W. Wei, and T. Pfister. Differentiable top-k with optimal transport. *Advances in neural information processing systems*, 33:20520–20531, 2020.

A Algorithmic Charts

A.1 Customized PAVA to solve $\text{prox}_{\rho g^*}(\boldsymbol{\mu})$

Algorithm 1 Customized PAVA to solve $\text{prox}_{\rho g^*}(\boldsymbol{\mu})$

Input: vector $\boldsymbol{\mu}$, scalar multiplier ρ, and threshold M of the Huber loss function $H_M(\cdot)$

Initialize $\boldsymbol{\rho} \in \mathbb{R}^n$ with $\rho_j = \rho$ if $j \in \{1, 2, ..., k\}$ and $\rho_j = 0$ otherwise.
▷*Sort $\boldsymbol{\mu}$ such that $|\mu_1| \geq |\mu_2| \geq ... \geq |\mu_p|$; $\boldsymbol{\pi}$ is the sorting order.*
$\boldsymbol{\mu}, \boldsymbol{\pi} = \text{SpecialSort}(\boldsymbol{\mu})$
▷*STEP 1: Initialize a pool of p blocks with start and end indices; each block initially has length equal to 1*
$\mathcal{J} = \{[1,1], [2,2], ..., [p,p]\}$
▷*STEP 2: Initialize $\hat{\nu}_j$ in each block by ignoring the isotonic constraints*
for $j = 1, 2, \ldots, p$ **do**
  $\hat{\nu}_j = \text{argmin}_\nu \frac{1}{2}(\nu - |\mu_j|)^2 + \rho_j H_M(\nu)$
end for
▷*STEP 3: Whenever there is an isotonic constraint violation between two adjacent blocks, merge the two blocks by setting all values to be the minimizer of the objective function restricted to this merged block*
while $\exists [a_1, a_2], [a_2+1, a_3] \in \mathcal{J}$ s.t. $\hat{\nu}_{a_1} < \hat{\nu}_{a_3}$ **do**
  $\mathcal{J} = \mathcal{J} \setminus \{[a_1, a_2]\} \setminus \{[a_2+1, a_3]\} \cup \{[a_1, a_3]\}$
  $\hat{\nu}_{[a_1:a_3]} = \underset{\nu}{\text{argmin}} \sum_{j=a_1}^{a_3} \left[\frac{1}{2}(\nu - |\mu_j|)^2 + \rho_j H_M(\nu)\right]$
end while
▷*Return $\hat{\boldsymbol{\nu}}$ with the inverse sorting order*
Return $\text{sgn}(\boldsymbol{\mu}) \odot \boldsymbol{\pi}^{-1}(\hat{\boldsymbol{\nu}})$

A.2 Algorithm to compute $g(\boldsymbol{\beta})$

Algorithm 2 Algorithm to compute $g(\boldsymbol{\beta})$

Input: vector $\boldsymbol{\beta} \in \mathbb{R}^p$ from Step 2 Line 8 in Algorithm 3.

Initialize: $\boldsymbol{\psi} = \mathbf{0} \in \mathbb{R}^k$
Sort $\boldsymbol{\beta}$ partially such that
$$|\beta_1| \geq |\beta_2| \geq ... \geq |\beta_k| \geq \max_{k+1,\ldots,p} \{|\beta_j|\}$$
$s = \sum_{j=1}^p |\beta_j|$
▷*Compute the majorization vector $\boldsymbol{\psi}$; see Appendix 2.6 for its definition and connection with $\boldsymbol{\beta}$*
for $j = 1, 2, \ldots, k$ **do**
  $\overline{s} = s/(k-j+1)$
  if $\overline{s} \geq |\beta_j|$ **then** $\psi_{j:k} = \overline{s}$; break **else** $\psi_j = |\beta_j|$
  $s = s - |\beta_j|$
end for
return $\frac{1}{2}\sum_{j=1}^k \psi_j^2$

A.3 Main Algorithm to solve Problem (2)

Algorithm 3 Main algorithm to solve problem (2)

Input: number of iterations T, coefficient λ_2 for the ℓ_2 regularization, and step size L (Lipschitz-continuity parameter of $\nabla_{\boldsymbol{\beta}} F(\boldsymbol{X}\boldsymbol{\beta})$)

Initialize: $\boldsymbol{\beta}^0 = \mathbf{0}$, $\boldsymbol{\beta}^1 = \mathbf{0}$, $\phi = 1$
Let: $\rho = L/(2\lambda_2)$, $\mathcal{L}^1 = f(\boldsymbol{\beta}^1, \boldsymbol{y})$
for $t = 1, 2, 3, ..., T$ **do**
  ▷*Step 1: momentum acceleration*
  $\boldsymbol{\gamma}^t = \boldsymbol{\beta}^t + \frac{\phi}{\phi+3}(\boldsymbol{\beta}^t - \boldsymbol{\beta}^{t-1})$
  ▷*Step 2: proximal gradient descent; use Algorithm 1*
  $\boldsymbol{\gamma}^t = \boldsymbol{\gamma}^t - \frac{1}{L}\nabla_{\boldsymbol{\gamma}} F(\boldsymbol{X}\boldsymbol{\gamma}^t)$
  $\boldsymbol{\beta}^{t+1} = \boldsymbol{\gamma}^t - \rho^{-1}\text{prox}_{\rho g^*}(\rho\boldsymbol{\gamma}^t)$
  ▷*Step 3: restart; use Algorithm 2*
  $\mathcal{L}^{t+1} = f(\boldsymbol{X}\boldsymbol{\beta}^{t+1}, \boldsymbol{y}) + 2\lambda_2 g(\boldsymbol{\beta}^{t+1})$
  if $\mathcal{L}^{t+1} \geq \mathcal{L}^t$ **then** $\phi = 1$ **else** $\phi = \phi + 1$
end for
return $\boldsymbol{\beta}^{T+1}$

B Proofs

This section contains all omitted proofs in the paper.

B.1 Proof of Lemma 2.1

Lemma 2.1. *The conjugate of g is given by*

$$g^*(\boldsymbol{\alpha}) = \text{TopSum}_k(\mathbf{H}_M(\boldsymbol{\alpha})).$$

Proof. Using the definition of the implicit function g in (4), we have

$$g^*(\boldsymbol{\alpha}) = \left\{ \begin{array}{ll} \max & \boldsymbol{\alpha}^\top \boldsymbol{\beta} - \frac{1}{2}\sum_{j\in[p]} \beta_j^2/z_j \\ \text{s.t.} & \boldsymbol{\beta} \in \mathbb{R}^p,\, \boldsymbol{z} \in [0,1]^p,\, \mathbf{1}^\top \boldsymbol{z} \leq k, \\ & -Mz_j \leq \beta_j \leq Mz_j \;\forall j \in [p] \end{array} \right. \tag{10}$$

For any fixed feasible $\boldsymbol{z}$, the maximization problem over $\boldsymbol{\beta}$ is a simple constrained quadratic problem, that can be solved analytically by the vector $\beta^\star$ whose j'th element is given by $\beta_j^\star = \text{sgn}(\alpha_j)\min(|\alpha_j|, M)z_j$. Substituting the optimizer $\beta^\star$, the objective function of the maximization problem in (10) simplifies to

$$\begin{aligned} \boldsymbol{\alpha}^\top \boldsymbol{\beta}^\star - \frac{1}{2}\sum_{j\in[p]} \beta_j^{\star 2}/z_j &= \sum_{j\in[p]} \alpha_j \cdot \text{sgn}(\alpha_j)\min(|\alpha_j|, M)z_j - \frac{\left(\text{sgn}\left(\alpha_j\right)\min\left(|\alpha_j|, M\right)z_j\right)^2}{2z_j} \\ &= \sum_{j\in[p]} \left(|\alpha_j|\min(|\alpha_j|, M) - \frac{1}{2}\min(\alpha_j^2, M^2)\right) z_j \\ &= \sum_{j\in[p]} H_M(\alpha_j)z_j, \end{aligned}$$

where the second equality holds as $\boldsymbol{z}$ is a binary vector, and the last equality follows from the definition of the Huber loss function:

$$H_M(x) = \begin{cases} \frac{1}{2}x^2 & \text{if } |x| \leq M \\ M|x| - \frac{1}{2}M^2 & \text{if } |x| > M \end{cases}.$$

We thus arrive at

$$g^*(\boldsymbol{\alpha}) = \max_{\boldsymbol{z}\in[0,1]^p} \left\{ \sum_{j\in[p]} H_M(\alpha_j)z_j : \mathbf{1}^\top \boldsymbol{z} \leq k \right\} = \text{TopSum}_k(\mathbf{H}_M(\boldsymbol{\alpha})).$$

This completes the proof.

□

Note that recent works [17, 10] have discussed the smooth approximation of the $\text{TopSum}_k(\cdot)$. However, such a smooth approximation is not suitable for this work. The effectiveness of the proximal algorithm relies on the exact evaluation of the proximal operator, which we will talk about next. Replacing $\text{TopSum}_k(\cdot)$ would lead to solving a different problem rather than the proximal operator evaluation. This will not help us to use FISTA to solve the perspective relaxation. As a result, this approach would not guarantee valid lower bounds necessary for optimality certification.

B.2 Proof of Lemma 2.3

Lemma 2.3. *For any $\boldsymbol{\mu} \in \mathbb{R}^p$, we have*

$$\text{prox}_{\rho g^*}(\boldsymbol{\mu}) = \text{sgn}(\boldsymbol{\mu}) \odot \boldsymbol{\nu}^\star,$$

where $\odot$ denotes the Hadamard (element-wise) product, $\boldsymbol{\nu}^\star$ is the unique solution of the following optimization problem

$$\begin{array}{ll} \min\limits_{\boldsymbol{\nu} \in \mathbb{R}^p} & \frac{1}{2}\sum_{j\in[p]}(\nu_j - |\mu_j|)^2 + \rho \sum_{j\in\mathcal{J}} H_M(\nu_j) \\ \text{s.t.} & \nu_j \geq \nu_l \;\; \textit{if} \;\; |\mu_j| \geq |\mu_l| \;\; \forall j, l \in [p], \end{array} \tag{11}$$

and $\mathcal{J}$ is the set of indices of the top k largest elements of $|\mu_j|, j \in [p]$.

Proof. For simplicity, let $\boldsymbol{\alpha}^\star = \text{prox}_{\rho g^*}(\boldsymbol{\mu})$, that is,

$$\boldsymbol{\alpha}^\star = \underset{\boldsymbol{\alpha}\in\mathbb{R}^p}{\text{argmin}} \;\; \frac{1}{2}\|\boldsymbol{\alpha} - \boldsymbol{\mu}\|_2^2 + \rho g^*(\boldsymbol{\alpha}). \tag{12}$$

We first show that $\text{sgn}(\boldsymbol{\alpha}^\star) = \text{sgn}(\boldsymbol{\mu})$ (step 1) and then establish that for every $j, l \in [p]$ with $|\mu_j| \geq |\mu_l|$, we have $|\alpha_j^\star| \geq |\alpha_l^\star|$ (step 2). We then conclude the proof using these observations.

- ◇ **Step 1.** We prove the sign-preserving property through a proof by contradiction. For the sake of contradiction, suppose that there exists some $j \in [p]$ such that $\text{sgn}(\alpha_j^\star) \neq \text{sgn}(\mu_j)$. Hence, we can construct a new $\boldsymbol{\alpha}'$ by flipping the sign of $\alpha_j^\star$, i.e., $\alpha_j' = -\alpha_j^\star$, and keeping the rest of the elements the same as $\boldsymbol{\alpha}^\star$. Now under the assumption that $\text{sgn}(\alpha_j^\star) \neq \text{sgn}(\mu_j)$, we have $\left|\alpha_j^\star - \mu_j\right| > \left||\alpha_j^\star| - |\mu_j|\right| = \left|\alpha_j' - \mu_j\right|$, so the j-th term in the first summation of the objective function will decrease while everything else remains the same. This leads to a smaller objective value for $\boldsymbol{\alpha}'$ than $\boldsymbol{\alpha}^\star$, which contradicts the optimality of $\boldsymbol{\alpha}^\star$. Thus, the claim follows.
- ◇ **Step 2.** We prove the relative magnitude-preserving property through a proof by contradiction. For the sake of contradiction, suppose that there exists some $j, l \in [p]$ such that $|\mu_j| \geq |\mu_l|$ but $|\alpha_j^\star| < |\alpha_l^\star|$. Then, we can construct a new $\boldsymbol{\alpha}'$ by swapping $\alpha_j^\star$ and $\alpha_l^\star$, i.e., $\alpha_j' = \alpha_l^\star$ and $\alpha_l' = \alpha_j^\star$, and keeping the rest of the elements the same as $\boldsymbol{\alpha}^\star$. Under the assumption that $|\mu_j| \geq |\mu_l|$ but $|\alpha_j^\star| < |\alpha_l^\star|$, we have $\left|\alpha_j^\star - \mu_j\right| + |\alpha_l^\star - \mu_l| > |\alpha_l^\star - \mu_j| + \left|\alpha_j^\star - \mu_l\right| = \left|\alpha_j' - \mu_j\right| + |\alpha_l' - \mu_l|$, so the sum of the j-th and l-th terms in the first summation of the objective function will decrease while everything else remains the same. This leads to a smaller objective value for $\boldsymbol{\alpha}'$ than $\boldsymbol{\alpha}^\star$, which contradicts the optimality of $\boldsymbol{\alpha}^\star$. Thus, the claim follows.

Using these two observations, we are ready to prove that $\boldsymbol{\alpha}^\star = \text{sgn}(\boldsymbol{\mu}) \odot \boldsymbol{\nu}^\star$. We first reparametrize the minimization problem (12) by substituting the decision variable $\boldsymbol{\alpha}$ with a new variable $\boldsymbol{\nu} \in \mathbb{R}_+^p$ satisfying $\boldsymbol{\alpha} = \text{sgn}(\boldsymbol{\mu}) \odot \boldsymbol{\nu}$. By the sign-preserving property in step 1, it is easy to show the equivalence between the optimization problem in (12) and the following optimization problem

$$\min_{\boldsymbol{\nu}\in\mathbb{R}_+^p} \;\; \frac{1}{2}\sum_{j\in[p]}(\nu_j - |\mu_j|)^2 + \rho\, \text{TopSum}_k\left(\mathbf{H}_M(\boldsymbol{\nu})\right).$$

By the relative magnitude-preserving property in step 2, we can further set the equivalence between the minimization problem in (12) and the following optimization problem

$$\begin{array}{ll} \min\limits_{\boldsymbol{\nu} \in \mathbb{R}^p_+} & \frac{1}{2}\sum_{j\in[p]}(\nu_j - |\mu_j|)^2 + \rho \sum_{j\in\mathcal{J}} H_M(\nu_j), \\ \text{s.t.} & \nu_j \geq \nu_l \;\; \text{if} \;\; |\mu_j| \geq |\mu_l|. \end{array}$$

Lastly, the nonnegative constraint on $\boldsymbol{\nu}$ can be removed as the second summation term in the objective function implies that $\nu_j \geq 0$. Thus, we have shown that any feasible point $\boldsymbol{\alpha}$ in the minimization problem (12) can be reconstructed by any feasible point $\boldsymbol{\nu}$ in the minimization problem in the statement of lemma, while maintaining the same objective value. Hence, we may conclude that $\boldsymbol{\alpha}^\star = \text{sgn}(\boldsymbol{\mu}) \odot \boldsymbol{\nu}^\star$, as required. □

B.3 Proof of Lemma 2.4

Lemma 2.4. *The vector $\hat{\boldsymbol{\nu}}$ returned by PAVA (Algorithm 1) solves* (8) *exactly.*

Proof. The minimization problem (8) is an instance of a generalized isotonic regression problem taking the form

$$\min_{\boldsymbol{\nu}} \sum_{j=1}^{p} h_j(\nu_j) \quad \text{s.t.} \quad \nu_1 \geq \nu_2 \geq \cdots \geq \nu_J, \tag{13}$$

where $h_j(\nu) = \frac{1}{2}(\nu - \mu_j)^2 + \rho_j H_M(\nu)$, $\rho_j = \rho$ if $j \in \mathcal{J}$ and $\rho_j = 0$ otherwise, and the set $\mathcal{J}$ is the set of indices of top k largest elements of $|\mu_j|$, as defined in the statement of Lemma 2.3. Thanks to [5, 1], the optimizer of (13) satisfies two key properties:

- ◇ **Property 1: Optimal solution for a merged block is single-valued.** Suppose we have two adjacent blocks $[a_1, a_2]$ and $[a_2 + 1, a_3]$ such that the optimal solution of each block is single-valued, that is, the minimization problems

$$\begin{cases} \min\limits_{\boldsymbol{\nu}_{a_1:a_2}} & \sum_{j=a_1}^{a_2} h_j(\nu_j) \\ \text{s.t.} & \nu_{a_1} \geq \cdots \geq \nu_{a_2} \end{cases} \quad \text{and} \quad \begin{cases} \min\limits_{\boldsymbol{\nu}_{a_2+1:a_3}} & \sum_{j=a_2+1}^{a_3} h_j(\nu_j) \\ \text{s.t.} & \nu_{a_2+1} \geq \cdots \geq \nu_{a_3} \end{cases}$$

  are solved by $\boldsymbol{\nu}^\star_{a_1:a_2}$ and $\boldsymbol{\nu}^\star_{a_2+1:a_3}$ with $\nu^\star_{a_1} = \cdots = \nu^\star_{a_2}$ and $\nu^\star_{a_2+1} = \cdots = \nu^\star_{a_3}$, respectively. If $\nu^\star_{a_1} \leq \nu^\star_{a_2+1}$, then the optimal solution for the merged block $[a_1, a_3]$ is single-valued, that is, the minimization problem

$$\begin{cases} \min\limits_{\boldsymbol{\nu}_{a_1:a_3}} & \sum_{j=a_1}^{a_3} h_j(\nu_j) \\ \text{s.t.} & \nu_{a_1} \geq \cdots \geq \nu_{a_3} \end{cases}$$

  is solved by $\boldsymbol{\nu}^\star_{a_1:a_3}$ with $\nu^\star_{a_1} = \cdots = \nu^\star_{a_3}$.

- ◇ **Property 2: No isotonic constraint violation between single-valued blocks implies the solution is optimal.** Suppose that we have s blocks $[a_1, a_2], [a_2 + 1, a_3], \ldots, [a_s + 1, a_{s+1}]$ (with $a_1 = 1$ and $a_{s+1} = p$) such that the optimal solution for each block is single-valued, that is, $\nu^\star_{a_l+1} = \cdots = \nu^\star_{a_{l+1}}$ for all $l \in [s]$. Then, if $\hat{\nu}_{a_1} \geq \hat{\nu}_{a_2+1} \geq \ldots \hat{\nu}_{a_s}$, then $\hat{\boldsymbol{\nu}}$ is the optimal solution to (13).

Using these two properties, it is now easy to see why Algorithm 1 returns the optimal solution. We start by constructing blocks which have length 1. The initial value restricted to each block is optimal. Then, we iteratively merge adjacent blocks and update the values of ν_j's whenever there is a violation of the isotonic constraint. By the first property, the optimal solution for the merged block is single-valued. Therefore, we can compute the optimal solution for the merged block by solving a univariate optimization problem. We keep merging blocks until there is no isotonic constraint violation. When this happens, by construction, the solution for each block is single-valued and optimal. By the second property, the final vector $\hat{\boldsymbol{\nu}}$ is the optimal solution to (13), as required. □

B.4 Proof of Lemma 2.5

Lemma 2.5. *Aside from sorting, PAVA (Algorithm 1) has linear time complexity $\mathcal{O}(p)$.*

Proof. A detailed implementation of line 11-14 (Step 3) of the PAVA Algorithm 1 that achieves a linear time complexity is presented in Algorithm 4. In the following, we first show that Algorithm 4 accomplishes the objective in lines 11-14 of Algorithm 1. We then establish that Algorithm 4 runs in linear time complexity.

To prove the first claim, we show that the parameters P_b, S_b, and ν_b amount to

$$P_b = \textstyle\sum_{j \in \mathcal{B}(b)} \rho_j,\ S_b = \sum_{j \in \mathcal{B}(b)} |\mu_j|,\ \nu_b = \operatorname{prox}_{\sum_{j \in \mathcal{B}(b)} \rho_j H_M}(|\mu_j|)$$

for each block index b, where $\mathcal{B}(b)$ denoting the set of indices in the b'th block. It is easy to verify that Algorithm 4 recursively computes P_b and S_b. Thus, we will focus on ν_b. Note that the computation of the proximal operator in ν_b is reduced to solving a univariate optimization problem for each b and satisfies

$$\begin{aligned} \nu_b &= \operatorname*{argmin}_{v \in \mathbb{R}} \sum_{j \in \mathcal{B}(b)} \left(\frac{1}{2}(v - |\mu_j|)^2 + \rho_j H_M(v) \right) \\ &= \operatorname*{argmin}_{v} \sum_{j \in \mathcal{B}(b)} \left(\frac{1}{2} v^2 - v|\mu_j| + \rho_j H_M(v) \right) \\ &= \operatorname*{argmin}_{v} \left(\frac{1}{2} v^2 - \frac{S_b}{N_b}|\mu_j| + \frac{P_b}{N_b} H_M(v) \right) = \operatorname*{argmin}_{v} \left(\frac{1}{2} \left(v - \frac{S_b}{N_b} \right)^2 + \frac{P_b}{N_b} H_M(v) \right) = \operatorname{prox}_{\frac{P_b}{N_b} H_M}\left(\frac{S_b}{N_b}\right). \end{aligned}$$

Algorithm 4 Up and Down Block Algorithm for Merging in PAVA

Input: vector $\boldsymbol{\mu} \in \mathbb{R}^p$, nonnegative weights $\boldsymbol{\rho} \in \mathbb{R}^p_+$ ($\rho_{[1:k]} = \rho, \rho_{k+1:p} = 0$), vector $\hat{\boldsymbol{\nu}}$ ($\hat{\nu}_j = \text{prox}_{\rho_j H_M}(|\mu_j|)$), integer $k \in \mathbb{N}$ (first k elements subject to Huber penalty), and threshold $M > 0$ for the Huber loss function.

1: ▷*Initialization for the first block*
2: Initialize $b = 1$, $P_1 = \rho_1$, $S_1 = |\mu_1|$, $N_b = 1$, ν_1, $r_1 = 1$.
3: $\nu_{\text{prev}} = \hat{\nu}_1$, $j = 2$
4: **while** $j \leq n$ **do**
5: $\quad b = b + 1$
6: $\quad P_b = \rho_j$, $S_b = |\mu_j|$, $N_b = 1$, $\nu = \hat{\nu}_j$
7: $\quad$ ▷*If the value for the current singleton block is greater that of the previous block (isotonic violation), merge the current block with the previous block*
8: $\quad$ **if** $\nu > v_{\text{prev}}$ **then**
9: $\quad\quad b = b - 1$
10: $\quad\quad P_b = P_b + \rho_j$, $S_b = S_b + |\mu_j|$, $N_b = N_b + 1$, $\nu = \text{prox}_{\frac{P_b}{N_b} H_M}(\frac{S_b}{N_b})$
11: $\quad\quad$ ▷*Look forward: keep merging the current block with the next block if the isotonic violation persists*
12: $\quad\quad$ **while** $j < n$ **and** $\nu \leq \hat{\nu}_j$ **do**
13: $\quad\quad\quad j = j + 1$
14: $\quad\quad\quad P_b = P_b + \rho_j$, $S_b = S_b + |\mu_j|$, $N_b = N_b + 1$, $\nu = \text{prox}_{\frac{P_b}{N_b} H_M}(\frac{S_b}{N_b})$
15: $\quad\quad$ **end while**
16: $\quad\quad$ ▷*Look backward: keep merging the current block with the previous block if the isotonic violation persists*
17: $\quad\quad$ **while** $b > 1$ **and** $\nu_{b-1} < \nu$ **do**
18: $\quad\quad\quad b = b - 1$
19: $\quad\quad\quad P_b = P_b + P_{b+1}$, $S_b = S_b + S_{b+1}$, $N_b = N_b + N_{b+1}$, $\nu = \text{prox}_{\frac{P_b}{N_b} H_M}(\frac{S_b}{N_b})$
20: $\quad\quad$ **end while**
21: $\quad$ **end if**
22: $\quad$ ▷*Save the current block's value and the index of the last element in the block*
23: $\quad \nu_b = \nu$, $r_b = j$
24: $\quad$ ▷*Start fresh on the next element*
25: $\quad \nu_{\text{prev}} = \nu$, $j = j + 1$
26: **end while**
27: ▷*Modify the output vector to have the same new value for all elements in each block*
28: **for** $l = 1, ..., b$ **do**
29: $\quad \hat{\nu}_{[r_{l-1}+1:r_l]} = \nu_l$
30: **end for**
31: **return** $\hat{\boldsymbol{\nu}}$

Thus, Algorithm 4 merges two adjacent blocks if the isotonic violation persists, and the output of the proximal operator is the minimizer of the univariate function in the merged block. This is exactly the same as the objective in lines 11-14 of Algorithm 1. Hence, the first claim follows.

To show that the algorithm runs in linear time, notice that in the while loop $j \leq p$ in Algorithm 4, the variable j is incremented by 1 in each iteration, and the loop terminates when $j = p$. Although there are two while loops inside the main while loop, the total number of iterations in the two inner while loops is at most p. This is because we start with p blocks, and each iteration of the inner while loops either merges two blocks forward or merges two blocks backward. The total number of merging operations is at most $p - 1$. Thus, the total number of iterations in the while loop $j \leq p$ is at most p. Lastly, since we can evaluate the proximal operator of the Huber loss function, $\mathbf{H}_M$, in constant time complexity, the total time complexity of Algorithm 4 is $O(p)$. □

B.5 Proof of Theorem 2.2

Theorem 2.2. *For any $\boldsymbol{\mu} \in \mathbb{R}^p$, PAVA (Algorithm 1) evaluates $\text{prox}_{\rho g^*}(\boldsymbol{\mu})$ exactly in $\mathcal{O}(p \log p)$.*

Proof. By Lemmas 2.3 and 2.4, the output of Algorithm 1 computes $\text{prox}_{\rho g^*}$ exactly. The log-linear time complexity statement also holds thanks to Lemma 2.5 and the initial sorting step. □

B.6 Proof of Theorem 2.6

Theorem 2.6. *For any $\boldsymbol{\beta} \in \mathbb{R}^p$, majorization (Algorithm 2) computes $g(\boldsymbol{\beta})$ exactly in $\mathcal{O}(p + p \log k)$.*

Proof. We first show that the algorithm correctly computes the value of $g(\boldsymbol{\beta})$ and then analyze its computational complexity. Define the mixed-binary set

$$\mathcal{S}_0 = \left\{ (t, \boldsymbol{\beta}) \;\middle|\; \tfrac{1}{2} \textstyle\sum_{j\in[p]} \beta_j^2 \leq t,\, \|\boldsymbol{\beta}\|_\infty \leq M,\, \|\boldsymbol{\beta}\|_0 \leq k \right\}.$$

Using the perspective and big-M reformulation techniques, the set $\mathcal{S}_0$ admits the equivalent representation

$$\mathcal{S}_0 = \left\{ (t, \boldsymbol{\beta}) \;\middle|\; \exists \boldsymbol{z} \in \{0,1\}^p \text{ s.t. } \tfrac{1}{2} \textstyle\sum_{j\in[p]} \beta_j^2 / z_j \leq t,\, \mathbf{1}^\top \boldsymbol{z} \leq k,\, -Mz_j \leq \beta_j \leq Mz_j \;\; \forall j \in [p] \right\}.$$

One can show that the closed convex hull of $\mathcal{S}_0$ is given by [16]

$$\operatorname{cl}\operatorname{conv}(\mathcal{S}_0) = \left\{ (t, \boldsymbol{\beta}) \;\middle|\; \exists \boldsymbol{z} \in [0,1]^p \text{ s.t. } \tfrac{1}{2} \textstyle\sum_{j\in[p]} \beta_j^2 / z_j \leq t,\, \mathbf{1}^\top \boldsymbol{z} \leq k,\, -Mz_j \leq \beta_j \leq Mz_j \;\; \forall j \in [p] \right\}.$$

Therefore, the implicit function g can be written as the evaluation of the support function of $\operatorname{cl}\operatorname{conv}(\mathcal{S}_0)$ at $(1, \mathbf{0})$, that is,

$$g(\boldsymbol{\beta}) = \min\{t : (t, \boldsymbol{\beta}) \in \operatorname{cl}\operatorname{conv}(\mathcal{S}_0)\}. \tag{14}$$

Notice that the set $\mathcal{S}_0$ is sign- and permutation-invariants. Hence, by [9, Theorem 4], its closed convex hull admits the following (different) lifted represenation

$$\operatorname{cl}\operatorname{conv}(\mathcal{S}_0) = \left\{ (t, \boldsymbol{\beta}) \;\middle|\; \exists \boldsymbol{\phi} \in \mathbb{R}^p \text{ s.t. } \begin{array}{l} \frac{1}{2} \sum_{j\in[p]} \phi_j^2 \leq t,\, |\boldsymbol{\beta}| \preceq_m \boldsymbol{\phi}, \\ 0 \leq \phi_k \leq \ldots \leq \phi_1 \leq M, \\ \phi_{k+1} = \phi_{k+2} = \ldots = \phi_n = 0 \end{array} \right\}, \tag{15}$$

where the absolute value operator $|\cdot|$ is applied to a vector in an element-wise fashion, and the constraint $|\boldsymbol{\beta}| \preceq_m \boldsymbol{\phi}$ denotes that $\boldsymbol{\phi}$ majorizes $|\boldsymbol{\beta}|$, that is,

$$|\boldsymbol{\beta}| \preceq_m \boldsymbol{\phi} \quad \Longleftrightarrow \quad \textstyle\sum_{j\in[l]} |\beta_j| \leq \sum_{j\in[l]} \phi_j \quad \forall l \in [p-1] \quad \text{and} \quad \sum_{j\in[p]} \phi_j = \sum_{j\in[p]} |\beta_j|.$$

Using this alternative convex hull description of $\mathcal{S}_0$ in (15) and the implicit formulation (14), we may conclude that

$$g(\boldsymbol{\beta}) = \min_{\boldsymbol{\phi}\in\mathbb{R}^p} \left\{ \tfrac{1}{2} \textstyle\sum_{j\in[p]} \phi_j^2 : |\boldsymbol{\beta}| \preceq_m \boldsymbol{\phi},\, 0 \leq \phi_k \leq \ldots \leq \phi_1 \leq M,\, \phi_{k+1} = \phi_{k+2} = \ldots = \phi_n = 0 \right\}. \tag{16}$$

In the following we show that Algorithm 2 can efficiently solve the minimization problem in (16). At the first iteration $j = 1$ of the algorithm, we have

$$k\phi_1 \geq \textstyle\sum_{j\in[k]} \phi_j = \sum_{j\in[p]} \phi_j \geq \sum_{j\in[p]} |\beta_j| \quad \Rightarrow \quad \phi_1 \geq \frac{1}{k} \sum_{j\in[p]} |\beta_j|.$$

At the same time, we also need to satisfy $|\beta_1| \leq \phi_1$ from the first majorization constraint. We now discuss two cases

- ◇ **Case 1:** If $\frac{1}{k} \sum_{j\in[p]} |\beta_j| \geq |\beta_1|$, in order to solve the minimization problem in (16), we set $\phi_1 = \frac{1}{k} \sum_{j=1}^n |\beta_j|$. Notice that $\phi_1 \leq M$ is automatically satisfied because $\phi_1 = \frac{1}{k} \sum_{j\in[p]} |\beta_j| = \frac{1}{k} \sum_{j\in[p]} Mz_j \leq M$. This leads to $\phi_2 = \ldots = \phi_k = \frac{1}{k} \sum_{j\in[p]} |\beta_j|$. To see this, for the sake of contradition, assume that $\exists j \in \{2, \ldots, k\}$ such that $\phi_j < \frac{1}{k} \sum_{j\in[p]} |\beta_j|$. Since $\phi_j \leq \phi_1 = \frac{1}{k} \sum_{j\in[p]} |\beta_j|$, we have $\sum_{j\in[k]} \phi_j < \sum_{j\in[k]} \frac{1}{k} \sum_{j\in[p]} |\beta_j| = \sum_{j\in[p]} |\beta_j|$, which contradicts the majorization constraint.
- ◇ **Case 2:** If $\frac{1}{k} \sum_{j\in[n]} |\beta_j| < |\beta_1|$, we can set $\phi_1 = |\beta_1|$. Notice that $\phi_1 \leq M$ is automatically satisfied because $|\beta_1| \leq Mz_1 \leq M$. Then we are left with $k - 1$ coefficients to set, and we can follow the same argument as we did for $j = 1$ with slight difference that the majorization constraints are changed to

$$\textstyle\sum_{j=2}^l \phi_j \geq \sum_{j=2}^l |\beta_j| \quad \forall l \in \{2, \ldots, p-1\} \quad \text{and} \quad \sum_{j=2}^p \phi_j = \sum_{j=2}^p |\beta_j|.$$

We repeat this process until we set all k coefficients $\phi_1, \ldots, \phi_k$, as implemented by Algorithm 2. The output of the algorithm coincides with the optimal value of the minimization problem in (16). Hence, the first claim follows.

As for the complexity claim, it is easy to see that Algorithm 2. only requires partial sorting step on Line 2, which has a complexity of $\mathcal{O}(p \log k)$. The summation step on Line 3 has a complexity of $\mathcal{O}(p)$. The for-loop step on Line 4-8 has a complexity of $\mathcal{O}(k)$, so does the final summation step on Line 9. Therefore, the overall computational complexity of Algorithm 2 is $\mathcal{O}(p + p \log k)$. This concludes the proof. □