Feature splitting parallel algorithm for Dantzig selectors

Xiaofei Wu[1†], Yue Chao[2†], Rongmei Liang[3], Shi Tang[3], Zhiming Zhang[1*]

[1]College of Mathematics and Statistics, Chongqing University, Huxi, Chongqing, 401331, Chongqing, China.
[2]Department of Statistics and Data Science, Xiamen University, Siming, Xiamen, 361005, Fujian, China.
[3]Big Data and Intelligence Engineering School, Chongqing College of International Business and Economics, Xuefu, Chongqing, 401520, Chongqing, China.

*Corresponding author(s). E-mail(s): zmzhang@cqu.edu.cn;
Contributing authors: xfwu1016@163.com; cyasy@xmu.edu.cn;
liang_r_m@163.com; loveqq200305@163.com;
†These authors contributed equally to this work.

Abstract

The Dantzig selector is a widely used and effective method for variable selection in ultra-high-dimensional data. Feature splitting is an efficient processing technique that involves dividing these ultra-high-dimensional variable datasets into manageable subsets that can be stored and processed more easily on a single machine. This paper proposes a variable splitting parallel algorithm for solving both convex and nonconvex Dantzig selectors based on the proximal point algorithm. The primary advantage of our parallel algorithm, compared to existing parallel approaches, is the significantly reduced number of iteration variables, which greatly enhances computational efficiency and accelerates the convergence speed of the algorithm. Furthermore, we show that our solution remains unchanged regardless of how the data is partitioned, a property referred to as partition-insensitive. In theory, we use a concise proof framework to demonstrate that the algorithm exhibits linear convergence. Numerical experiments indicate that our algorithm performs competitively in both parallel and nonparallel environments. The R package for implementing the proposed algorithm can be obtained at https://github.com/xfwu1016/PPADS.

1

1 Introduction

As scientific and technological advancements progress, data collection has become increasingly easier. One manifestation of this is the growing number of features (variables) in datasets, even though the number of truly useful features remains limited. Consequently, many variable selection methods have been proposed, such as subset selection, stepwise regression, and regularization techniques. Among these, the Dantzig selector (abbreviated as DS), introduced by Candès and Tao (2007b), is one of the most well-known methods for ultra-high-dimensional data (the number of features far exceeds the number of samples), defined as follows,

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^p} \quad \|\boldsymbol{\beta}\|_1 \quad \text{s.t.} \quad \|\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y})/n\|_\infty \leq \lambda, \tag{1}$$

where

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{X}_1 \ \dots \ \boldsymbol{X}_p \end{pmatrix} \text{ and } \boldsymbol{y} = \begin{pmatrix} y_1 \ \dots \ y_n \end{pmatrix}^\top$$

denote the $n \times p$ design matrix and the response vector, respectively; λ denotes a non-negative tuning parameter. Here, we write $\|\mathbf{x}\|_q$ for the ℓ_q norm of $\mathbf{x} \in \mathbb{R}^p$, where $1 \leq q \leq \infty$. We operate under the assumption of a linear regression model $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon}$ represents the error vector. The true regression coefficient vector β remains generally unknown. The parameter λ is of paramount importance. A smaller λ tightens the constraint $\|\boldsymbol{X}^\top(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})\|_\infty \leq \lambda$, resulting in a sparser estimator. In contrast, a larger λ relaxes the constraint, potentially leading to the DS estimator having more non-zero elements. DS was devised to tackle the variable selection and estimation challenges in ultra-high-dimensional linear regression. When the number of features p significantly surpasses the sample size n, traditional linear regression methods become ineffective. DS overcomes this limitation by integrating an ℓ_1 norm constraint and an inequality constraint, enabling efficient variable selection and estimation. Moreover, some nonconvex DS models in Wen et al. (2024) possesses the Oracle property, signifying that under specific conditions, it can identify the true variable set with a probability approaching 1, and the estimated coefficients exhibit asymptotic normality. Next, we will review the theoretical and algorithmic work on DS and some of its variants.

The optimal ℓ_2 rate properties for estimator of DS were established under a sparsity scenario, and impressive empirical performance on real-world problems involving large values of p was demonstrated in Candès and Tao (2007b). Due to its outstanding performance in managing ultra-high dimensional data and its tight error bounds, the Dantzig selector (DS) has attracted considerable attention. Discussions on the Dantzig selector can be found in the works of Bickel (2007), Cai and Lv (2007), Candès and Tao (2007a), Efron et al. (2007), Friedlander and Saunders (2007), Meinshausen et al. (2007), and Ritov (2007). In addition, the approximate equivalence relationship

2

between DS and another popular variable selection method Lasso (Tibshirani (1996)) was established by Bickel et al. (2008) and James et al. (2008).

Many studies on DS variants have emerged in recent years. Dicker and Lin (2013) proposed the adaptive DS to enhance the selection performance of the original DS by incorporating the adaptive weighting concept introduced by Zou (2006). Liu et al. (2010) suggested using group regularization terms instead of ℓ_1 regularization to develop a group DS, allowing it to better adapt to data containing grouping prior information. Chatterjee et al. (2014) introduced a generalized DS for linear models, where any norm that captures the parameter structure can be utilized for estimation. Recently, Ge and Li (2021) and Wen et al. (2024) proposed the nonconvex DSs based on $\ell_1 - \alpha\ell_2$ regularization, SCAD and MCP. DS has also been extended to other model settings, including generalized linear models in James et al. (2008), censored linear regression models in Li et al. (2014), and multiple quantile regression models in Park et al. (2017).

As discussed by Candès and Tao (2007b), a natural approach to solving (1) is to reformulate it as a linear programming (LP) problem using a primal–dual interior point algorithm (Boyd and Vandenberghe (2004)). However, interior point methods are typically inefficient for high-dimensional problems ($p > n$), as they require solving dense Newton systems at each iteration. An alternative approach for solving (1) involves using homotopy methods to compute the entire solution path of the Dantzig selector, see James et al. (2008). Nonetheless, as noted in Becker et al. (2010) (Section 1.2), these methods also struggle with high-dimensional problems. To enable the DS to be applicable to high-dimensional datasets, Becker et al. (2010) reformulated (1) as a linear cone programming problem, for which a smooth approximation of its dual problem is solved using a projected gradient algorithm (Beck and Teboulle (2009)).

Observing (1), it contains ℓ_1 and ℓ_∞ terms that are not smooth. A widely applicable algorithm for addressing this is the Alternating Direction Method of Multipliers (ADMM, Boyd et al. (2010)). Lu et al. (2012) was the first to rewrite (1) into an optimization form with separable equality constraints, and utilize ADMM to find the Dantzig selector. Numerical experiments in Lu et al. (2012) demonstrated that this method generally outperforms the approach presented by Becker et al. (2010) in terms of CPU time while producing solutions of comparable quality. It is worth noting that ADMM is an iterative algorithm, and the efficiency of the algorithm depends on whether each subproblem can be efficiently solved. In each iteration of ADMM described by Lu et al. (2012), two subproblems needed to be solved successively. One of the subproblems had a closed-form solution, while the other did not and was approximated using a nonmonotone gradient method (NGM). However, NGM is also an iterative algorithm, and embedding it within ADMM can lead to double loops, resulting in excessive computational burden. To address the challenges arising from the subproblem with double loops, Wang and Yuan (2012) proposed a linearized version of ADMM (LADMM) specifically tailored for the Dantzig selector. This modified approach demonstrated superior efficiency compared to ADMM in Lu et al. (2012) when solving both synthetic and real-world datasets. Li et al. (2015) encapsulated the

3

LADMM algorithm in an efficient and user-friendly R package called "flare". In addition, the ADMM algorithm and LADMM have been employed by Ge and Li (2021) and Chatterjee et al. (2014) to solve the $\ell_1 - \alpha\ell_2$ DS and the generalized DS, respectively.

In addition to the above two popular ADMM algorithms, several other algorithms have been proposed to solve the DS. With the aid of proximal operators, Prater et al. (2015) skillfully introduced two simple iterative methods through a fixed-point reformulation of the solutions to (1). They noted that their first iterative algorithm followed the same steps as the LADMM iteration presented by Wang and Yuan (2012), while the second iterative algorithm aligned with the steps of the primal-dual iteration algorithm introduced by Chambolle and Pock (2011) for DS. A fast splitting method introduced in He et al. (2015) shows competitive performance when compared to both ADMM and LADMM. Subsequently, a series of other splitting algorithms, including partially linearized ADMM and customized proximal point algorithms (CPPA, proposed by Gu et al. (2014)), has been proposed in He and Xu (2017) to tackle the DS. Recently, Fang et al. (2021) reformulated DS problem as an equivalent convex composite optimization problem and proposes a semismooth Newton augmented Lagrangian algorithm to solve it, while also applying a proximal point dual semismooth Newton algorithm for another equivalent form of the problem. However, the iterative algorithms mentioned above were designed by formulating the optimization of DS as a constrained optimization problem. During the solution process of each subproblem, it was necessary to introduce dual variables to eliminate the constraints. Mao et al. (2021) was the first to reformulate DS as an unconstrained optimization problem and proposed a partially proximal linearized alternating minimization method (P-PLAM). Notably, its iterative process does not require updates of dual variables, and two subproblems have straightforward closed-form solutions.

It is worth noting that the aforementioned algorithms do not consider the use of variable splitting and parallel computation frameworks to enhance the efficiency of DS in handling high-dimensional data. More recently, inspired by the works of Sun et al. (2015), Li et al. (2023), Wen et al. (2023) and Cai et al. (2024), Wen et al. (2024) proposed an efficient computational algorithm for solving DS and nonconvex DSs, utilizing a feature splitting approach. This approach not only reduces memory usage but also facilitates the parallel implementation of the computational framework. This parallel algorithm is based on the three-block ADMM algorithm proposed by Sun et al. (2015). However, as noted by Chen et al. (2016), the directly extended three-block ADMM algorithm cannot theoretically guarantee convergence. To ensure convergence, Sun et al. (2015) introduced updates for certain intermediate variables at each iteration. Although this algorithm has theoretical convergence properties, its application within a parallel computing structure results in excessive redundant intermediate variables. As pointed out by Lin et al. (2022), these excessive redundant intermediate variables in ADMM algorithm can degrade solution precision and slow down algorithm convergence. Specifically, the parallel algorithm proposed by Wen et al. (2024) necessitates updating $3Kp$ variables per iteration, where K represents the number of variable partitions. Clearly, in the context of ultra-high dimensions, if K is notably large, the number of variables that the algorithm needs to update becomes excessive.

This paper aims to propose a unified optimization algorithm for solving both DS and nonconvex DS models, effectively mitigating the issue of excessive intermediate variables associated with the three-block ADMM in Wen et al. (2024). This new parallel algorithm is framed within the PPA (proximal point algorithms in Parikh and Boyd (2013)) structure. Compared to the three-block ADMM algorithm in Wen et al. (2024), our algorithm has the following three significant advantages.

1. The proposed algorithm requires updating only $3p$ variables in each iteration, which is independent of the number of variable partitions K. Fewer updated variables not only enhance the computational speed of each iteration, accelerating algorithm convergence, but also result in solutions with higher precision.
2. Our parallel algorithm exhibits partitioning insensitivity through specific structural design. Specifically, in parallel environment, the solution of algorithm remains invariant regardless of how the matrix is partitioned. The partition insensitivity ensures the reliability and accuracy of solutions in a parallel computing environment, independent of the chosen parallelization strategy. This facilitates a more flexible adaptation of computing resources to accommodate diverse computational requirements and hardware setups.
3. We provide a concise proof demonstrating that the proposed algorithm achieves a linear convergence rate. The linear convergence speed of the algorithm ensures stable progress and efficient computational performance, enabling reliable approximation of the exact solution in a shorter amount of time.

The structure of this paper is organized as follows. In Section 2, we review relevant algorithms for solving DS. Section 3 discusses two parallel PPA algorithms for solving DS and its nonconvex variants, including their insensitivity to algorithm partitioning and convergence properties. Synthetic numerical simulations are presented in Section 4, while real data is discussed in Section 5. Section 6 summarizes the findings and concludes the paper, with a discussion on future research directions. Technical proofs and supplementary experiments are included in the Appendix. The R package for implementing the proposed algorithm can be obtained at https://github.com/xfwu1016/PPADS.

Notations: $\mathbf{0}_n$ and $\mathbf{1}_n$ represent n-dimensional vectors with all elements being 0 and 1, respectively. $\boldsymbol{I}_n$ represents the n-dimensional identity matrix. The Hadamard product is denoted by $\odot$. The $\text{sign}(\cdot)$ function is defined component-wise such that $\text{sign}(t) = 1$ if $t > 0$, $\text{sign}(t) = 0$ if $t = 0$, and $\text{sign}(t) = -1$ if $t < 0$. $(\cdot)_+$ signifies the element-wise operation of extracting the positive component, while $|\cdot|$ denotes the element-wise absolute value function. For any vector $\boldsymbol{u}$, $\|\boldsymbol{u}\|_1$, $\|\boldsymbol{u}\|_2$, and $\|\boldsymbol{u}\|_\infty$ denote the ℓ_1 norm, the ℓ_2 norm, and the ℓ_∞ norm of $\boldsymbol{u}$, respectively. $\|\boldsymbol{u}\|_{\boldsymbol{H}} := \sqrt{\boldsymbol{u}^\top \boldsymbol{H} \boldsymbol{u}}$ is used to denote the norm of $\boldsymbol{u}$ under the matrix $\boldsymbol{H}$, where $\boldsymbol{H}$ is a matrix. If $\boldsymbol{H}$ is positive semi-definite, we denote it as $\boldsymbol{H} \succeq \mathbf{0}$; if the matrix is positive definite, we denote it as $\boldsymbol{H} \succ \mathbf{0}$.

2 Review of Related Algorithms

As the focus of this paper is on parallel computation, currently, the only algorithm capable of implementing parallel computation for the DS optimization objective function is ADMM in Wen et al. (2024). Therefore, this paper solely reviews some ADMM algorithms. For detailed introductions to other first-order and second-order algorithms for computing DS, readers can refer to the literature on algorithms mentioned in the introduction of this paper. ADMM algorithm is an iterative optimization method designed to tackle complex convex minimization problems with linear constraints. It operates by decomposing the original problem into smaller, more manageable subproblems that are easier to solve. ADMM alternates between optimizing these subproblems and updating dual variables to enforce the constraints, making it particularly effective for large-scale problems and various statistical learning applications. For a comprehensive review, readers can refer to Boyd et al. (2010).

In order to make (1) satisfy the condition that ADMM can handle problems, we need to introduce the auxiliary variable $\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y}) = \boldsymbol{z}$. To simplify the symbol, we make $\boldsymbol{A} = \boldsymbol{X}^\top \boldsymbol{X}$. Then, (1) can be rewritten into the following equation constrained optimization form,

$$
\begin{aligned}
\min_{\boldsymbol{\beta}, \boldsymbol{z}} \quad & \|\boldsymbol{\beta}\|_1 + \delta_{\mathcal{Z}_0(\boldsymbol{z})} \\
\text{s.t.} \quad & \boldsymbol{A}\boldsymbol{\beta} - \boldsymbol{z} = \boldsymbol{X}^\top \boldsymbol{y}.
\end{aligned} \tag{2}
$$

where $\mathcal{Z}_0(\boldsymbol{z}) = \{\boldsymbol{z} : \|\boldsymbol{z}\|_\infty \leq n\lambda\}$ and $\delta_{\mathcal{Z}_0(\boldsymbol{z})} = \begin{cases} 0, & \text{if } \boldsymbol{z} \in \mathcal{Z}_0, \\ +\infty, & \text{otherwise.} \end{cases}$

The augmented Lagrangian form of (2) is

$$
L_\mu(\boldsymbol{\beta}, \boldsymbol{z}; \boldsymbol{u}) = \|\boldsymbol{\beta}\|_1 + \delta_{\mathcal{Z}_0(\boldsymbol{z})} + \boldsymbol{u}^\top(\boldsymbol{A}\boldsymbol{\beta} - \boldsymbol{z} - \boldsymbol{X}^\top \boldsymbol{y}) + \frac{\mu}{2}\|\boldsymbol{A}\boldsymbol{\beta} - \boldsymbol{z} - \boldsymbol{X}^\top \boldsymbol{y}\|_2^2, \tag{3}
$$

where $\boldsymbol{u}$ is the dual variable corresponding to the linear constraint, and $\mu > 0$ is a given augmented parameter.

2.1 ADMM and LADMM for DS

Given $(\boldsymbol{z}^0, \boldsymbol{u}^0)$, the iterative scheme of ADMM for problem (3) is as follows,

$$
\begin{aligned}
\boldsymbol{\beta}^{t+1} &\leftarrow \arg\min_{\boldsymbol{\beta}} \left\{ L_\mu(\boldsymbol{\beta}, \boldsymbol{z}^t; \boldsymbol{u}^t) \right\}; \\
\boldsymbol{z}^{t+1} &\leftarrow \arg\min_{\boldsymbol{z}} \left\{ L_\mu(\boldsymbol{\beta}^{t+1}, \boldsymbol{z}; \boldsymbol{u}^t) \right\}; \\
\boldsymbol{u}^{t+1} &\leftarrow \boldsymbol{u}^t + \mu(\boldsymbol{A}\boldsymbol{\beta}^{t+1} - \boldsymbol{z}^{t+1} - \boldsymbol{X}^\top \boldsymbol{y}).
\end{aligned} \tag{4}
$$

Note that the ADMM iteration order mentioned in Lu et al. (2012) is $\boldsymbol{z} \to \boldsymbol{\beta} \to \boldsymbol{u}$. However, the order of the iterations does not affect the convergence or efficiency of the ADMM algorithm. The choice of the iteration order in (4) is made to maintain consistency with the LADMM presented in Wang and Yuan (2012) and the parallel ADMM described in Wen et al. (2024).

For the three subproblems of the ADMM iteration, the $\boldsymbol{\beta}$ subproblem

$$\boldsymbol{\beta}^{t+1} \leftarrow \arg\min_{\boldsymbol{\beta}} \left\{ \|\boldsymbol{\beta}\|_1 + \frac{\mu}{2}\|\boldsymbol{A}\boldsymbol{\beta} - \boldsymbol{z}^t - \boldsymbol{X}^\top \boldsymbol{y} + \frac{\boldsymbol{u}^t}{\mu}\|_2^2 \right\}, \tag{5}$$

which is similar to a Lasso problem and does not have a closed-form solution. Lu et al. (2012) suggests using a nonmonotone gradient method to iteratively solve it. The $\boldsymbol{z}$ subproblem has a closed-form solution (see Proposition 2.1 in Lu et al. (2012)), while the $\boldsymbol{u}$ subproblem involves a linear operation that is easy to implement. Wang and Yuan (2012) pointed out that the lack of a closed-form solution in the $\boldsymbol{\beta}$ step would impact the overall efficiency of the ADMM algorithm iteration. Therefore, it is recommended to use linearization methods to approximate this iteration step, ensuring that this subproblem yields a closed-form solution. Specifically, this involves adding a quadratic term to the objective function of the optimization problem for the $\boldsymbol{\beta}$ subproblem, as follows,

$$\boldsymbol{\beta}^{t+1} \leftarrow \arg\min_{\boldsymbol{\beta}} \left\{ \|\boldsymbol{\beta}\|_1 + \frac{\mu}{2}\|\boldsymbol{A}\boldsymbol{\beta} - \boldsymbol{z}^t - \boldsymbol{X}^\top \boldsymbol{y} + \frac{\boldsymbol{u}^t}{\mu}\|_2^2 + \frac{1}{2}\|\boldsymbol{\beta} - \boldsymbol{\beta}^t\|_{\boldsymbol{S}}^2 \right\}, \tag{6}$$

where $\boldsymbol{S} = \eta\boldsymbol{I}_p - \mu\boldsymbol{A}^\top\boldsymbol{A}$ and η is larger than the maximum eigenvalue of $\mu\boldsymbol{A}^\top\boldsymbol{A}$. Rewrite the optimization formula above to eliminate some constant terms unrelated to the optimization variable $\boldsymbol{\beta}$, and the subproblem of $\boldsymbol{\beta}$ becomes

$$\boldsymbol{\beta}^{t+1} \leftarrow \arg\min_{\boldsymbol{\beta}} \left\{ \|\boldsymbol{\beta}\|_1 + \frac{\eta}{2}\|\boldsymbol{\beta} - \boldsymbol{\beta}^t + \frac{\mu}{\eta}\boldsymbol{A}^\top(\boldsymbol{A}\boldsymbol{\beta}^t - \boldsymbol{z}^t - \boldsymbol{X}^\top \boldsymbol{y} + \frac{\boldsymbol{u}^t}{\mu})\|_2^2 \right\}. \tag{7}$$

Clearly, the above objective function for optimization is a soft-thresholding operator and has a closed-form solution.

In recent years, the linearization technique of the ADMM algorithm has garnered significant attention and been applied in various fields, such as Li et al. (2014), Gu et al. (2018), Liang et al. (2024), and Wu et al. (2025b). It is worth mentioning that Liang et al. (2024) provides an efficient power method for computing the maximum eigenvalue of a positive semidefinite matrix in the design of LADMM. The power method itself is an iterative algorithm, but each iteration only involves matrix-vector multiplications, without matrix-matrix multiplications. In terms of numerical performance, it typically converges in a few dozen steps at most. In LADMM, note that η does not require a very accurate solution; it only needs to be greater than the maximum eigenvalue of the positive semidefinite matrix. Therefore, under relaxed convergence conditions, the power method can converge more quickly. In this paper, we recommend using this efficient power method to compute the maximum eigenvalue of all positive semidefinite matrices.

2.2 Parallel ADMM for DS

In a parallel computing environment, the estimation coefficients of DS can be divided into K blocks, represented as $\boldsymbol{\beta} = (\boldsymbol{\beta}_{1\cdot}^\top, \boldsymbol{\beta}_{2\cdot}^\top, \cdots, \boldsymbol{\beta}_{K\cdot}^\top)^\top$, where each $\boldsymbol{\beta}_{i\cdot} \in \mathbb{R}^{p_i}$ and

$\sum_{i=1}^{K} p_i = p$. Correspondingly, we can partition the gram matrix $\boldsymbol{A} = \boldsymbol{X}^\top \boldsymbol{X}$ into $\boldsymbol{A} = (\boldsymbol{A}_1, \boldsymbol{A}_2, \cdots, \boldsymbol{A}_K)$, where $\boldsymbol{A}_i \in \mathbb{R}^{p \times p_i}$. Wen et al. (2024) introduced slack variables $\boldsymbol{\omega} = (\boldsymbol{\omega}_2, \cdots, \boldsymbol{\omega}_K)$, where $\boldsymbol{\omega}_i \in \mathbb{R}^p$ for $i \in \{2, \cdots, K\}$, to reformulate the optimization problem as follows:

$$
\begin{aligned}
\min_{\boldsymbol{\beta}, \boldsymbol{z}, \boldsymbol{\omega}} \quad & \sum_{i=1}^{K} \|\boldsymbol{\beta}_i\|_1 + \delta_{\mathcal{Z}_0(\boldsymbol{z})} \\
\text{s.t.} \quad & \boldsymbol{A}_1 \boldsymbol{\beta}_{1.} + \boldsymbol{\omega}_2 + \cdots + \boldsymbol{\omega}_K - \boldsymbol{z} = \boldsymbol{X}^\top \boldsymbol{y}, \\
& \omega_i = \boldsymbol{A}_i \boldsymbol{\beta}_i, \quad i \in \{2, \ldots, K\}.
\end{aligned}
\tag{8}
$$

The augmented Lagrangian form of (8) is

$$
\begin{aligned}
L_\mu(\boldsymbol{\beta}, \boldsymbol{z}, \boldsymbol{\omega}; \boldsymbol{u}) = & \sum_{i=1}^{K} \|\boldsymbol{\beta}_{i.}\|_1 + \delta_{\mathcal{Z}_0(\boldsymbol{z})} + \boldsymbol{u}_1^\top \left(\boldsymbol{A}_1 \boldsymbol{\beta}_{1.} + \sum_{i=2}^{K} \boldsymbol{\omega}_i - \boldsymbol{z} - \boldsymbol{X}^\top \boldsymbol{y} \right) \\
& + \frac{\mu}{2} \| \boldsymbol{A}_1 \boldsymbol{\beta}_{1.} + \sum_{i=2}^{K} \boldsymbol{\omega}_i - \boldsymbol{z} - \boldsymbol{X}^\top \boldsymbol{y} \|_2^2 \\
& + \sum_{i=2}^{K} \left[\boldsymbol{u}_i^\top \left(\boldsymbol{A}_i \boldsymbol{\beta}_{i.} - \boldsymbol{\omega}_i \right) + \frac{\mu}{2} \| \boldsymbol{A}_i \boldsymbol{\beta}_{i.} - \boldsymbol{\omega}_i \|_2^2 \right].
\end{aligned}
\tag{9}
$$

It is easy to see that the separability of the K blocks of $\boldsymbol{\beta}$ in the augmented Lagrangian function enables a natural parallelization for updating $\boldsymbol{\beta}$. Similar to the iterative steps of Sun et al. (2015) and Wen et al. (2023), Wen et al. (2024) uses the following iterative procedure,

$$
\begin{cases}
\boldsymbol{\beta}_{1.}^{t+1} \leftarrow \arg\min_{\boldsymbol{\beta}_{1.}} \left\{ \|\boldsymbol{\beta}_{1.}\|_1 + \frac{\mu}{2} \| \boldsymbol{A}_1 \boldsymbol{\beta}_{1.} + \sum_{i=2}^{K} \boldsymbol{\omega}_i^t - \boldsymbol{z}^t - \boldsymbol{X}^\top \boldsymbol{y} + \frac{\boldsymbol{u}_1^t}{\mu} \|_2^2 + \frac{1}{2} \| \boldsymbol{\beta}_{1.} - \boldsymbol{\beta}_{1.}^t \|_{\boldsymbol{S}_1}^2 \right\}; \\
\boldsymbol{\beta}_{i.}^{t+1} \leftarrow \arg\min_{\boldsymbol{\beta}_{i.}} \left\{ \|\boldsymbol{\beta}_{i.}\|_1 + \frac{\mu}{2} \| \boldsymbol{A}_i \boldsymbol{\beta}_{i.} - \boldsymbol{\omega}_i^t + \frac{\boldsymbol{u}_i^t}{\mu} \|_2^2 + \frac{1}{2} \| \boldsymbol{\beta}_{i.} - \boldsymbol{\beta}_{i.}^t \|_{\boldsymbol{S}_i}^2 \right\}, i \in \{2, \cdots, K\}; \\
\boldsymbol{\omega}_i^{t+\frac{1}{2}} \leftarrow (\boldsymbol{X}^\top \boldsymbol{y} + \boldsymbol{z}^t + K \boldsymbol{A}_i \boldsymbol{\beta}_{i.}^{t+1} - \boldsymbol{A} \boldsymbol{\beta}^{t+1})/K, i \in \{2, \cdots, K\}; \\
\boldsymbol{z}^{t+1} \leftarrow \arg\min_{\boldsymbol{z}} \left\{ \delta_{\mathcal{Z}_0(\boldsymbol{z})} + \frac{\mu}{2} \| \boldsymbol{A}_1 \boldsymbol{\beta}_{1.}^{t+1} + \sum_{i=2}^{K} \boldsymbol{\omega}_i^{t+\frac{1}{2}} - \boldsymbol{z} - \boldsymbol{X}^\top \boldsymbol{y} + \frac{\boldsymbol{u}_1^t}{\mu} \|_2^2 \right\}; \\
\boldsymbol{\omega}_i^{t+1} \leftarrow (\boldsymbol{X}^\top \boldsymbol{y} + \boldsymbol{z}^{t+1} + K \boldsymbol{A}_i \boldsymbol{\beta}_{i.}^{t+1} - \boldsymbol{A} \boldsymbol{\beta}^{t+1})/K, i \in \{2, \cdots, K\}; \\
\boldsymbol{u}_1^{k+1} \leftarrow \boldsymbol{u}_1^k + \mu (\boldsymbol{A}_1 \boldsymbol{\beta}_{1.}^{t+1} + \sum_{i=2}^{K} \boldsymbol{\omega}_i^{t+1} - \boldsymbol{z}^{t+1} - \boldsymbol{X}^\top \boldsymbol{y}); \\
\boldsymbol{u}_i^{k+1} \leftarrow \boldsymbol{u}_i^k + \mu \left(\boldsymbol{A}_i \boldsymbol{\beta}_{i.}^{t+1} - \boldsymbol{\omega}_i^{t+1} \right), i \in \{2, \cdots, K\}.
\end{cases}
$$

The updates for $\boldsymbol{\beta}_{1.}$ and $\boldsymbol{\beta}_{i.}$ are carried out after linearization, where $\boldsymbol{S}_i = \eta_i \boldsymbol{I}_p - \mu \boldsymbol{A}_i^\top \boldsymbol{A}_i$, with η_i required to exceed the maximum eigenvalue of $\mu \boldsymbol{A}_i^\top \boldsymbol{A}_i$. It is not

difficult to observe that, similar to (7), the updates for $\boldsymbol{\beta}_1.$ and $\boldsymbol{\beta}_{i\cdot}$ have closed-form solutions. The update for $\boldsymbol{z}$ is analogous to (4), and a closed-form solution can also be derived based on Proposition 2.1 from Lu et al. (2012). The updates of $\boldsymbol{\omega}_i$ and $\boldsymbol{u}_i$ are simple and easy to implement algebraic operations. Clearly, $\boldsymbol{\omega}_i$ and $\boldsymbol{u}_i$ are ultra-high dimensional vector constructed to achieve parallel computing effects. Thus, $\boldsymbol{\omega}$ (including all $\boldsymbol{\omega}_i$) is a $(K-1)p$-dimensional vector, while $\boldsymbol{u}$ (including all $\boldsymbol{u}_i$) is a Kp-dimensional vector. Due to its excessively large dimensionality, it reduces the convergence speed of the algorithm during implementation and lowers the precision of the solution.

3 Proximal Point Algorithms for DS

Proximal Point Algorithms (PPA) are iterative optimization methods used to solve convex optimization problems, particularly those involving non-smooth functions. The core idea of these algorithms is to replace the original problem with a series of simpler subproblems that can be more easily solved using the proximal operator. The ability of PPA to handle non-smooth losses and the iterative form of variable splitting are similar to those of the ADMM algorithm. Parikh and Boyd (2013) offers a comprehensive overview and introduction to PPA and proximal operators, while Cai et al. (2013) delves into a detailed discussion on the relationship between PPA and ADMM algorithms.

However, in the fields of statistics and machine learning, PPA is far less well-known than ADMM. This difference may be attributed to three main reasons: first, comprehensive review articles on ADMM (Boyd et al. (2010)) were published earlier; second, PPA and ADMM have a high degree of overlap in their applicable scenarios; and finally, in practice, PPA iterative forms tend to be more flexible and varied, whereas the alternating iteration format of the ADMM algorithm is more widely recognized. Using the two PPAs for solving DS proposed by He and Xu (2017) (Algorithms 4 and 5 in their paper) as examples, these PPAs consist of two iterative steps. The first step, akin to the ADMM iteration, is called the prediction step, while the second step is referred to as the correction step, which essentially serves as a relaxation step. However, the prediction phase does not have a fixed iteration sequence like the ADMM algorithm (where one variable is held constant while the other is updated); instead, it allows for a variable iteration order. In their Algorithm 4, the dual variables are updated first, followed by the primal variables, whereas in their Algorithm 5, the primal variables are updated first, followed by the dual variable. This variability, while making the PPA approach more flexible for solving problems, also hinders its application in other disciplines due to its lack of memorability.

In this paper, in order to make PPA easier to remember and use, we adopt an iterative approach similar to the ADMM algorithm, which is a primal-dual alternating iteration. Next, we will demonstrate that the PPA is more suitable than ADMM for solving DS in the context of parallel computation. Before introducing the PPA algorithm proposed in this paper, we first review the constrained optimization form

of DS,

$$\min_{\boldsymbol{\beta}, \boldsymbol{z}} \quad \|\boldsymbol{\beta}\|_1 + \delta_{\mathcal{Z}_0(\boldsymbol{z})}$$
$$\text{s.t.} \quad \boldsymbol{A}\boldsymbol{\beta} - \boldsymbol{z} = \boldsymbol{X}^\top \boldsymbol{y}. \tag{10}$$

The Lagrange multiplier form of (10) is

$$L(\boldsymbol{\beta}, \boldsymbol{z}; \boldsymbol{u}) = \|\boldsymbol{\beta}\|_1 + \delta_{\mathcal{Z}_0(\boldsymbol{z})} - \boldsymbol{u}^\top(\boldsymbol{A}\boldsymbol{\beta} - \boldsymbol{z} - \boldsymbol{X}^\top \boldsymbol{y}),$$

where $\boldsymbol{u}$ is a Lagrange multiplier (dual variable).

3.1 Nonparallel version

With the Lagrangian form of DS, we can introduce our PPA method in the case where the features are not split. With given $(\boldsymbol{\beta}^0, \boldsymbol{z}^0, \boldsymbol{u}^0)$, the iterative scheme of the PPA for problem (10) is as follows:

$$\boldsymbol{\beta}^{t+1} \leftarrow \arg\min_{\boldsymbol{\beta}} \left\{ L(\boldsymbol{\beta}, \boldsymbol{z}^t; \boldsymbol{u}^t) + \frac{\mu}{2}\|\boldsymbol{A}(\boldsymbol{\beta} - \boldsymbol{\beta}^t)\|_2^2 \right\};$$
$$\boldsymbol{z}^{t+1} \leftarrow \arg\min_{\boldsymbol{z}} \left\{ L(\boldsymbol{\beta}^{t+1}, \boldsymbol{z}; \boldsymbol{u}^t) + \frac{\mu}{2}\|\boldsymbol{z} - \boldsymbol{z}^t\|_2^2 \right\}; \tag{11}$$
$$\boldsymbol{u}^{t+1} \leftarrow \boldsymbol{u}^t - \frac{\mu}{2}\left[2(\boldsymbol{A}\boldsymbol{\beta}^{t+1} - \boldsymbol{z}^{t+1} - \boldsymbol{X}^\top \boldsymbol{y}) - (\boldsymbol{A}\boldsymbol{\beta}^t - \boldsymbol{z}^t - \boldsymbol{X}^\top \boldsymbol{y}) \right];$$

where $\mu > 0$ is a given augmented parameter. Note that the Lagrange multiplier function L used in equation (11) differ from the augmented Lagrange multiplier L_μ employed in ADMM and LADMM, as they do not include quadratic augmented terms. However, the quadratic augmented terms are crucial for the convergence of ADMM and PPA, as discussed in Boyd et al. (2010) and Parikh and Boyd (2013). Therefore, we also introduce the corresponding quadratic augmented terms for the subproblems of $\boldsymbol{\beta}^{t+1}$ and $\boldsymbol{z}^{t+1}$, denoted as $\|\boldsymbol{A}(\boldsymbol{\beta} - \boldsymbol{\beta}^t)\|_2^2$ and $\|\boldsymbol{z} - \boldsymbol{z}^t\|_2^2$, respectively. Compared to the quadratic augmented term in ADMM or LADMM, the augmented term in PPA is easier to manage and more conducive to expansion in parallel computing environments.

For the $\boldsymbol{\beta}$ subproblem of PPA, by organizing the optimized form of $\boldsymbol{\beta}$, it can be concluded that,

$$\boldsymbol{\beta}^{t+1} \leftarrow \arg\min_{\boldsymbol{\beta}} \left\{ \|\boldsymbol{\beta}\|_1 + \frac{\mu}{2}\|\boldsymbol{A}\boldsymbol{\beta} - \boldsymbol{A}\boldsymbol{\beta}^t - \frac{\boldsymbol{u}^t}{\mu}\|_2^2 \right\}.$$

Similar to (5), it is a Lasso problem without a closed-form solution. Thus, we can also use linearization techniques to give it a closed-form solution, that is

$$\boldsymbol{\beta}^{t+1} \leftarrow \arg\min_{\boldsymbol{\beta}} \left\{ \|\boldsymbol{\beta}\|_1 + \frac{\mu}{2}\|\boldsymbol{A}\boldsymbol{\beta} - \boldsymbol{A}\boldsymbol{\beta}^t - \frac{\boldsymbol{u}^t}{\mu}\|_2^2 + \frac{1}{2}\|\boldsymbol{\beta} - \boldsymbol{\beta}^t\|_{\boldsymbol{S}}^2 \right\} \tag{12}$$

$$= \arg\min_{\boldsymbol{\beta}} \left\{ \|\boldsymbol{\beta}\|_1 + \frac{\eta}{2} \|\boldsymbol{\beta} - \boldsymbol{\beta}^t - \frac{\boldsymbol{A}^\top \boldsymbol{u}^t}{\eta}\|_2^2 \right\},$$

where $\boldsymbol{S} = \eta \boldsymbol{I}_p - \mu \boldsymbol{A}^\top \boldsymbol{A}$, and η is chosen to be larger than the maximum eigenvalue of $\mu \boldsymbol{A}^\top \boldsymbol{A}$ (eigen($\mu \boldsymbol{A}^\top \boldsymbol{A}$)) to ensure that the $\boldsymbol{S}$ matrix is positive definite. Clearly, the above expression is a soft thresholding operator, which has the following closed-form solution:

$$\boldsymbol{\beta}^{t+1} \leftarrow \text{sign}(\boldsymbol{\beta}^t + \frac{\boldsymbol{A}^\top \boldsymbol{u}^t}{\eta}) \ \odot \max \left\{ |\boldsymbol{\beta}^t + \frac{\boldsymbol{A}^\top \boldsymbol{u}^t}{\eta}| - \frac{1}{\eta}, 0 \right\}. \tag{13}$$

The second subproblem regarding $\boldsymbol{z}$ in (11)

$$\boldsymbol{z}^{t+1} \leftarrow \arg\min_{\boldsymbol{z}} \left\{ \delta_{\mathcal{Z}_0(\boldsymbol{z})} + \frac{\mu}{2} \|\boldsymbol{z} - \boldsymbol{z}^t + \frac{\boldsymbol{u}^t}{\mu}\|_2^2 \right\},$$

is similar to the $\boldsymbol{z}$ problem in the ADMM proposed by Lu et al. (2012), with the main difference being the quadratic augmented term. However, this distinction does not affect its closed-form solution property. Similar to Proposition 2.1 in Lu et al. (2012), the closed-form solution is as follows:

$$\boldsymbol{z}^{t+1} \leftarrow \min \left\{ \max \left\{ \boldsymbol{z}^t - \frac{\boldsymbol{u}^t}{\mu}, -n\lambda \right\}, n\lambda \right\}. \tag{14}$$

For the update of the dual variable $\boldsymbol{u}^{t+1}$, PPA includes an additional term ($\boldsymbol{A}\boldsymbol{\beta}^t - \boldsymbol{z}^t - \boldsymbol{X}^\top \boldsymbol{y}$) compared to the update of $\boldsymbol{u}^{t+1}$ in LADMM. However, please note that this additional term has already been computed during the update of $\boldsymbol{u}^t$, so it does not require any extra computation during the update process. The iterative steps of PPA algorithm are summarized in Algorithm 1. Although calculating the maximum eigenvalue of $\boldsymbol{A}^\top \boldsymbol{A}$ is necessary for determining the value of η, the power method in Liang et al. (2024) simplifies this by only requiring the input of $\boldsymbol{A}$. Consequently, the precomputation step in Algorithm 1 avoids the need to explicitly compute $\boldsymbol{A}^\top \boldsymbol{A}$, which can be computationally intensive, particularly when p is large.

Algorithm 1 PPA for DS

Input: $\boldsymbol{X}, \boldsymbol{y}, \mu, \lambda$ and $\boldsymbol{\beta}^0, \boldsymbol{z}^0$ and $\boldsymbol{u}^0$.
Pre-computation: $\boldsymbol{X}^\top \boldsymbol{y}$, $\boldsymbol{A} = \boldsymbol{X}^\top \boldsymbol{X}$ and $\eta = \text{eigen}(\mu \boldsymbol{A}^\top \boldsymbol{A}) + 1$.
Output: the total number of iterations T, $\boldsymbol{\beta}^T$.
while not converged **do**
    1. Update $\boldsymbol{\beta}^{t+1}$ using (13),
    2. Update $\boldsymbol{z}^{t+1}$ using (14),
    3. Update $\boldsymbol{u}^{t+1}$ using the last equation in (11),
    4. $t \leftarrow t + 1$.
end while
return solution.

It is easy to see that the updates for $\boldsymbol{\beta}^{t+1}$ and $\boldsymbol{z}^{t+1}$ in PPA have slightly lower computational complexity compared to LADMM. Specifically, in LADMM, the primary computational burdens for the updates of $\boldsymbol{\beta}^{t+1}$ and $\boldsymbol{z}^{t+1}$ arise from the need to compute $\boldsymbol{A}^\top \boldsymbol{A} \boldsymbol{\beta}^t$ and $\boldsymbol{X}^\top \boldsymbol{X} \boldsymbol{\beta}^{t+1}$, respectively. The $\boldsymbol{\beta}^{t+1}$ update in PPA only requires computing $\boldsymbol{A}^\top \boldsymbol{u}^t$, while the $\boldsymbol{z}^{t+1}$ update involves simple operations without the need for matrix-vector multiplication. The advantage of this computational complexity is not very pronounced in a non-parallel computing environment; however, in a parallel computing setting, PPA significantly saves on both storage and computational power compared to the parallel ADMM proposed by Wen et al. (2024).

3.2 Parallel version

To adapt to the parallel computing environment, we need to rewrite the constraint form of DS. Unlike Wen et al. (2024), which requires the introduction of an additional $(K-1)p$-dimensional auxiliary variable $\boldsymbol{\omega}$, our reformulation does not require the addition of any auxiliary variables. The constrained optimization form of DS solved by PPA in a parallel computing environment is as follows:

$$
\begin{aligned}
\min_{\boldsymbol{\beta}, \boldsymbol{z}} \quad & \sum_{i=1}^{K} \|\boldsymbol{\beta}_{i\cdot}\|_1 + \delta_{\mathcal{Z}_0(\boldsymbol{z})} \\
\text{s.t.} \quad & \sum_{i=1}^{K} \boldsymbol{A}_i \boldsymbol{\beta}_{i\cdot} - \boldsymbol{z} = \boldsymbol{X}^\top \boldsymbol{y},
\end{aligned}
\tag{15}
$$

where $\boldsymbol{A}_i = \boldsymbol{X}^\top \boldsymbol{X}_i$ for $i = 1, \dots, K$ and $(\boldsymbol{X}_1, \boldsymbol{X}_2, \dots, \boldsymbol{X}_K) = \boldsymbol{X}$. Since $\sum_{i=1}^{K} \|\boldsymbol{\beta}_{i\cdot}\|_1 = \|\boldsymbol{\beta}\|_1$ and $\sum_{i=1}^{K} \boldsymbol{A}_i \boldsymbol{\beta}_{i\cdot} = \boldsymbol{A} \boldsymbol{\beta}$, (15) and (10) are equal. This means that the convergence solutions of the DS obtained using iterative algorithms in both parallel and non-parallel computing environments are the same.

The Lagrange multiplier form of (15) is

$$
L(\boldsymbol{\beta}, \boldsymbol{z}; \boldsymbol{u}) = \sum_{i=1}^{K} \|\boldsymbol{\beta}_{i\cdot}\|_1 + \delta_{\mathcal{Z}_0(\boldsymbol{z})} - \boldsymbol{u}^\top \left(\sum_{i=1}^{K} \boldsymbol{A}_i \boldsymbol{\beta}_{i\cdot} - \boldsymbol{z} - \boldsymbol{X}^\top \boldsymbol{y} \right).
\tag{16}
$$

With given $(\boldsymbol{\beta}^0, \boldsymbol{z}^0, \boldsymbol{u}^0)$, the iterative scheme of the parallel PPA (with linearization) for problem (15) is as follows:

$$
\begin{aligned}
\boldsymbol{\beta}_{i\cdot}^{t+1} &\leftarrow \arg\min_{\boldsymbol{\beta}_{i\cdot}} \left\{ L(\boldsymbol{\beta}, \boldsymbol{z}^t; \boldsymbol{u}^t) + \frac{\mu}{2} \|\boldsymbol{A}_i(\boldsymbol{\beta}_{i\cdot} - \boldsymbol{\beta}_{i\cdot}^t)\|_2^2 + \frac{1}{2} \|\boldsymbol{\beta}_{i\cdot} - \boldsymbol{\beta}_{i\cdot}^t\|_{\boldsymbol{S}_i}^2 \right\}, i \in \{1, \cdots, K\}; \\
\boldsymbol{z}^{t+1} &\leftarrow \arg\min_{\boldsymbol{z}} \left\{ L(\boldsymbol{\beta}^{t+1}, \boldsymbol{z}; \boldsymbol{u}^t) + \frac{\mu}{2} \|\boldsymbol{z} - \boldsymbol{z}^t\|_2^2 \right\}; \\
\boldsymbol{u}^{t+1} &\leftarrow \boldsymbol{u}^t - \frac{\mu}{2} \left[2 \left(\sum_{i=1}^{K} \boldsymbol{A}_i \boldsymbol{\beta}_{i\cdot}^{t+1} - \boldsymbol{z}^{t+1} - \boldsymbol{X}^\top \boldsymbol{y} \right) - \left(\sum_{i=1}^{K} \boldsymbol{A}_i \boldsymbol{\beta}_{i\cdot}^t - \boldsymbol{z}^t - \boldsymbol{X}^\top \boldsymbol{y} \right) \right];
\end{aligned}
\tag{17}
$$

where $\mu > 0$ is a given augmented parameter, $\boldsymbol{S}_i = \eta \boldsymbol{I}_{p_i} - \mu \boldsymbol{A}_i^\top \boldsymbol{A}_i$, and η is larger than the maximum eigenvalue of $\mu \boldsymbol{A}^\top \boldsymbol{A}$. Note that η is always greater than the maximum eigenvalue of $\mu \boldsymbol{A}_i^\top \boldsymbol{A}_i$ because $\boldsymbol{A}_i$ is a submatrix of $\boldsymbol{A}$. This ensures the positive definiteness of $\boldsymbol{S}_i$. It is straightforward to verify that when $K = 1$, $\boldsymbol{S}_1 = \boldsymbol{S}$ holds, thereby reducing the parallel PPA into Algorithm 1.

For the $\boldsymbol{\beta}_{i\cdot}$ subproblem of the parallel PPA, by organizing the optimized form of $\boldsymbol{\beta}_{i\cdot}$, we can conclude that

$$\boldsymbol{\beta}_{i\cdot}^{t+1} \leftarrow \underset{\boldsymbol{\beta}_{i\cdot}}{\arg\min} \left\{ \|\boldsymbol{\beta}_{i\cdot}\|_1 + \frac{\eta}{2} \|\boldsymbol{\beta}_{i\cdot} - \boldsymbol{\beta}_{i\cdot}^t - \frac{\boldsymbol{A}_i^\top \boldsymbol{u}^t}{\eta} \|_2^2 \right\},$$

which has the closed-form solution

$$\boldsymbol{\beta}_{i\cdot}^{t+1} \leftarrow \text{sign}(\boldsymbol{\beta}_{i\cdot}^t + \frac{\boldsymbol{A}_i^\top \boldsymbol{u}^t}{\eta}) \odot \max \left\{ |\boldsymbol{\beta}_{i\cdot}^t + \frac{\boldsymbol{A}_i^\top \boldsymbol{u}^t}{\eta}| - \frac{1}{\eta}, 0 \right\}. \tag{18}$$

It is clear that the updates of different $\boldsymbol{\beta}_{i\cdot}^{t+1}$ are independent of each other and can be implemented in parallel. Note that in the process of updating $\boldsymbol{\beta}_{i\cdot}^{t+1}$, we only need to compute $\boldsymbol{A}_i^\top \boldsymbol{u}^t$, unlike in the updates of $\boldsymbol{\beta}_{i\cdot}^{t+1}$ in parallel ADMM in Section 2.2, which require multiple matrix-vector multiplications. This also makes our parallel algorithm more intuitive and concise compared to Wen et al. (2024), which introduces an auxiliary variable $\boldsymbol{\omega}$ to construct the parallel structure.

The updates for parallel versions of $\boldsymbol{z}^{t+1}$ and $\boldsymbol{u}^{t+1}$ are the same as those for non parallel version. Similarly, due to the simpler construction of our parallel structure, the intermediate iterations in our approach are also more concise compared to the updates of $\boldsymbol{z}^{t+1}$ and $\boldsymbol{u}_i^{t+1}$ in parallel ADMM. In addition, our dual variable consists of only one p-dimensional $\boldsymbol{u}^{t+1}$, unlike parallel ADMM, which introduces multiple constraints for the parallel structure and requires iterating over K p-dimensional vectors $\boldsymbol{u}_i^{t+1}$.

Algorithm 2 Parallel PPA for DS

Input: $\boldsymbol{X}, \boldsymbol{y}, \mu, \lambda, \boldsymbol{\beta}^0, \boldsymbol{z}^0$ and $\boldsymbol{u}^0$.
Pre-computation: $\boldsymbol{X}^\top \boldsymbol{y}$, $\boldsymbol{A}_i = \boldsymbol{X}^\top \boldsymbol{X}_i$ and $\eta = \text{eigen}(\mu \boldsymbol{A}^\top \boldsymbol{A}) + 1$.
Output: the total number of iterations T, $\boldsymbol{\beta}^T$.
while not converged **do**
    1. Update $\boldsymbol{\beta}_{i\cdot}^{t+1}$ in parallel using (18),
    2. Update $\boldsymbol{z}^{k+1}$ using (14),
    3. Update $\boldsymbol{u}^{k+1}$ using the last equation in (17),
    4. $t \leftarrow t + 1$.
end while
return solution.

We summarize the process of parallel PPA in Algorithm 2. As discussed earlier, when $K = 1$, Algorithm 2 reduces to Algorithm 1. Note that in the precomputation of Algorithm 2, we only need to compute the submatrices $\boldsymbol{A}_i$ of $\boldsymbol{A}$, rather than the entire matrix $\boldsymbol{A}$, because the updates for $\boldsymbol{\beta}$, $\boldsymbol{z}$, and $\boldsymbol{u}$ do not require $\boldsymbol{A}$. However,

the power method (Liang et al. (2024)) requires $\boldsymbol{A}$ to calculate η. To avoid additional computational burden, we can combine the individual $\boldsymbol{A}_i$ into $\boldsymbol{A}$ and substitute it into the power method to obtain eigen$(\mu\boldsymbol{A}^\top\boldsymbol{A})$.

From the above discussion, it can be seen that Algorithm 1 and Algorithm 2 ($K \geq 2$) have the same variable updates, except for the update of $\boldsymbol{\beta}^{t+1}$. Therefore, is there any relationship or connection between the solutions of these two algorithms? Through some simple matrix decompositions, we can prove that the solutions of Algorithm 1 and Algorithm 2 are the same. We present this interesting conclusion in Theorem 1. For ease of distinction, let us denote $\left\{\hat{\boldsymbol{\beta}}^t, \hat{\boldsymbol{z}}^t, \hat{\boldsymbol{u}}^t\right\}$ as the t-th iteration results of Algorithm 1, and $\left\{\tilde{\boldsymbol{\beta}}^t, \tilde{\boldsymbol{z}}^t, \tilde{\boldsymbol{u}}^t\right\}$ as the t-th iteration results of Algorithm 2.

Theorem 1. *(Partition insensitivity). If we use the same initial iteration variables* $\{\hat{\boldsymbol{\beta}}^0, \hat{\boldsymbol{z}}^0, \hat{\boldsymbol{u}}^0\} = \{\tilde{\boldsymbol{\beta}}^0, \tilde{\boldsymbol{z}}^0, \tilde{\boldsymbol{u}}^0\}$, *the iterative solutions obtained from Algorithm 1 and Algorithm 2 (for any K) are indeed the same, that is,*

$$\left\{\hat{\boldsymbol{\beta}}^t, \hat{\boldsymbol{z}}^t, \hat{\boldsymbol{u}}^t\right\} = \left\{\tilde{\boldsymbol{\beta}}^t, \tilde{\boldsymbol{z}}^t, \tilde{\boldsymbol{u}}^t\right\}, \quad \text{for all } t. \tag{19}$$

Although the conclusion of Theorem 1 is surprising, its proof is quite straightforward. We only need to partition the update equation for $\boldsymbol{\beta}^{t+1}$ in (13) based on the columns of $\boldsymbol{A}$ into K parts, which results in the update equation in (18). In fact, the constrained optimization problem in Algorithm 2 involves $K + 1$ primal variables ($\{\boldsymbol{\beta}_{i\cdot}\}_{i=1}^K$ and $\boldsymbol{z}$). The special structure of the proposed PPA algorithm allows the updates of the variables in Algorithm 2 to be independent. Thus, $\{\boldsymbol{\beta}_{i\cdot}\}_{i=1}^K$ can be concatenated and treated as a single primal variable $\boldsymbol{\beta}$. This is also the underlying reason for the validity of the conclusion of Theorem 1. Moreover, we provide a rigorous mathematical derivation of Theorem 1 in Appendix A.1.

Theorem 1 states that regardless of how $\boldsymbol{A}$ is partitioned by columns, the solution of Algorithm 2 remains unchanged and is the same as the solution of Algorithm 1. In Wu et al. (2023), this property is referred to as partition insensitivity. However, in the work of Wu et al. (2023), their partitioning is done by rows, while in this paper, the partitioning is done by columns. The insensitivity of partition ensures the reliability and accuracy of solutions in a parallel computing environment, regardless of the chosen parallelization strategy. It allows for a more flexible adjustment of computing resources to meet varying computational needs and hardware configurations. The insensitivity of the partition is not observed in the parallel ADMM discussed in Wen et al. (2024), as the nature of the constraints managed by the algorithm varies with K.

Based on the conclusion of Theorem 1, it is evident that the solutions obtained by the two algorithms are the same. Therefore, when discussing convergence, it suffices to consider only one of the algorithms. Next, we will provide the global convergence of two PPAs and their linear convergence rate. The proof of the theorem is attached in Appendix B.2.

Theorem 2. *Let the sequence $\{\boldsymbol{g}^t = (\boldsymbol{\beta}^t, \boldsymbol{z}^t, \boldsymbol{u}^t)\}$ be generated by Algorithm 1 or Algorithm 2.*

14

1. *(Algorithm global convergence). It converges to some $\boldsymbol{g}^* = (\boldsymbol{\beta}^*, \boldsymbol{z}^*, \boldsymbol{u}^*)$ that belongs to Ω^*, where Ω^* denotes the set of saddle points of ([16](#)) and is assumed to be non-empty.*

2. *(Linear convergence rate). For any integer $T > 0$, we have*

$$\|\boldsymbol{g}^T - \boldsymbol{g}^{T+1}\|_{\boldsymbol{H}}^2 \leq \frac{1}{(T+1)} \|\boldsymbol{g}^0 - \boldsymbol{g}^*\|_{\boldsymbol{H}}^2, \tag{20}$$

where $\boldsymbol{H}$ is a positive definite matrix, and can be found in ([B11](#)).

Remark 1. *It is not difficult to see that ([16](#)) is convex, and the component $(\boldsymbol{\beta}^*, \boldsymbol{z}^*)$ of its saddle point $(\boldsymbol{\beta}^*, \boldsymbol{z}^*, \boldsymbol{u}^*)$ corresponds to the optimal solution for ([15](#)).*

Remark 2. *It is clear that $\|\boldsymbol{g}^0 - \boldsymbol{g}^*\|_{\boldsymbol{H}}^2$ is a positive constant. Therefore, $\|\boldsymbol{g}^T - \boldsymbol{g}^{T+1}\|_{\boldsymbol{H}}^2 \leq \mathcal{O}\left(\frac{1}{T}\right)$, which is known as a linear convergence rate. In addition, during the proof of this theorem, we demonstrated that $\|\boldsymbol{g}^t - \boldsymbol{g}^*\|_{\boldsymbol{H}}^2$ and $\|\boldsymbol{g}^t - \boldsymbol{g}^{t+1}\|_{\boldsymbol{H}}^2$ are monotonically non-increasing, that is, $\|\boldsymbol{g}^{t+1} - \boldsymbol{g}^*\|_{\boldsymbol{H}}^2 \leq \|\boldsymbol{g}^t - \boldsymbol{g}^*\|_{\boldsymbol{H}}^2$ and $\|\boldsymbol{g}^{t+1} - \boldsymbol{g}^{t+2}\|_{\boldsymbol{H}}^2 \leq \|\boldsymbol{g}^t - \boldsymbol{g}^{t+1}\|_{\boldsymbol{H}}^2$, see Proposition 2 and Proposition 4 in the Appendix.*

3.3 Improved parallel version

In Algorithm 2, we need to concatenate $\boldsymbol{A}_i$ into a complete matrix $\boldsymbol{A}$ during the linearization step and then use the power method to compute $\mathrm{eigen}(\mu \boldsymbol{A}^\top \boldsymbol{A})$. When p is excessively large, concatenating all $\boldsymbol{A}_i$ into a single matrix $\boldsymbol{A}$ and applying the power method becomes impractical. To enable the parallel PPA to handle such situations, we will develop the following improved version of PPA.

With given $(\boldsymbol{\beta}^0, \boldsymbol{z}^0, \boldsymbol{u}^0)$, the iterative scheme of the improved parallel PPA (with linearization) for problem ([15](#)) is as follows:

$$\boldsymbol{\beta}_{i\cdot}^{t+1} \leftarrow \underset{\boldsymbol{\beta}_{i\cdot}}{\arg\min} \left\{ L(\boldsymbol{\beta}, \boldsymbol{z}^t; \boldsymbol{u}^t) + \frac{\mu}{2} \|\boldsymbol{A}_i(\boldsymbol{\beta}_{i\cdot} - \boldsymbol{\beta}_{i\cdot}^t)\|_2^2 + \frac{1}{2} \|\boldsymbol{\beta}_{i\cdot} - \boldsymbol{\beta}_{i\cdot}^t\|_{\boldsymbol{S}_i'}^2 \right\}, i \in \{1, \cdots, K\};$$

$$\boldsymbol{z}^{t+1} \leftarrow \underset{\boldsymbol{z}}{\arg\min} \left\{ L(\boldsymbol{\beta}^{t+1}, \boldsymbol{z}; \boldsymbol{u}^t) + \frac{\mu}{2} \|\boldsymbol{z} - \boldsymbol{z}^t\|_2^2 \right\}; \tag{21}$$

$$\boldsymbol{u}^{t+1} \leftarrow \boldsymbol{u}^t - \frac{\mu}{K+1} \left[2(\sum_{i=1}^K \boldsymbol{A}_i \boldsymbol{\beta}_{i\cdot}^{t+1} - \boldsymbol{z}^{t+1} - \boldsymbol{X}^\top \boldsymbol{y}) - (\sum_{i=1}^K \boldsymbol{A}_i \boldsymbol{\beta}_{i\cdot}^t - \boldsymbol{z}^t - \boldsymbol{X}^\top \boldsymbol{y}) \right];$$

where $\boldsymbol{S}_i' = \eta_i \boldsymbol{I}_{p_i} - \mu \boldsymbol{A}_i^\top \boldsymbol{A}_i$, and η_i is larger than $\mathrm{eigen}(\mu \boldsymbol{A}_i^\top \boldsymbol{A}_i)$. It is straightforward to verify that when $K = 1$, $\boldsymbol{S}_1' = \boldsymbol{S}_1 = \boldsymbol{S}$, thereby reducing the improved parallel PPA into Algorithm 1 and Algorithm 2 ($K = 1$).

The closed-form solution of $\boldsymbol{\beta}_{i\cdot}^{t+1}$ can be obtained by

$$\boldsymbol{\beta}_{i\cdot}^{t+1} \leftarrow \mathrm{sign}(\boldsymbol{\beta}_{i\cdot}^t + \frac{\boldsymbol{A}_i^\top \boldsymbol{u}^t}{\eta_i}) \odot \max\left\{ |\boldsymbol{\beta}_{i\cdot}^t + \frac{\boldsymbol{A}_i^\top \boldsymbol{u}^t}{\eta_i}| - \frac{1}{\eta_i}, 0 \right\}. \tag{22}$$

The only difference between ([22](#)) and ([18](#)) is that η has been replaced with η_i. The update for $\boldsymbol{z}^t$ remains unchanged compared to Algorithm 1 and Algorithm 2. However,

for the update of $\boldsymbol{u}^{t+1}$, a slight modification is needed to ensure the linear convergence of the algorithm, specifically changing $\frac{\mu}{2}$ in Algorithm 2 to $\frac{\mu}{K+1}$. This adjustment is necessary because we use K instances of η_i to approximate the entire η. This introduces additional perturbations in the $\boldsymbol{\beta}_i$. subproblem, necessitating adjustments to certain constants in the $\boldsymbol{u}$ update process. It is worth noting that this change may not improve the computational efficiency of Algorithm 2. On the contrary, due to the approximation of η, more iterations may be required at times. The so-called improvement merely makes the parallel PPA applicable in extreme cases where p is so large that the memory required to store matrix $\boldsymbol{A}^\top \boldsymbol{A}$ and compute η exceeds the available memory of the machine. We summarize improved parallel PPA in Algorithm 3.

Algorithm 3 does not exhibit the partition insensitivity (refer to Theorem 1) that is characteristic of Algorithm 2. Nevertheless, the iterative convergence solution of Algorithm 3 is identical to that of Algorithm 2, and Algorithm 3 maintains the same linear convergence rate as Algorithm 2. Next, we utilize the following theorem to illustrate that.

Theorem 3. *Let the sequence $\{\boldsymbol{g}^t = (\boldsymbol{\beta}^t, \boldsymbol{z}^t, \boldsymbol{u}^t)\}$ be generated by Algorithm 3.*

1. *(Algorithm global convergence). It converges to some $\boldsymbol{g}^* = (\boldsymbol{\beta}^*, \boldsymbol{z}^*, \boldsymbol{u}^*)$ that belongs to Ω^*, where Ω^* denotes the set of saddle points of (16) and is assumed to be non-empty.*

2. *(Linear convergence rate). For any integer $T > 0$, we have*

$$\|\boldsymbol{g}^T - \boldsymbol{g}^{T+1}\|_{\boldsymbol{H_K}}^2 \leq \frac{1}{(T+1)}\|\boldsymbol{g}^0 - \boldsymbol{g}^*\|_{\boldsymbol{H_K}}^2, \tag{23}$$

where $\boldsymbol{H_K}$ is a positive definite matrix , and can be found in (B12).

Algorithm 3 Improved Parallel PPA for DS

Input: $\boldsymbol{X}, \boldsymbol{y}, \mu, \lambda, \boldsymbol{\beta}^0, \boldsymbol{z}^0$ and $\boldsymbol{u}^0$.
Pre-computation: $\boldsymbol{X}^\top \boldsymbol{y}$, $\boldsymbol{A}_i = \boldsymbol{X}^\top \boldsymbol{X}_i$ and $\eta_i = \text{eigen}(\mu \boldsymbol{A}_i^\top \boldsymbol{A}_i) + 1$.
Output: the total number of iterations T, $\boldsymbol{\beta}^T$.
while not converged **do**
      1. Update $\boldsymbol{\beta}^{k+1}$ using (22),
      2. Update $\boldsymbol{z}^{k+1}$ using (14),
      3. Update $\boldsymbol{u}^{k+1}$ using the last equation in (21),
      4. $t \leftarrow t + 1$
end while
return solution.

The conclusion and proof of Theorem 3 closely mirror those of Theorem 2, with the primary distinction being the substitution of the $\boldsymbol{H}$ matrix with $\boldsymbol{H_K}$. A detailed proof is provided in Appendix B.2. The iterative convergence solutions of Algorithm 1, Algorithm 2, and Algorithm 3 are identical because all three algorithms handle the same form of equality constraint optimization in (15). This property is not present in the parallel ADMM discussed in Wen et al. (2024), as the form of constraints

handled by the algorithm changes with increasing K. The linear convergence rate of Algorithm 3 is $\|\boldsymbol{g}^T - \boldsymbol{g}^{T+1}\|^2_{\boldsymbol{H_K}} \leq \mathcal{O}\left(\frac{1}{T}\right)$, and its iterative sequences $\|\boldsymbol{g}^t - \boldsymbol{g}^*\|^2_{\boldsymbol{H_K}}$ and $\|\boldsymbol{g}^t - \boldsymbol{g}^{t+1}\|^2_{\boldsymbol{H_K}}$ also monotonically do not increase.

Understanding the computational resource requirements of the algorithm for data of different dimensions is crucial for assessing its feasibility in practical applications. Next, we conduct an analysis of the time and space complexity of the algorithms. Without loss of generality, assume the columns of matrix $\boldsymbol{A}$ are evenly split into K parts. We use the following theorem to clarify these two complexities, and the proof of the theorem is provided in Appendix A.2.

Theorem 4. *The total time complexities of Algorithm 2 and Algorithm 3 are given by*

$$
\begin{cases}
\left[\mathcal{O}(\frac{np^2}{K}) + \mathcal{O}(p^2)\right] + \mathcal{O}(\frac{p^2}{K}) \times T & \text{Algorithm 2}, \\
\mathcal{O}(\frac{np^2}{K}) + \mathcal{O}(\frac{p^2}{K}) \times T & \text{Algorithm 3},
\end{cases}
\tag{24}
$$

where $\mathcal{O}$ depicts the asymptotic upper bound of algorithm complexity (time or space) as the input size expands, and T denotes the total number of iterations in the parallel PPA algorithms. The total space complexities of Algorithm 2 and Algorithm 3 are given by

$$
\begin{cases}
\mathcal{O}(np) + \mathcal{O}(p^2) & \text{Algorithm 2}, \\
\mathcal{O}(np) + \mathcal{O}(\frac{p^2}{K}) & \text{Algorithm 3}.
\end{cases}
\tag{25}
$$

We do not discuss Algorithm 1 because when $K = 1$, both Algorithm 2 and Algorithm 3 will degenerate into Algorithm 1. The main difference in the computational time complexity between Algorithm 2 and Algorithm 3 lies in the preceding pre-computation process. When $n \geq K$, their complexities are the same. When $n < K$, the complexity of Algorithm 2 is higher than that of Algorithm 3. It is readily apparent that an increase in K will substantially reduce the computational time complexity of the two parallel algorithms. It should be noted that when the matrix is divided unevenly by columns, the p/K in the computational time complexity needs to be replaced with $\max\{p_k\}_{k=1}^K$, as referred to in Wu et al. (2025a). The conclusion regarding space storage in (25) also demonstrates that Algorithm 3 can address the drawbacks of Algorithm 2 when the required storage space for matrix $\boldsymbol{A}$ and η is too large. This is because it divides the square of p by K. In other words, increasing the value of K can effectively alleviate the excessive storage burden.

3.4 Nonconvex extension

In high-dimensional linear regression, convex regularization terms such as Lasso ensure global optimality and computational efficiency. In contrast, nonconvex regularization terms may offer better estimation and prediction performance but present computational challenges due to the lack of global optimality. Recently, in the context of DS models with nonconvex penalties, Wen et al. (2024) indicated that these nonconvex DSs can be uniformly solved by combining local linear approximation (LLA, Zou and Li (2008)) methods with an effective solution for weighted Lasso penalty.

In this paper, we mainly consider two popular nonconvex regularizers, SCAD penalty (Fan and Li (2001)) and MCP penalty (Zhang (2010)) in statistics. According to the suggestion in Zou and Li (2008)), we can use a unified method named local linear approximation (LLA) to handle the nonconvex penalty, that is

$$P_{a,\lambda}(|\boldsymbol{\beta}|) \approx P_{a,\lambda}(|\boldsymbol{\beta}^l|) + \nabla P_{a,\lambda}(|\boldsymbol{\beta}^l|)^T (|\boldsymbol{\beta}| - |\boldsymbol{\beta}^l|), \text{ for } \boldsymbol{\beta} \approx \boldsymbol{\beta}^l, \qquad (26)$$

where a is a given constant, $\boldsymbol{\beta}^l$ is the solution from the last iteration, and

$$\nabla P_{a,\lambda}(|\boldsymbol{\beta}^l|) = (\nabla P_{a,\lambda}(|\boldsymbol{\beta}^l_1|), \nabla P_{a,\lambda}(|\boldsymbol{\beta}^l_2|), \dots, \nabla P_{a,\lambda}(|\boldsymbol{\beta}^l_p|))^\top.$$

- For SCAD, we have

$$\nabla P_{a,\lambda}(|\boldsymbol{\beta}_j|) = \begin{cases} \lambda, & \text{if } |\boldsymbol{\beta}_j| \leq \lambda, \\ \frac{a\lambda - |\boldsymbol{\beta}_j|}{a-1}, & \text{if } \lambda < |\boldsymbol{\beta}_j| < a\lambda, \\ 0, & \text{if } |\boldsymbol{\beta}_j| \geq a\lambda. \end{cases} \qquad (27)$$

- For MCP, we have

$$\nabla P_{a,\lambda}(|\boldsymbol{\beta}_j|) = \begin{cases} \lambda - \frac{|\boldsymbol{\beta}_j|}{a}, & \text{if } |\boldsymbol{\beta}_j| \leq a\lambda, \\ 0, & \text{if } |\boldsymbol{\beta}_j| > a\lambda. \end{cases} \qquad (28)$$

The nonconvex DS in Wen et al. (2024) can be written as

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^p} \quad \sum_{j=1}^p P_{a,\lambda}(|\boldsymbol{\beta}_j|)/\lambda \text{ s.t. } |\boldsymbol{X}_j^\top (\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y})/n| \leq \nabla P_{a,\lambda}(|\boldsymbol{\beta}_j|), j = \{1, \dots, p\}, \quad (29)$$

where $P_{a,\lambda}(|\boldsymbol{\beta}|) = \text{SCAD}(\boldsymbol{\beta})$ or $\text{MCP}(\boldsymbol{\beta})$. By substituting equation (26) into equation (29), we can obtain the following optimized form in a weighted manner,

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^p} \sum_{j=1}^p \left[\frac{\nabla P_{a,\lambda}(|\boldsymbol{\beta}^l_j|)}{\lambda} |\boldsymbol{\beta}_j| \right] \text{ s.t. } |\boldsymbol{X}^\top (\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y})/n| \leq \nabla P_{a,\lambda}(|\boldsymbol{\beta}^l_j|), j = \{1, \dots, p\}.$$
$$(30)$$

Note that we only need to make two small changes to solve this weighted optimization form using Algorithms 1, 2 and 3. The first change requires replacing $\|\boldsymbol{\beta}\|_1$ (or $\sum_{j=1}^p |\boldsymbol{\beta}_j|$) in (1) with $\sum_{j=1}^p \left[\frac{\nabla P_{a,\lambda}(|\boldsymbol{\beta}^l_j|)}{\lambda} |\boldsymbol{\beta}_j| \right]$ in (30). In other words, substitute the equal weight 1 following the $\sum$ with $\nabla P_{a,\lambda}(|\boldsymbol{\beta}^l_j|)/\lambda$. The second change is in the form of the constraint, where the ℓ_∞-norm is replaced with p weighted ℓ_1-norm.

To solve nonconvex DS using the LLA algorithm, it is necessary to find a good initial value. As suggested by Wen et al. (2024), we can use the solution of $P_\lambda(|\boldsymbol{\beta}|) = \lambda\|\boldsymbol{\beta}\|_1$ in (29) as the initial value. Then, we get the solution of (29) by solving a

sequence of weighted Lasso DS. We summarize the detailed iterative steps of this method in Algorithm 4.

Algorithm 4 PPA for nonconvex DS

---

1. Initialize $\boldsymbol{\beta}$ with $\boldsymbol{\beta}^1$, where $\boldsymbol{\beta}^1$ is obtained by Algorithms 1, 2 or 3.

2. For $l = 1, 2, \ldots, L$, continue iterating the LLA iteration until convergence is achieved.

   2.1. Compute the weights $\nabla P_\lambda(|\boldsymbol{\beta}^l|) = (\nabla P_\lambda(|\boldsymbol{\beta}_1^l|), \nabla P_\lambda(|\boldsymbol{\beta}_2^l|), \ldots, \nabla P_\lambda(|\boldsymbol{\beta}_p^l|))^\top$ by (27) or (28).

   2.2. Solve weighted ℓ_1 DS. For $t = 0, \ldots, T$,

      2.2.1. $\boldsymbol{\beta}^{l,t+1}$-subproblem: solve the weighted ℓ_1 problem in (13), (18) and (22) by replacing 1 with $\nabla P_{a,\lambda}(|\boldsymbol{\beta}_j^l|)/\lambda$,

      2.2.2. $\boldsymbol{z}^{l,t+1}$-subproblem: $\boldsymbol{z}_j^{l,t+1} \leftarrow \min\left\{\max\left\{\boldsymbol{z}_j^{l,t} - \frac{\boldsymbol{u}_j^t}{\mu}, -n\nabla P_\lambda(|\boldsymbol{\beta}_j^l|)\right\}, n\nabla P_\lambda(|\boldsymbol{\beta}_j^l|)\right\}$,

$j \in \{1, \ldots, p\}$,

      2.2.3. $\boldsymbol{u}^{l,t+1}$-subproblem: Update $\boldsymbol{u}^{l,t+1}$ according to the corresponding algorithm (Algorithm 1, Algorithm 2 or Algorithm 3).

      2.3. Let $\boldsymbol{\beta}^{l+1} = \boldsymbol{\beta}^{l,T}$.

Below, we will describe in detail the specific implementation of 2.2.1 in Algorithm 4. If we use Algorithm 1 and 2 to solve nonconvex DS, we need to replace $1/\eta$ in (13) or (18) with $\nabla P_{a,\lambda}(|\boldsymbol{\beta}_j^l|)/(\lambda\eta)$. If we use Algorithm 3 to solve nonconvex DS, we need to replace $1/\eta_i$ in (22) with $\nabla P_{a,\lambda}(|\boldsymbol{\beta}_j^l|)/(\lambda\eta_i)$. The above discussion indicates that the nonconvex DS is solved through multiple iterations of weighted Lasso penalized DS. Moreover, Wen et al. (2024) demonstrated that, theoretically, only two iterations are sufficient to obtain a solution of (29) with high statistical accuracy. In the specific implementation of Algorithm 4, we also implement the warm-start technique (Friedman et al. (2010)), which uses the current solution $\boldsymbol{\beta}^l$ as the initial value for the next solution $\boldsymbol{\beta}^{l+1}$. This method significantly reduces the number of iterations needed in step 2.2 of Algorithm 4, often achieving convergence in just a few steps.

Next, we will discuss the convergence of Algorithm 4. As per Theorem 2.2 in Wen et al. (2024), a high-precision statistical estimator can be achieved when the outer loop is repeated twice, i.e., $L = 2$. Therefore, it is only necessary to discuss the convergence of step 2.2 of the inner loop solution for weighted ℓ_1 DS. Since the weighted form does not change the convexity of the objective function, we can draw the following corollary based on the conclusions of Theorem 2 and Theorem 3. The reasons justifying the validity of the corollary have been presented in Appendix B.2.

Corollary 3.1. *Let the sequence $\{\boldsymbol{g}^t = (\boldsymbol{\beta}^t, \boldsymbol{z}^t, \boldsymbol{u}^t)\}$ be generated by step 2.2 in Algorithm 4.*

1. *(Algorithm global convergence). It converges to some $\boldsymbol{g}^* = (\boldsymbol{\beta}^*, \boldsymbol{z}^*, \boldsymbol{u}^*)$ that belongs to $\tilde{\Omega}^*$, where $\tilde{\Omega}^*$ denotes the set of saddle points of weighted ℓ_1 DS with current weight, and is assumed to be non-empty.*

2. *(Linear convergence rate). For any integer $T > 0$, we have*

$$\|\boldsymbol{g}^T - \boldsymbol{g}^{T+1}\|^2_{\boldsymbol{H}_*} \leq \frac{1}{(T+1)}\|\boldsymbol{g}^0 - \boldsymbol{g}^*\|^2_{\boldsymbol{H}_*}, \tag{31}$$

where $\boldsymbol{H}_* = \boldsymbol{H}$ for Algorithm 1 and Algorithm 2, and $\boldsymbol{H}_* = \boldsymbol{H}_K$ for Algorithm 3.

4 Synthetic numerical simulations

In this section, we use synthetic datasets to demonstrate the accuracy, stability, and scalability of the proposed parallel PPA algorithms. All experiments in this paper were performed using R on a computer equipped with an AMD Ryzen 9 7950X 16-Core Processor running at 4.50 GHz and with 32 GB RAM. For the selection of the tuning parameter λ, we adopt the modified "HBIC" criteria suggested by Fan et al. (2021). We terminate all iterative algorithms as described in Lu et al. (2012) when

$$\frac{\|\boldsymbol{\beta}^{t+1} - \boldsymbol{\beta}^t\|_2}{\max\{\|\boldsymbol{\beta}^{t+1}\|_2, 1\}} \leq 1 \times 10^{-4},$$

or if the number of iterations exceeds 500.

4.1 Nonparallel environment

In this subsection, we first evaluate the effectiveness and stability of our proposed algorithm in a nonparallel environment ($K = 1$). As Wen et al. (2024), we construct the design matrix $\boldsymbol{X}$ by sampling each row from a multivariate normal distribution with a mean of zero and a covariance matrix $\boldsymbol{\Sigma} = (\rho_{j,j'})_{p \times p}$. The response y_i is generated according to the linear regression model

$$\mathbf{y} = \boldsymbol{X}\boldsymbol{\beta}^* + \varepsilon,$$

where $\boldsymbol{X} = (\boldsymbol{x}_1^\top, \ldots, \boldsymbol{x}_n^\top)^\top$, $\mathbf{x}_i \sim \mathcal{N}(0, \boldsymbol{\Sigma})$ and $\varepsilon_i \sim \mathcal{N}(0, 1)$ for $i \in \{1, \ldots, n\}$. Furthermore, we consider the correlation structure defined as $\rho_{j,j'} = \rho^{|j-j'|}$ for indices j and j' belonging to the set $\{1, \ldots, p\}$. Our analysis concentrates on three distinct correlation levels: $\rho \in \{0.1, 0.5, 0.9\}$. To generate the true coefficient vector $\boldsymbol{\beta}^*$, we randomly select a subset $\mathcal{A}$ from the set $\{1, \ldots, p\}$ such that the cardinality of $\mathcal{A}$ is equal to 8, that is $|\mathcal{A}| = 8$. Next, we assign the values 3, 1.5, 10, 4, 2, 5, 2.5, and 4.5 without replacement to $\boldsymbol{\beta}_j^*$ for $j \in \mathcal{A}$, and set $\boldsymbol{\beta}_j^* = 0$ for $j \notin \mathcal{A}$. Here, we consider two different combinations of sample size and data dimensions: $(n, p) = (500, 1000)$ and $(n, p) = (1000, 10,000)$. For larger scale numerical experiments, we conduct them in parallel computing environments. All initial values in the algorithm are set to a small constant, such as 0.001.

The simulation study evaluates and contrasts the performance of the following algorithms on solving ℓ_1-DS: ADMM (Lu et al. (2012)), FSM (Fast splitting method, He et al. (2015)), CPPA-PD (customized proximal point algorithm, He and Xu (2017)), partially proximal linearized alternating minimization method (P-PLAM, Mao et al.

(2021)), three blocks ADMM (TADMM, Wen et al. (2024)). Note that in a nonparallel environment, the TADMM used by Wen et al. (2024) is the same as the LADMM used by Wang and Yuan (2012), so we only compared the latest TADMM. In addition, the three PPA algorithms proposed in this paper are the same in nonparallel environments. There are three scenarios for CPPA-PD. In the first scenario, the parameters $s = \log p/5$, $r = 1.2\|\boldsymbol{A}^\top\boldsymbol{A}\|/s$, and $\tau = 1.2$ are adopted. This variant is denoted as CPPA-PD1, which is also the set of parameters used in Section 3.1 of He and Xu (2017). In the second scenario, the parameters $s = \log p/5$, $r = 1.2\|\boldsymbol{A}^\top\boldsymbol{A}\|/s$, and $\tau = 1$ are employed, and it is labeled as CPPA-PD2. In the third scenario, the parameters $s = 1$, $r = \|\boldsymbol{A}^\top\boldsymbol{A}\|$, and $\tau = 1$ are utilized, and it is designated as CPPA-PD3. The performance of the mentioned methods is examined under seven evaluation criteria: (1) the ℓ_1-error, expressed by $\|\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*\|_1$; (2) the ℓ_2-error, quantified as $\|\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*\|_2^2$; (3) the model error, calculated via $(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*)^\top\boldsymbol{\Sigma}(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*)$; (4) false positives (FP), representing the quantity of non-significant features chosen; (5) false negatives (FN), indicating the number of significant features omitted; (6) number of iterations (NI); and (7) CPU runtime (Time), measured in seconds. Metrics (1)–(3) serve to gauge estimation precision, metrics (4) and (5) are used to assess feature selection reliability, and metrics (6) and (7) evaluate computational efficiency. The simulation outcomes, based on 100 replicates, are presented in Table 1 and Table 2. The numbers in the table represent the mean of one hundred repeated experiments, and the standard deviation is indicated in parentheses. Because the values of FN are all 0, they are not presented in Table 1 and Table 2. For numerical experiments on nonconvex DS, such as SCAD-DS and MCP-DS, we have included them in the Appendix C.1.

As evidenced by Table 1 and Table 2, our proposed PPA method demonstrates significant superiority over other approaches in solving ℓ_1-DS problem, both in terms of estimation accuracy and feature selection effectiveness. Notably, the number of false positives (FP) is substantially smaller, indicating that the PPA algorithm proposed is highly selective, minimizing the probability of incorrectly identifying zero coefficients as non-zero. In terms of number of iterations (NI), FSM demands the least, with ADMM following closely behind, primarily due to its avoidance of linearization steps. Linearization, despite simplifying subproblem resolution, merely offers approximate solutions, necessitating additional iterations. As He et al. (2020) observed, increased η values lead to slower convergence and extended iteration durations. When considering CPU runtime (Time), P-PLAM emerges as the top performer, PPA comes next, while ADMM and TADMM lag behind. The reason why P-PLAM requires the least computation time is that its optimization formulation is unconstrained, obviating the need to update dual variables. As a result, it boasts the minimum number of iterative variables, thereby achieving the fastest computation speed. PPA follows closely in terms of computation time because it also involves a manageable number of iterative variables, and each iteration is computationally straightforward. ADMM underperforms due to the embedded double loop, and TADMM has an extended duration attributed to its large NI.

A reviewer pointed out that in Table 1 and Table 2, the numerical performance of PPA proposed in this paper is better than CPPA-PD1, but both algorithms belong to proximal point algorithm and have high similarity. Therefore, it is necessary to

Table 1 Comparison of various algorithms for solving ℓ_1-DS in nonparallel environments with data scale $(n, p) = (500, 1000)$.

Method	ρ	ℓ_1 error	ℓ_2 error	Model error	FP	NI	Time(s)
ADMM	0.1	5.65(0.231)	1.23(0.047)	1.15(0.039)	19.78(2.03)	154.3(14.1)	10.76(0.78)
	0.5	6.11(0.305)	1.47(0.050)	1.21(0.042)	22.50(2.17)	159.2(14.7)	11.32(0.75)
	0.9	7.34(0.456)	1.52(0.063)	1.47(0.055)	27.41(2.52)	163.3(15.3)	11.95(0.81)
FSM	0.1	3.58(0.223)	1.01(0.029)	0.99(0.032)	16.23(1.88)	**146.0(13.0)**	4.69(0.32)
	0.5	3.29(0.217)	1.23(0.033)	1.10(0.036)	17.45(2.13)	**157.1(13.5)**	4.55(0.27)
	0.9	3.77(0.289)	1.78(0.051)	1.65(0.040)	19.38(2.29)	**142.5(12.7)**	4.72(0.29)
CPPA-PD1	0.1	3.23(0.243)	1.47(0.030)	1.28(0.038)	15.33(1.99)	203.8(19.5)	3.56(0.25)
	0.5	3.17(0.228)	1.35(0.029)	1.19(0.036)	16.77(2.36)	211.9(20.5)	3.79(0.26)
	0.9	3.27(0.256)	1.50(0.032)	1.24(0.041)	19.84(2.71)	200.3(19.7)	3.42(0.21)
CPPA-PD2	0.1	2.04(0.112)	0.77(0.016)	0.69(0.013)	9.11(1.02)	208.4(19.7)	2.74(0.13)
	0.5	2.01(0.109)	0.74(0.015)	0.70(0.012)	9.08(0.99)	213.6(18.9)	2.78(0.14)
	0.9	2.14(0.112)	0.81(0.018)	0.73(0.013)	10.1(1.12)	221.8(19.0)	2.81(0.13)
CPPA-PD3	0.1	**1.13(0.040)**	0.28(0.005)	0.23(0.004)	**0.00(0.000)**	213.4(20.2)	2.80(0.13)
	0.5	**1.09(0.037)**	0.22(0.004)	**0.21(0.004)**	0.06(0.001)	220.5(15.4)	2.79(0.13)
	0.9	1.28(0.046)	**0.24(0.003)**	**0.25(0.004)**	**0.08(0.001)**	210.5(14.6)	2.79(0.14)
P-PLAM	0.1	2.33(0.112)	0.77(0.025)	0.71(0.021)	10.36(1.27)	169.4(12.1)	**2.53(0.18)**
	0.5	2.29(0.125)	0.75(0.023)	0.68(0.019)	12.47(1.32)	181.7(13.8)	**2.74(0.16)**
	0.9	2.57(0.173)	0.98(0.041)	0.77(0.030)	14.08(2.08)	172.5(12.6)	**2.61(0.15)**
TADMM	0.1	1.45(0.057)	0.25(0.005)	0.21(0.005)	15.63(1.61)	357.3(28.3)	6.88(0.65)
	0.5	1.39(0.053)	0.24(0.004)	0.22(0.005)	16.22(1.67)	369.7(27.5)	7.03(0.72)
	0.9	1.67(0.061)	0.28(0.006)	0.27(0.006)	23.41(2.13)	382.2(26.2)	7.25(0.71)
PPA	0.1	1.15(0.042)	**0.24(0.004)**	**0.20(0.004)**	**0.00(0.000)**	219.0(15.3)	2.79(0.12)
	0.5	1.11(0.039)	**0.21(0.004)**	0.22(0.005)	**0.05(0.001)**	222.2(15.7)	2.81(0.15)
	0.9	**1.27(0.045)**	0.25(0.004)	0.25(0.005)	0.09(0.001)	215.3(14.9)	2.75(0.12)

provide a detailed explanation of this phenomenon. Different parameter settings alter the iterative form of CPPA-PD. When $\tau = 1$, CPPA-PD does not have a correction step. When $s = 1$, $r = \eta$, then the $\boldsymbol{\beta}$ update of CPPA-PPA is the same as that of Algorithm 1. Among the three variants of CPPA-PD, CPPA-PD1 performs the worst. This is mainly because the parameters corresponding to CPPA-PD1 are not suitable for models with extremely high sparsity. The solution obtained through computation has an excessively high false positive (FP) value, which in turn leads to poor performance in the first three error-related metrics. There are primarily two reasons for this phenomenon. The first reason pertains to the correction mechanism. The correction step, given by the formula $\boldsymbol{\beta}^{t+1} = \boldsymbol{\beta}^t - 1.2(\boldsymbol{\beta}^t - \tilde{\boldsymbol{\beta}}^t)$, which combines information from $\boldsymbol{\beta}^t$ and $\tilde{\boldsymbol{\beta}}^t$, can impede the estimation of highly sparse data. Specifically, as long as either $\boldsymbol{\beta}^t$ or $\tilde{\boldsymbol{\beta}}^t$ is non-zero, $\boldsymbol{\beta}^{t+1}$ will also be non-zero. In contrast, Algorithm 1 does not incorporate such a correction step. The second reason is related to the

parameter values in high-dimensional models, as presented in Table 1 and Table 2. In these cases, the value of s exceeds 1.2, while r is less than $\|\boldsymbol{A}^\top \boldsymbol{A}\|$. This value of r results in the matrix corresponding to the added linearized quadratic term in the $\boldsymbol{\beta}$-subproblem no longer being positive definite (refer to (12)). Although a complement was introduced in the $\boldsymbol{u}$ - subproblem to guarantee the convergence of CPPA-PD1, the non-positive definite quadratic approximation makes the $\boldsymbol{\beta}$-subproblem less accurate compared to the $\boldsymbol{\beta}$-subproblem solved by Algorithm 1. CPPA-PD2 aims to address the first issue. By setting $\tau = 1$ and eliminating the correction step, CPPA-PD2 shows improved numerical performance compared to CPPA-PD1. Building on CPPA-PD2, CPPA-PD3 further resolves the positive-definiteness problem of the linearization in the $\boldsymbol{\beta}$-subproblem, thereby achieving even better numerical performance. Notably, CPPA-PD3 exhibits similar performance to PPA. This can be explained by their iterative forms in the algorithm. The $\boldsymbol{\beta}$-subproblems of CPPA-PD3 and PPA are the same (the $\boldsymbol{\beta}$-subproblem is the main iterative step for solving the DS model). Although there are some differences in constant terms when updating $\boldsymbol{z}$ and $\boldsymbol{u}$, these differences do not affect the efficiency of algorithm.

4.2 Parallel environment

In the parallel environment, we focus on denser coefficients (many coefficients are non-zero) rather than sparse ones. In fact, dense coefficients often appear in numerical experiments related to DS algorithms, as seen in Lu et al. (2012), Wang and Yuan (2012), Prater et al. (2015), He et al. (2015) and Mao et al. (2021). Here, we set $(n, p, |\mathcal{A}|) = (720s, 2560s, 320s)$ for $s = 1, \ldots, 10$. Under this configuration, non-zero coefficients constitute one-eighth of the total coefficients. Consequently, we partition p into 80 segments and randomly select 10 segments as non-zero coefficients, each containing $32s$ non-zero entries. These non-zero coefficients are defined as follows:

$$\beta_j^* = \begin{cases} \xi_j(1 + |a_j|), & \text{if } j \in \mathcal{A}, \\ 0, & \text{otherwise,} \end{cases} \tag{32}$$

where ξ_j is randomly selected from the set $\{+1, -1\}$ and a_j follows a $\mathcal{N}(0, 1)$ distribution. When implementing parallel computing, matrix $\boldsymbol{A}$ will be divided into K blocks. Currently, the only parallel computing algorithm for DS is the three-block ADMM algorithm (TADMM) proposed by Wen et al. (2024). Therefore, we will only compare the two parallel PPA algorithms presented in this paper with TADMM.

Unlike the evaluation criteria for sparse coefficients, we use the following five evaluation criteria in the example of dense coefficients: (1) absolute error (AE), expressed by $\|\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*\|_1/p$; (2) false positives (FP), representing the quantity of non-significant features chosen; (3) false negatives (FN), indicating the number of significant features omitted; (4) number of iterations (NI); and (5) CPU runtime (Time), measured in seconds. To save space, we present results only for $K = 1, 5, 10, 20$ and $\rho = 0.5$ in Table 3 and Table 4. Note that $K = 1$ means computing in a nonparallel environment. Results for other values of K will be shown in Figure 1 and Figure 2. The above numerical experimental results are all for convex ℓ_1-DS. For numerical experiments on nonconvex DS, such as SCAD-DS and MCP-DS, we have included them in the Appendix C.2.

Table 2 Comparison of various algorithms for solving ℓ_1-DS in nonparallel environments with data scale $(n, p) = (1000, 10,000)$.

Method	ρ	ℓ_1 error	ℓ_2 error	Model error	FP	NI	Time(s)
	0.1	9.31(0.523)	2.78(0.075)	2.36(0.067)	35.27(4.42)	429.2(32.7)	1086.3(76.2)
ADMM	0.5	9.24(0.516)	2.74(0.071)	2.41(0.069)	36.11(4.58)	430.1(33.5)	1092.2(78.5)
	0.9	10.50(0.54)	2.89(0.082)	2.45(0.072)	37.59(4.67)	441.8(34.9)	1103.4(82.0)
	0.1	6.65(0.391)	2.23(0.065)	2.10(0.057)	27.32(4.02)	**425.3(31.8)**	662.1(35.2)
FSM	0.5	6.57(0.382)	2.15(0.061)	2.09(0.063)	25.87(3.87)	**423.7(31.2)**	652.3(34.1)
	0.9	6.01(0.395)	2.34(0.068)	2.24(0.068)	24.96(4.13)	**437.4(38.9)**	671.0(40.3)
	0.1	2.73(0.076)	0.67(0.019)	0.59(0.052)	20.45(3.32)	493.0(43.5)	563.2(39.5)
CPPA-PD1	0.5	2.68(0.072)	0.61(0.017)	0.52(0.049)	19.57(3.10)	497.6(47.2)	572.0(40.3)
	0.9	2.95(0.083)	0.75(0.023)	0.48(0.041)	24.33(3.54)	488.2(50.3)	588.4(41.2)
	0.1	2.06(0.034)	0.45(0.013)	0.41(0.023)	13.67(2.51)	477.1(42.4)	540.3(40.0)
CPPA-PD2	0.5	2.11(0.033)	0.40(0.010)	0.38(0.046)	14.40(2.41)	472.2(44.8)	533.0(42.2)
	0.9	2.08(0.036)	0.51(0.020)	0.47(0.038)	13.86(1.99)	469.6(47.1)	530.3(45.5)
	0.1	**1.55(0.018)**	**0.40(0.008)**	0.32(0.009)	**12.10(0.78)**	452.1(35.2)	457.0(26.0)
CPPA-PD3	0.5	**1.60(0.022)**	**0.49(0.009)**	0.36(0.010)	11.93(0.77)	475.8(39.3)	477.3(29.1)
	0.9	1.75(0.033)	**0.61(0.011)**	0.37(0.009)	**12.40(0.86)**	488.4(44.6)	503.8(34.4)
	0.1	2.56(0.068)	0.58(0.015)	0.51(0.039)	36.32(4.20)	451.6(34.2)	**360.3(21.6)**
P-PLAM	0.5	2.47(0.061)	0.55(0.014)	0.50(0.034)	38.56(5.11)	438.2(36.9)	**352.4(19.8)**
	0.9	2.73(0.072)	0.63(0.017)	0.56(0.041)	40.32(4.97)	443.0(41.3)	**368.8(22.3)**
	0.1	4.37(0.269)	1.64(0.045)	1.51(0.032)	23.25(2.41)	500+(0.00)	800.3(42.4)
TADMM	0.5	4.25(0.261)	1.58(0.042)	1.46(0.030)	21.73(2.25)	500+(0.00)	813.6(50.8)
	0.9	4.69(0.278)	1.70(0.048)	1.55(0.034)	26.08(2.83)	500+(0.00)	798.1(46.3)
	0.1	1.57(0.020)	0.42(0.009)	**0.30(0.008)**	12.23(0.81)	456(36.3)	460.2(27.2)
PPA	0.5	1.62(0.024)	0.51(0.010)	**0.32(0.009)**	**11.91(0.75)**	473(40.5)	480.4(30.3)
	0.9	**1.74(0.031)**	0.63(0.012)	**0.35(0.009)**	12.55(0.89)	491(45.8)	507.2(35.6)

An interesting observation in two figures is that in Figure 1, when $K \geq 20$, increasing the number of partitions K of the gram matrix does not accelerate the computation speed for three parallel algorithms. In Figure 2, this phenomenon appears when $K \geq 30$. The observed phenomenon is likely a consequence of the combined effects of increased synchronization and communication overhead, potential load imbalance, memory bandwidth and cache constraints, non-linear scaling of algorithmic complexity, and hardware limitations in handling a large number of parallel tasks. Nevertheless, moderately and effectively increasing the number of parallelism is an effective way to accelerate the computation speed of DSs. In Tables 3 and 4, with the variation of K, all metrics in PPPA, excluding the Time metric, remain constant and are identical to those when $K = 1$ (nonparallel environment). The reduction in computation time can be attributed to two factors. Firstly, matrix partitioning shortens the time required for matrix-matrix or matrix-vector multiplications. Secondly, the iterative variables

Table 3 Comparison of parallel environment for solving ℓ_1-DS with $s = 5$.

Method	K	AE	FP	FN	Ite	Time(s)
TADMM	1	0.412(0.148)	123.6(14.2)	0.32(0.002)	500+(0.00)	2528.6(123.2)
	5	0.453(0.151)	131.5(15.6)	0.36(0.002)	500+(0.00)	623.4(45.6)
	10	0.528(0.165)	137.1(16.0)	0.47(0.003)	500+(0.00)	381.9(31.8)
	20	0.545(0.179)	141.2(16.3)	0.51(0.003)	500+(0.00)	237.0(22.5)
PPPA	1	**0.103(0.012)**	**85.52(9.2)**	0.12(0.001)	**442.3(31.8)**	994.2(63.1)
	5	**0.103(0.012)**	**85.52(9.2)**	**0.12(0.001)**	**442.3(31.8)**	278.4(23.5)
	10	**0.103(0.012)**	**85.52(9.2)**	**0.12(0.001)**	**442.3(31.8)**	**175.8(16.2)**
	20	**0.103(0.012)**	**85.52(9.2)**	**0.12(0.001)**	**442.3(31.8)**	**132.1(10.4)**
IPPPA	1	0.109(0.013)	87.28(9.90)	**0.11(0.001)**	456.6(33.7)	**902.3(57.9)**
	5	0.110(0.014)	89.15(10.3)	0.13(0.001)	463.2(34.2)	**268.6(20.6)**
	10	0.113(0.014)	91.34(10.9)	0.15(0.001)	466.8(35.1)	181.5(15.8)
	20	0.115(0.015)	93.40(11.5)	0.18(0.001)	475.4(36.5)	147.0(11.5)

Table 4 Comparison of parallel environment for solving ℓ_1-DS with $s = 10$.

Method	K	AE	FP	FN	Ite	Time(s)
TADMM	1	0.833(0.227)	352.5(30.4)	1.32(0.010)	500+(0.00)	5443.7(277.5)
	5	0.842(0.239)	361.0(32.7)	1.57(0.016)	500+(0.00)	1015.5(152.7)
	10	0.857(0.248)	372.2(35.1)	1.84(0.021)	500+(0.00)	692.7(72.9)
	20	0.866(0.256)	380.1(38.2)	2.23(0.027)	500+(0.00)	386.2(45.8)
PPPA	1	0.191(0.024)	**163.6(15.8)**	0.51(0.006)	**472.6(34.9)**	1624.1(92.5)
	5	0.191(0.024)	**163.6(15.8)**	**0.51(0.006)**	**472.6(34.9)**	357.5(35.2)
	10	**0.191(0.024)**	**163.6(15.8)**	**0.51(0.006)**	**472.6(34.9)**	**245.9(20.3)**
	20	**0.191(0.024)**	**163.6(15.8)**	**0.51(0.006)**	**472.6(34.9)**	**161.2(13.6)**
IPPPA	1	**0.185(0.022)**	172.3(16.4)	**0.47(0.06)**	487.8(40.1)	**1476.5(98.9)**
	5	**0.190(0.023)**	179.2(17.2)	0.53(0.07)	498.5(43.7)	**301.4(29.7)**
	10	0.196(0.024)	184.6(18.5)	0.61(0.08)	500+(50.2)	257.8(22.1)
	20	0.199(0.026)	190.2(20.7)	0.64(0.08)	500+(55.3)	212.52(19.2)

corresponding to each sub-matrix are processed in parallel. This is also the partition insensitivity described in Theorem 1. This insensitivity is also reflected in Figures 1 and 2, where the AE of PPA is shown as a horizontal line. The two PPA algorithms outperform TADMM in terms of estimation accuracy, variable selection, and computational efficiency. As we discussed earlier, our algorithms have fewer iterative variables, which not only enhances the accuracy of the solutions but also improves computational speed.

Fig. 1 A schematic diagram illustrating the variation of AE and Time with respect to K, where $s = 5$.

Fig. 2 A schematic diagram illustrating the variation of AE and Time with respect to K, where $s = 10$.

The numerical performance of the improved parallel PPA (IPPPA) compared to the parallel PPA (PPPA) presented in Tables 3 and 4 does not appear to have improved. In many cases, IPPPA may require even more iterations and longer computation times. The reason IPPPA outperforms PPPA in computational effectiveness is

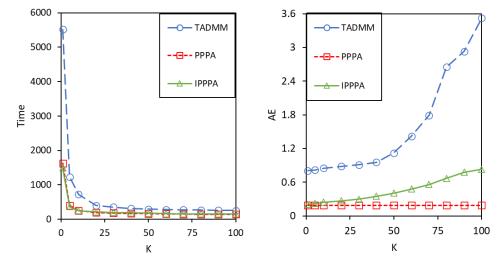that IPPPA approximates $\text{eigen}(\mu \boldsymbol{A}^\top \boldsymbol{A})$ by calculating the eigenvalues of submatrices $\text{eigen}(\mu \boldsymbol{A}_i^\top \boldsymbol{A}_i)$ instead of directly computing $\text{eigen}(\mu \boldsymbol{A}^\top \boldsymbol{A})$. This approach leads to additional iterations, and indeed, as K increases, the computational performance of IPPPA deteriorates and its efficiency decreases. The real advantage of IPPPA over PPPA comes into play when the storage requirements for the matrix are too large to allow for $\text{eigen}(\mu \boldsymbol{A}^\top \boldsymbol{A})$ calculations; in such cases, PPPA becomes infeasible, while IPPPA can still be implemented.

5 Numerical simulations using real dataset

In this section, we utilize two real-world datasets to construct three different DS models: ℓ_1-DS, SCAD-DS, and MCP-DS. These models are trained using the parallel computation algorithms TADMM, proposed by Wen et al. (2024), as well as two parallel PPA algorithms introduced in this paper.

5.1 Leukemia dataset

In this experiment, the convex and nonconvex Dantzig selectors generated by various algorithms are applied to a set of biomarker data to determine whether a patient may be diagnosed with a specific type of cancer. Leukemia is a type of cancer that affects the blood or bone marrow and is characterized by an abnormal increase in white blood cells. The leukemia dataset first introduced in Golub et al. (1999) consists of 38 training samples and 34 test samples. Among the 38 training samples, 27 are classified as acute lymphocytic leukemia (ALL), while the remaining 11 are classified as acute myelogenous leukemia (AML). Each sample in this dataset includes measurements of 7,129 genes. This data has been studied in the literature through various regularization methods and Dantzig selectors, such as in Lu et al. (2012), Wang and Yuan (2012) Prater et al. (2015) and Wu et al. (2024).

Table 5 Comparison of three DSs for leukaemia data.

Method	Algorithm	TRE	TEE	$|\hat{\mathcal{A}}|$	Ite	Time(s)
	TADMM	**0/38**	2/34	206	182	1.96
ℓ_1-DS	PPPA	**0/38**	**0/34**	**63**	**120**	**1.21**
	IPPPA	**0/38**	**0/34**	67	123	1.26
	TADMM	**0/38**	**0/34**	105	193	2.07
SCAD-DS	PPPA	**0/38**	**0/34**	44	126	1.28
	IPPPA	**0/38**	**0/34**	**42**	**125**	**1.24**
	TADMM	**0/38**	1/34	109	195	2.11
MCP-DS	PPPA	**0/38**	**0/34**	**49**	**127**	**1.30**
	IPPPA	**0/38**	**0/34**	50	130	1.33

Let $\boldsymbol{X}_{\text{train}} \in \mathbb{R}^{38 \times 7129}$ represent the leukemia dataset from the training set, where each row corresponds to the $7,129$ gene measurements of a single patient, and each column has been normalized to have unit ℓ_2 norm. Let $\boldsymbol{y}_{\text{train}} \in \mathbb{R}^{38}$ denote the column

vector indicating the diagnosis of each patient within the training set, defined as:

$$\boldsymbol{y}_{\text{train}}(j) = \begin{cases} 1, & \text{if patient } j \text{ in the training set is diagnosed with AML,} \\ 0, & \text{if patient } j \text{ in the training set is diagnosed with ALL.} \end{cases}$$

Similarly, define $\boldsymbol{X}_{\text{test}} \in \mathbb{R}^{34 \times 7129}$ and $\boldsymbol{y}_{\text{test}} \in \mathbb{R}^{34}$ using the data from the testing set.

During the testing phase, the trained parameter $\hat{\boldsymbol{\beta}}$ is utilized to predict the diagnoses of patients in the testing set. The predictive indicator vector $\hat{\boldsymbol{y}}_{\text{test}} \in \mathbb{R}^{34}$ is computed from $\boldsymbol{y} = \boldsymbol{X}_{\text{test}}\hat{\boldsymbol{\beta}}$ by applying thresholding and clustering to identify values near the threshold boundary. Define $\hat{\boldsymbol{y}}_{\text{test}}(j)$ as follows:

$$\hat{\boldsymbol{y}}_{\text{test}}(j) = \begin{cases} 1, & \text{if } y(j) \geq 0.5, \\ 0, & \text{if } y(j) < 0.49. \end{cases}$$

This allows us to categorize the predictions based on whether they fall above or below the specified thresholds.

In a parallel computing environment, we randomly partition the $7,129$ features into 5 groups, denoted as $K = 5$, with 4 groups containing $1,425$ features each and one group containing $1,429$ features. In Table 5, we present the results of the numerical experiments, where TRE represents the training error, TEE denotes the testing error, and $|\hat{\mathcal{A}}|$ indicates the number of estimated coefficients that are non-zero. Overall, the numerical results indicate that, in a parallel computing environment, the two parallel PPA algorithms yield lower prediction errors and require fewer iterations compared to TADMM when solving the DS models constructed using leukemia dataset. This substantial reduction in iterations significantly saves computational time.

Specifically, in terms of TRE metric, all three parallel algorithms can accurately identify symptoms of ALL and AML. However, in terms of TEE metric, the parallel PPA algorithm proposed in this paper will perform better than the TADMM algorithm. In details, on the prediction set, ℓ_1-DS model trained by the PPA algorithms accurately identifies 20 cases of ALL and 14 cases of AML. In contrast, ℓ_1-DS model trained by the TADMM algorithm misidentifies one case of AML as ALL and one case of ALL as AML. That is to say, TEE is 2 out of 34. The SCAD-DS model trained by the three algorithms can accurately recognize all ALL and AML. For the MCP-DS model trained by the three algorithms, the model trained by PPAs can accurately recognize all ALL and AML, but TADMM will incorrectly recognize one AML, which has a TEE of 1/34. More importantly, the two PPA algorithms select a smaller number of non-zero coefficients. In other words, the value of $|\hat{\mathcal{A}}|$ for the PPA algorithms is smaller than that for the TADMM algorithm. This characteristic holds a certain guiding significance in practical applications. It implies that in the future, one only needs to observe the features corresponding to the non-zero coefficients in each sample. Consequently, this approach can help save the cost of data collection.

5.2 Supermarket dataset

In this subsection, we assess the performance of the proposed algorithms using a supermarket dataset previously analyzed by Wang (2009). This data was also evaluated by Wen et al. (2024) for their proposed regularization method and algorithm. This dataset includes the daily number of customers and the daily sales volumes of $p = 6,398$ products from a Chinese supermarket over a period of $n = 464$ days. The supermarket manager is keen to determine which product's sales volume exhibits the strongest correlation with the number of customers, after accounting for the influence of other products. Then, DS models prove to be valuable. Hence, our goal is to predict the daily number of customers (the response variable of interest) using the daily sales volumes as predictors. In line with Wang (2009), both the response variable and predictors have been standardized to have a mean of zero and a variance of one for confidentiality reasons.

Table 6 Comparison of three DSs for supermarket data.

Method	Algorithm	TRE	TEE	$|\hat{\mathcal{A}}|$	Ite	Time(s)
	TADMM	0.232	0.329	172	223	2.53
ℓ_1-DS	PPPA	0.227	**0.275**	**96**	**141**	**1.59**
	IPPPA	**0.225**	0.298	103	152	1.62
	TADMM	0.226	0.349	41	271	2.98
SCAD-DS	PPPA	0.218	**0.312**	39	**149**	**1.70**
	IPPPA	**0.213**	0.320	**36**	163	1.81
	TADMM	0.231	0.332	50	268	2.85
MCP-DS	PPPA	**0.219**	**0.316**	**43**	**153**	**1.72**
	IPPPA	0.227	0.325	41	167	1.84

Following Wen et al. (2024), we divide the dataset into a training set comprising observations from the first 300 days and a testing set consisting of the remaining days. Firstly, using the training dataset, we construct three DS models: ℓ_1-DS, SCAD-DS, and MCP-DS. Next, we employ the TADMM proposed by Wen et al. (2024) and the two parallel PPA algorithms presented in this paper to solve these models. The obtained estimated coefficients are then used for prediction.

In a parallel computing environment, we randomly partition the $p = 6,398$ features into 5 groups, denoted as $K = 5$, with 4 groups containing $1,279$ features each and one group containing $1,282$ features. We present the results of the numerical experiments in the Table 6, where TRE represents the training error, TEE denotes the testing error, and $|\hat{\mathcal{A}}|$ indicates the number of estimated coefficients that are non-zero. The numerical results in Table 6 indicate that the two parallel PPA algorithms have better training and testing errors compared to TADMM when solving the DS models constructed with supermarket dataset, and also have improved computational efficiency. To be specific, two PPA algorithms offer more accurate predictions of the daily number of customers compared to the TADMM algorithm. This means that supermarket managers can utilize our algorithms to obtain more precise forecasts, enabling them to better arrange the quantity of goods and the number of service staff. In addition, the coefficients

6 Conclusion and further research

In this paper, we have successfully developed a novel variable splitting parallel algorithm for addressing both convex and nonconvex Dantzig selectors, leveraging the proximal point algorithm. Our approach uniquely minimizes the number of iteration variables, thereby significantly enhancing computational efficiency and accelerating convergence. Notably, our algorithm exhibits a partition-insensitive property, ensuring consistent performance regardless of data partitioning. Theoretical analysis has confirmed the linear convergence of our algorithm, and empirical results demonstrate its competitive performance across various computational environments. Additionally, to handle extreme situations where the matrix dimension is excessively large, and the maximum eigenvalue of the entire gram matrix cannot be computed, we have developed an enhanced version of this parallel PPA algorithm. The R package for our algorithm, available at https://github.com/xfwu1016/PPADS, invites contributions and additional applications from the research community.

While our work has shown promising results, future research could explore several avenues for further enhancement. Firstly, investigating adaptive partitioning strategies to optimize performance across diverse datasets could yield even greater efficiency gains. Secondly, extending the algorithm to handle streaming data scenarios, where data arrives sequentially, would broaden its applicability. Lastly, integrating our method with advanced machine learning models and leveraging GPU acceleration for even faster processing could further solidify its utility in large-scale data applications.

Acknowledgements

Appendix A   Proofs of Theorems 1 and 4

A.1   Proof of Theorem 1

Recall that we denote $\left\{\hat{\boldsymbol{\beta}}^t, \hat{\boldsymbol{z}}^t, \hat{\boldsymbol{u}}^t\right\}$ as the results of the t-th iteration of Algorithm 1, and $\left\{\tilde{\boldsymbol{\beta}}^t, \tilde{\boldsymbol{z}}^t, \tilde{\boldsymbol{u}}^t\right\}$ as the results of the i-th iteration of Algorithm 2. We make the

assumption that

$$\left\{ \hat{\boldsymbol{\beta}}^t, \hat{\boldsymbol{z}}^t, \hat{\boldsymbol{u}}^t \right\} = \left\{ \tilde{\boldsymbol{\beta}}^t, \tilde{\boldsymbol{z}}^t, \tilde{\boldsymbol{u}}^t \right\}, \tag{A1}$$

and subsequently, we will conduct an analysis of the iteration scenario at step $t + 1$.

Firstly, we will discuss the update of the $\boldsymbol{\beta}$-subproblem. For Algorithm 1, we have

$$\hat{\boldsymbol{\beta}}^{t+1} \leftarrow \arg\min_{\boldsymbol{\beta}} \left\{ \|\boldsymbol{\beta}\|_1 + \frac{\eta}{2} \|\boldsymbol{\beta} - \hat{\boldsymbol{\beta}}^t - \frac{\boldsymbol{A}^\top \hat{\boldsymbol{u}}^t}{\eta}\|_2^2 \right\}; \tag{A2}$$

and for Algorithm 2, we have

$$\tilde{\boldsymbol{\beta}}_{i\cdot}^{t+1} \leftarrow \arg\min_{\boldsymbol{\beta}_{i\cdot}} \left\{ \|\boldsymbol{\beta}_{i\cdot}\|_1 + \frac{\eta}{2} \|\boldsymbol{\beta}_{i\cdot} - \tilde{\boldsymbol{\beta}}_{i\cdot}^t - \frac{\boldsymbol{A}_i^\top \tilde{\boldsymbol{u}}^t}{\eta}\|_2^2 \right\}, i = 1, 2, \ldots, K. \tag{A3}$$

If $\hat{\boldsymbol{\beta}}^{t+1}$ is also decomposed from the column, that is,

$$\hat{\boldsymbol{\beta}}^{t+1} = ((\hat{\boldsymbol{\beta}}_{1\cdot}^{t+1})^\top, (\hat{\boldsymbol{\beta}}_{2\cdot}^{t+1})^\top, \ldots, (\hat{\boldsymbol{\beta}}_{K\cdot}^{t+1})^\top)^\top.$$

Since $\boldsymbol{A} = [\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_K]$, it can be concluded that

$$\hat{\boldsymbol{\beta}}_{i\cdot}^{t+1} \leftarrow \arg\min_{\boldsymbol{\beta}_{i\cdot}} \left\{ \|\boldsymbol{\beta}_{i\cdot}\|_1 + \frac{\eta}{2} \|\boldsymbol{\beta}_{i\cdot} - \hat{\boldsymbol{\beta}}_{i\cdot}^t - \frac{\boldsymbol{A}_i^\top \hat{\boldsymbol{u}}^t}{\eta}\|_2^2 \right\}, i = 1, 2, \ldots, K. \tag{A4}$$

According to the assumption in (A1), there is $\hat{\boldsymbol{\beta}}_{i\cdot}^t = \tilde{\boldsymbol{\beta}}_{i\cdot}^t$ and $\hat{\boldsymbol{u}}^t = \tilde{\boldsymbol{u}}^t$. It follows from (A3) and (A4) that

$$\hat{\boldsymbol{\beta}}_{i\cdot}^{t+1} = \tilde{\boldsymbol{\beta}}_{i\cdot}^{t+1}, i = 1, 2, \ldots, K. \tag{A5}$$

Next, we will discuss the update of the $\boldsymbol{z}$-subproblem. For Algorithm 1, we have

$$\hat{\boldsymbol{z}}^{t+1} \leftarrow \arg\min_{\boldsymbol{z}} \left\{ \delta_{\mathcal{Z}_0(\boldsymbol{z})} + \frac{\mu}{2} \|\boldsymbol{z} - \hat{\boldsymbol{z}}^t + \frac{\hat{\boldsymbol{u}}^t}{\mu}\|_2^2 \right\}; \tag{A6}$$

and for Algorithm 2, we get

$$\tilde{\boldsymbol{z}}^{t+1} \leftarrow \arg\min_{\boldsymbol{z}} \left\{ \delta_{\mathcal{Z}_0(\boldsymbol{z})} + \frac{\mu}{2} \|\boldsymbol{z} - \tilde{\boldsymbol{z}}^t + \frac{\tilde{\boldsymbol{u}}^t}{\mu}\|_2^2 \right\}; \tag{A7}$$

Again, according to the assumption in (A1), there is $\hat{\boldsymbol{z}}^t = \tilde{\boldsymbol{z}}^t$ and $\hat{\boldsymbol{u}}^t = \tilde{\boldsymbol{u}}^t$. It follows from (A6) and (A7) that

$$\hat{\boldsymbol{z}}^{t+1} = \tilde{\boldsymbol{z}}^{t+1}. \tag{A8}$$

31

Finally, we will discuss the update of the $\boldsymbol{u}$-subproblem. Since $\sum_{i=1}^{K} \boldsymbol{A}_i \boldsymbol{\beta}_{i\cdot} = \boldsymbol{A}\boldsymbol{\beta}$, It follows from (A5) and (A8) that

$$\hat{\boldsymbol{u}}^{t+1} = \tilde{\boldsymbol{u}}^{t+1}. \tag{A9}$$

Therefore, we can draw the conclusion that when the iteration sequences of the two algorithms at step t are the same, the iteration sequences at step $t+1$ will also be the same. Based on this, assuming that $\{\hat{\boldsymbol{\beta}}^0, \hat{\boldsymbol{z}}^0, \hat{\boldsymbol{u}}^0\} = \{\tilde{\boldsymbol{\beta}}^0, \tilde{\boldsymbol{z}}^0, \tilde{\boldsymbol{u}}^0\}$, it follows that $\{\hat{\boldsymbol{\beta}}^1, \hat{\boldsymbol{z}}^1, \hat{\boldsymbol{u}}^1\} = \{\tilde{\boldsymbol{\beta}}^1, \tilde{\boldsymbol{z}}^1, \tilde{\boldsymbol{u}}^1\}$. By the same token, we can deduce that

$$\left\{\hat{\boldsymbol{\beta}}^t, \hat{\boldsymbol{z}}^t, \hat{\boldsymbol{u}}^t\right\} = \left\{\tilde{\boldsymbol{\beta}}^t, \tilde{\boldsymbol{z}}^t, \tilde{\boldsymbol{u}}^t\right\}, \quad \text{for all } t. \tag{A10}$$

Up to this point, the proof of the theorem has been successfully completed.

A.2 Proof of Theorem 4

This proof is divided into two parts, first proving the time complexity of the algorithm, and then proving the space complexity of the algorithm.

(1). Reviewing the iterative processes of Algorithm 2 and Algorithm 3, we observe that the updates of the $\boldsymbol{\beta}$, $\boldsymbol{z}$ and $\boldsymbol{u}$ subproblems are nearly identical. The main computational burden of $\boldsymbol{\beta}$ updates is the calculation of $\boldsymbol{A}^\top \boldsymbol{u}$. Due to its parallel computation across K nodes, each node only needs to calculate $\boldsymbol{A}_i^\top \boldsymbol{u}$. Its computational complexity is $\mathcal{O}(\frac{p^2}{K})$. The update of the $\boldsymbol{z}$ subproblem only involves the addition, subtraction, and comparison of some p-dimensional vectors. Then, its computational complexity is $\mathcal{O}(p)$. The biggest computational burden of updating $\boldsymbol{u}$ lies in computing $\boldsymbol{A}\boldsymbol{\beta} = \sum_{i=1}^{K} \boldsymbol{A}_i \boldsymbol{\beta}_{i\cdot}$, which will be computed in parallel across K nodes. Its computational complexity is $\mathcal{O}(\frac{p^2}{K})$. Clearly, $\mathcal{O}(\frac{p^2}{K}) + \mathcal{O}(p) + \mathcal{O}(\frac{p^2}{K}) = \mathcal{O}(\frac{p^2}{K})$ because the number of column partitions cannot exceed the number of columns, i.e. $p \geq K$.

Next, we need to discuss the time complexity of the pre-computation for Algorithm 2 and Algorithm 3 separately. Algorithm 2 requires pre-computation of $\boldsymbol{X}^\top \boldsymbol{y}$, $\boldsymbol{A}_i = \boldsymbol{X}^\top \boldsymbol{X}_i$, and calculate η using the power method. The time complexity required for the first and second items is $\mathcal{O}(np)$ and $\mathcal{O}(\frac{np^2}{K})$, respectively. As discussed in Liang et al. (2024), the computational complexity required for the power method is quadratic with the number of columns in matrix $\boldsymbol{A}$, i.e. $\mathcal{O}(p^2)$. Thus, the time complexity of the pre-computation for Algorithm 2 is $\mathcal{O}(\frac{np^2}{K}) + \mathcal{O}(p^2)$. The main difference between Algorithm 3 pre-computation and Algorithm 2 is that one calculates η and the other calculates η_i. Similarly, the complexity of calculating η_i is $\mathcal{O}(\frac{p^2}{K^2})$. Thus, the time complexity of the pre-computation for Algorithm 3 is $\mathcal{O}(\frac{np^2}{K})$. By summarizing the above discussion, we can arrive at the conclusion in (24).

(2). Next, we will discuss the space complexity of the Algorithm 2 and Algorithm 3. The space complexity of algorithms mainly includes three parts: the required storage for inputs and outputs, and the additional storage required for computation. The maximum storage space required for the input and output of Algorithm 2 is matrix $\boldsymbol{X}$,

which is $\mathcal{O}(np)$. The largest additional storage during the precomputation process is the storage of matrix $\boldsymbol{A}$, which is $\mathcal{O}(p^2)$. The additional storage required for updating the three subproblems is relatively small, which is $\mathcal{O}(p)$. Therefore, the required storage space for Algorithm 2 is $\mathcal{O}(np) + \mathcal{O}(p^2)$.

Similarly, the maximum storage space required for the input and output of Algorithm 3 is matrix $\boldsymbol{X}$, which is $\mathcal{O}(np)$. he largest additional storage during the precomputation process is the storage of matrix $\boldsymbol{A}_i$, which is $\mathcal{O}(\frac{p^2}{K})$. The updates of the three sub problems in Algorithm 3 are similar to those in Algorithm 2, requiring only $\mathcal{O}(p)$ of additional storage space. Therefore, the required storage space for Algorithm 3 is $\mathcal{O}(np) + \mathcal{O}(\frac{p^2}{K})$.

Appendix B   Proofs of Theorems 2 and 3

B.1   preliminary

B.1.1   Lemma

Lemma 1. *(Lemma 2.1 in He et al. (2022)). Let $\mathbb{X} \subset \mathbb{R}^l$ be a closed convex set, and let $\theta : \mathbb{R}^l \to \mathbb{R}$ and $f : \mathbb{R}^l \to \mathbb{R}$ be convex functions. Suppose f is differentiable on an open set containing $\mathbb{X}$, and the minimization problem*

$$\min\{\theta(x) + f(x) \mid x \in \mathbb{Z}\}$$

has a nonempty solution set. Then, $x^* \in \arg\min\{\theta(x) + f(x) \mid x \in \mathbb{X}\}$ if and only if

$$x^* \in \mathbb{X} \quad and \quad \theta(x) - \theta(x^*) + (x - x^*)^\top \nabla f(x^*) \geq 0, \quad \forall x \in \mathbb{X}.$$

Lemma 2. *Assume that $\boldsymbol{H} \in \mathbb{R}^{n \times n}$ is a positive definite matrix, and $\boldsymbol{a}, \boldsymbol{b}$ are two arbitrary n-dimensional vectors. If $\boldsymbol{b}^\top \boldsymbol{H}(\boldsymbol{a} - \boldsymbol{b}) \geq 0$, then we can have*

$$\|\boldsymbol{b}\|_{\boldsymbol{H}}^2 \leq \|\boldsymbol{a}\|_{\boldsymbol{H}}^2 - \|\boldsymbol{a} - \boldsymbol{b}\|_{\boldsymbol{H}}^2.$$

The conclusion of Lemma 2 is very easy to verify. Though simple, this conclusion is widely used to prove PPA and ADMM convergence, as seen in Cai et al. (2013), Gu et al. (2014), He and Yuan (2018), and He et al. (2022).

B.1.2   Four matrices

To simplify the presentation of analysis, we define the following four matrices in (B11) and (B12).

$$\boldsymbol{M} = \begin{pmatrix} \mu\boldsymbol{A}^\top\boldsymbol{A} & \boldsymbol{0} & \boldsymbol{A}^\top \\ \boldsymbol{0} & \mu\boldsymbol{I}_p & -\boldsymbol{I}_p \\ \boldsymbol{A} & -\boldsymbol{I}_p & \frac{2}{\mu}\boldsymbol{I}_p \end{pmatrix}, \quad \boldsymbol{H} = \begin{pmatrix} \mu\boldsymbol{A}^\top\boldsymbol{A} + \boldsymbol{S} & \boldsymbol{0} & \boldsymbol{A}^\top \\ \boldsymbol{0} & \mu\boldsymbol{I}_p & -\boldsymbol{I}_p \\ \boldsymbol{A} & -\boldsymbol{I}_p & \frac{2}{\mu}\boldsymbol{I}_p \end{pmatrix}, \qquad \text{(B11)}$$

where $\boldsymbol{S} = \eta \boldsymbol{I}_p - \mu \boldsymbol{A}^\top \boldsymbol{A}$, and $\eta > \mathrm{eigen}(\mu \boldsymbol{A}^\top \boldsymbol{A})$.

$$\boldsymbol{M}_K = \begin{pmatrix} \boldsymbol{\Lambda}_K & \boldsymbol{G}^\top \\ \boldsymbol{G} & \frac{K+1}{\mu} \boldsymbol{I}_p \end{pmatrix}, \ \boldsymbol{H}_K = \begin{pmatrix} \tilde{\boldsymbol{\Lambda}}_K & \boldsymbol{G}^\top \\ \boldsymbol{G} & \frac{K+1}{\mu} \boldsymbol{I}_p \end{pmatrix}, \tag{B12}$$

where $\boldsymbol{S}_i^{'} = \eta_i \boldsymbol{I}_p - \mu \boldsymbol{A}_i^\top \boldsymbol{A}_i$, $\eta_i > \mathrm{eigen}(\mu \boldsymbol{A}_i^\top \boldsymbol{A}_i)$ and $i \in \{1, \ldots, K\}$. Here,

$$\boldsymbol{G} = (\boldsymbol{A}_1, \ldots, \boldsymbol{A}_K, -\boldsymbol{I}_p),$$

$$\boldsymbol{\Lambda}_K = \mathrm{diag}(\mu \boldsymbol{A}_1^\top \boldsymbol{A}_1, \ldots, \mu \boldsymbol{A}_K^\top \boldsymbol{A}_K, \mu \boldsymbol{I}_p),$$

and

$$\tilde{\boldsymbol{\Lambda}}_K = \mathrm{diag}(\mu \boldsymbol{A}_1^\top \boldsymbol{A}_1 + \boldsymbol{S}_1^{'}, \ldots, \mu \boldsymbol{A}_K^\top \boldsymbol{A}_K + \boldsymbol{S}_K^{'}, \mu \boldsymbol{I}_p) = \mathrm{diag}(\eta_1 \boldsymbol{I}_p, \ldots, \eta_K \boldsymbol{I}_p, \mu \boldsymbol{I}_p).$$

The two matrices in (B11) are utilized for the proof of Theorem 2, while the two matrices in (B12) are employed for the proof of Theorem 3. Among them, $\boldsymbol{M}$ and $\boldsymbol{M}_K$ are positive semidefinite matrices, and $\boldsymbol{H}$ and $\boldsymbol{H}_K$ are positive definite matrices. We employ the following proposition to elucidate this concept.

Proposition 1. $\boldsymbol{M} \succeq \boldsymbol{0}$, $\boldsymbol{M}_K \succeq \boldsymbol{0}$, $\boldsymbol{H} \succ \boldsymbol{0}$ and $\boldsymbol{H}_K \succ \boldsymbol{0}$.

Proof. $\boldsymbol{M}$ and $\boldsymbol{M}_K$ can be represented as

$$\begin{pmatrix} \sqrt{\mu} \boldsymbol{A}^\top \\ \boldsymbol{0} \\ \frac{1}{\sqrt{\mu}} \boldsymbol{I}_p \end{pmatrix} \begin{pmatrix} \sqrt{\mu} \boldsymbol{A} & \boldsymbol{0} & \frac{1}{\sqrt{\mu}} \boldsymbol{I}_p \end{pmatrix} + \begin{pmatrix} \boldsymbol{0} \\ -\sqrt{\mu} \boldsymbol{I}_p \\ \frac{1}{\sqrt{\mu}} \boldsymbol{I}_p \end{pmatrix} \begin{pmatrix} \boldsymbol{0} & -\sqrt{\mu} \boldsymbol{I}_p & \frac{1}{\sqrt{\mu}} \boldsymbol{I}_p \end{pmatrix},$$

and

$$\sum_{i=1}^{K} \left[\begin{pmatrix} \vdots \\ \sqrt{\mu} \boldsymbol{A}_i^\top \\ \vdots \\ \frac{1}{\sqrt{\mu}} \boldsymbol{I}_p \end{pmatrix} \begin{pmatrix} \cdots & \sqrt{\mu} \boldsymbol{A}_i & \cdots & \frac{1}{\sqrt{\mu}} \boldsymbol{I}_p \end{pmatrix} \right] + \begin{pmatrix} \vdots \\ -\sqrt{\mu} \boldsymbol{I}_p \\ \frac{1}{\sqrt{\mu}} \boldsymbol{I}_p \end{pmatrix} \begin{pmatrix} \cdots & -\sqrt{\mu} \boldsymbol{I}_p & \frac{1}{\sqrt{\mu}} \boldsymbol{I}_p \end{pmatrix},$$

respectively. For any non-zero vector $\boldsymbol{v}$, we have $\boldsymbol{v}^T \boldsymbol{M} \boldsymbol{v} \geq 0$ and $\boldsymbol{v}^T \boldsymbol{M}_K \boldsymbol{v} \geq 0$, thus $\boldsymbol{M} \succeq \boldsymbol{0}$ and $\boldsymbol{M}_K \succeq \boldsymbol{0}$. It is evident that $\boldsymbol{M}$ and $\boldsymbol{M}_K$ are at least positive semidefinite matrices; if $\boldsymbol{A}$ and $\boldsymbol{A}_i$ have full column rank, then $\boldsymbol{M}$ and $\boldsymbol{M}_K$ are positive definite matrices.

Similarly, $\boldsymbol{H}$ and $\boldsymbol{H}_K$ can be represented as

$$\boldsymbol{H} = \boldsymbol{M} + \mathrm{diag}(\boldsymbol{S}, \boldsymbol{0}, \boldsymbol{0}),$$

and

$$\boldsymbol{H}_K = \boldsymbol{M}_K + \mathrm{diag}(\boldsymbol{S}_1^{'}, \cdots, \boldsymbol{S}_K^{'}, \boldsymbol{0}, \boldsymbol{0}),$$

respectively. For any non-zero vector $\boldsymbol{v}$, we have $\boldsymbol{v}^T \boldsymbol{H} \boldsymbol{v} > 0$ and $\boldsymbol{v}^T \boldsymbol{H}_K \boldsymbol{v} > 0$, thus $\boldsymbol{H} \succ \boldsymbol{0}$ and $\boldsymbol{H}_K \succ \boldsymbol{0}$. $\qquad \square$

B.1.3 Variational inequality characterization

The constrained optimization form of DS is defined as,

$$\min_{\boldsymbol{\beta}, \boldsymbol{z}} \quad \sum_{i=1}^{K} \|\boldsymbol{\beta}_{i\cdot}\|_1 + \delta_{\mathcal{Z}_0(\boldsymbol{z})}$$

$$\text{s.t.} \quad \sum_{i=1}^{K} \boldsymbol{A}_i \boldsymbol{\beta}_{i\cdot} - \boldsymbol{z} = \boldsymbol{X}^{\top} \boldsymbol{y}, \tag{B13}$$

And the Lagrange multiplier form of (B13) is

$$L(\boldsymbol{\beta}, \boldsymbol{z}; \boldsymbol{u}) = \sum_{i=1}^{K} \|\boldsymbol{\beta}_{i\cdot}\|_1 + \delta_{\mathcal{Z}_0(\boldsymbol{z})} - \boldsymbol{u}^{\top} \left(\sum_{i=1}^{K} \boldsymbol{A}_i \boldsymbol{\beta}_{i\cdot} - \boldsymbol{z} - \boldsymbol{X}^{\top} \boldsymbol{y} \right). \tag{B14}$$

Since $\sum_{i=1}^{K} \|\boldsymbol{\beta}_{i\cdot}\|_1 = \|\boldsymbol{\beta}\|_1$ and $\sum_{i=1}^{K} \boldsymbol{A}_i \boldsymbol{\beta}_{i\cdot} = \boldsymbol{A}\boldsymbol{\beta}$, the parallel and nonparallel constraint optimization forms of DS are the same.

According to the variational inequality characterization in section 2 of He and Yuan (2012), we know that the solution of the constrained optimization function above is the saddle point of the following Lagrangian function in (B14). However, $\sum_{i=1}^{K} \|\boldsymbol{\beta}_{i\cdot}\|_1$ and $\delta_{\mathcal{Z}_0(\boldsymbol{z})}$ are non differentiable, so the variational inequalities they mentioned cannot be directly applied to the scenario in this paper. A recent research work, He et al. (2022), can include the non differentiable scenario designed in our work. As described in their section 2.2, finding a saddle point of $L(\boldsymbol{\beta}, \boldsymbol{z}; \boldsymbol{u})$ is equivalent to finding $\boldsymbol{\beta}^*, \boldsymbol{z}^*, \boldsymbol{u}^*$ such that the following inequalities are satisfied:

$$\boldsymbol{h}^* \in \Omega, \ \theta(\boldsymbol{f}) - \theta(\boldsymbol{f}^*) + (\boldsymbol{g} - \boldsymbol{g}^*)^{\top} F(\boldsymbol{g}^*) \geq 0, \ \forall \boldsymbol{g} \in \Omega, \tag{B15}$$

where $\boldsymbol{f} = (\boldsymbol{\beta}^{\top}, \boldsymbol{z}^{\top})^{\top}$, $\boldsymbol{g} = (\boldsymbol{\beta}^{\top}, \boldsymbol{z}^{\top}, \boldsymbol{u}^{\top})^{\top}$,

$$\theta(\boldsymbol{f}) = \sum_{i=1}^{K} \|\boldsymbol{\beta}_{i\cdot}\|_1 + \delta_{\mathcal{Z}_0(\boldsymbol{z})}, \quad \Omega = \mathbb{R}^p \times \mathbb{R}^p \times \mathbb{R}^p, \tag{B16}$$

and

$$F(\boldsymbol{g}) = \begin{pmatrix} -\boldsymbol{A}_1^{\top} \boldsymbol{u} \\ \vdots \\ -\boldsymbol{A}_K^{\top} \boldsymbol{u} \\ \sum_{i=1}^{K} \boldsymbol{A}_i \boldsymbol{\beta}_{i\cdot} - \boldsymbol{z} - \boldsymbol{X}^{\top} \boldsymbol{y} \end{pmatrix}. \tag{B17}$$

Note that the operator F defined in (B17) is monotone, because

$$(\boldsymbol{g}_1 - \boldsymbol{g}_2)^{\top} [F(\boldsymbol{g}_1) - F(\boldsymbol{g}_2)] \equiv 0, \quad \forall \boldsymbol{g}_1, \boldsymbol{g}_2 \in \Omega. \tag{B18}$$

Throughout, we denote by Ω^* the solution set of (B15), which is also the set of saddle points of the Lagrangian function (B14) of the model (B13).

B.2 Proof

From the iterative steps of the parallel PPA (with linearization) in (17) and Lemma 1, we have

$$
\begin{aligned}
\boldsymbol{\beta}_{i\cdot} \ \text{step:} \quad & \|\boldsymbol{\beta}_{i\cdot}\|_1 - \|\boldsymbol{\beta}_{i\cdot}^{t+1}\|_1 + (\boldsymbol{\beta}_{i\cdot} - \boldsymbol{\beta}_{i\cdot}^{t+1})^\top \left[-\boldsymbol{A}_i^\top \boldsymbol{u}^t + \mu \boldsymbol{A}_i^\top \boldsymbol{A}_i (\boldsymbol{\beta}_{i\cdot}^{t+1} - \boldsymbol{\beta}_{i\cdot}^t) \right. \\
& \left. + \boldsymbol{S}_i (\boldsymbol{\beta}_{i\cdot}^{t+1} - \boldsymbol{\beta}_{i\cdot}^t) \right] \geq 0, i \in \{1, \cdots, K\}; \\
\boldsymbol{z} \ \text{step:} \quad & \delta_{\mathcal{Z}_0(\boldsymbol{z})} - \delta_{\mathcal{Z}_0(\boldsymbol{z}^{t+1})} + (\boldsymbol{z} - \boldsymbol{z}^{t+1})^\top \left[\boldsymbol{u}^t + \mu(\boldsymbol{z}^{t+1} - \boldsymbol{z}^t) \right] \geq 0; \\
\boldsymbol{u} \ \text{step:} \quad & (\boldsymbol{u} - \boldsymbol{u}^{t+1})^\top \left[(\sum_{i=1}^K \boldsymbol{A}_i \boldsymbol{\beta}_{i\cdot}^{t+1} - \boldsymbol{z}^{t+1} - \boldsymbol{X}^\top \boldsymbol{y}) + \sum_{i=1}^K \boldsymbol{A}_i (\boldsymbol{\beta}_{i\cdot}^{t+1} - \boldsymbol{\beta}_{i\cdot}^t) \right. \\
& \left. - (\boldsymbol{z}^{t+1} - \boldsymbol{z}^t) + \frac{2}{\mu}(\boldsymbol{u}^{t+1} - \boldsymbol{u}^t) \right] = 0;
\end{aligned}
\tag{B19}
$$

where $\boldsymbol{S}_i = \eta \boldsymbol{I}_{p_i} - \mu \boldsymbol{A}_i^\top \boldsymbol{A}_i$, $\eta > \mathrm{eigen}(\mu \boldsymbol{A}^\top \boldsymbol{A})$, and $i \in \{1, \ldots, K\}$. The last equation is derived from the last equation of (17). By integrating the three equations in (B19) together, we can obtain

$$
\theta(\boldsymbol{f}) - \theta(\boldsymbol{f}^{t+1}) + (\boldsymbol{g} - \boldsymbol{g}^{t+1})^\top F(\boldsymbol{g}^{t+1}) \geq (\boldsymbol{g} - \boldsymbol{g}^{t+1})^T \boldsymbol{H}(\boldsymbol{g}^t - \boldsymbol{g}^{t+1}), \ \forall \boldsymbol{g} \in \Omega, \tag{B20}
$$

where $\boldsymbol{g} = (\boldsymbol{\beta}^\top, \boldsymbol{z}^\top, \boldsymbol{u}^\top)^\top$, $\boldsymbol{f}$ is a component of $\boldsymbol{g}$ by definition, and $\boldsymbol{H}$ can be found in (B11). It is worth noting that if the $\boldsymbol{\beta}$ subproblem is not linearized, then the $\boldsymbol{H}$ matrix on the right side of the inequality above needs to be replaced with the $\boldsymbol{M}$ matrix in (B11).

Similarly, from the iterative steps of the improved parallel PPA (with linearization) in (21) and Lemma 1, it follows that

$$
\begin{aligned}
\boldsymbol{\beta}_{i\cdot} \ \text{step:} \quad & \|\boldsymbol{\beta}_{i\cdot}\|_1 - \|\boldsymbol{\beta}_{i\cdot}^{t+1}\|_1 + (\boldsymbol{\beta}_{i\cdot} - \boldsymbol{\beta}_{i\cdot}^{t+1})^\top \left[-\boldsymbol{A}_i^\top \boldsymbol{u}^t + \mu \boldsymbol{A}_i^\top \boldsymbol{A}_i (\boldsymbol{\beta}_{i\cdot}^{t+1} - \boldsymbol{\beta}_{i\cdot}^t) \right. \\
& \left. + \boldsymbol{S}_i' (\boldsymbol{\beta}_{i\cdot}^{t+1} - \boldsymbol{\beta}_{i\cdot}^t) \right] \geq 0, i \in \{1, \cdots, K\}; \\
\boldsymbol{z} \ \text{step:} \quad & \delta_{\mathcal{Z}_0(\boldsymbol{z})} - \delta_{\mathcal{Z}_0(\boldsymbol{z}^{t+1})} + (\boldsymbol{z} - \boldsymbol{z}^{t+1})^\top \left[\boldsymbol{u}^t + \mu(\boldsymbol{z}^{t+1} - \boldsymbol{z}^t) \right] \geq 0;
\end{aligned}
\tag{B21}
$$

$$
\begin{aligned}
\boldsymbol{u} \ \text{step:} \quad & (\boldsymbol{u} - \boldsymbol{u}^{t+1})^\top \left[(\sum_{i=1}^K \boldsymbol{A}_i \boldsymbol{\beta}_{i\cdot}^{t+1} - \boldsymbol{z}^{t+1} - \boldsymbol{X}^\top \boldsymbol{y}) + \sum_{i=1}^K \boldsymbol{A}_i (\boldsymbol{\beta}_{i\cdot}^{t+1} - \boldsymbol{\beta}_{i\cdot}^t) \right. \\
& \left. - (\boldsymbol{z}^{t+1} - \boldsymbol{z}^t) + \frac{K}{\mu}(\boldsymbol{u}^{t+1} - \boldsymbol{u}^t) \right] = 0;
\end{aligned}
$$

where $\boldsymbol{S}_i' = \eta_i \boldsymbol{I}_{p_i} - \mu \boldsymbol{A}_i^\top \boldsymbol{A}_i$, $\eta_i > \mathrm{eigen}(\mu \boldsymbol{A}_i^\top \boldsymbol{A}_i)$ and $i \in \{1, \ldots, K\}$. The last equation is derived from the last equation of (21). By integrating the three equations in (B21) together, we can obtain

$$
\theta(\boldsymbol{f}) - \theta(\boldsymbol{f}^{t+1}) + (\boldsymbol{g} - \boldsymbol{g}^{t+1})^\top F(\boldsymbol{g}^{t+1}) \geq (\boldsymbol{g} - \boldsymbol{g}^{t+1})^\top \boldsymbol{H}_K(\boldsymbol{g}^t - \boldsymbol{g}^{t+1}), \ \forall \boldsymbol{g} \in \Omega, \tag{B22}
$$

where $\boldsymbol{H}_K$ can be found in (B12). If the $\boldsymbol{\beta}$ subproblem is not linearized, then the $\boldsymbol{H}_K$ matrix on the right side of the inequality above needs to be replaced with the $\boldsymbol{M}_K$ matrix in (B12).

Since the weighted form does not alter the convexity of $\|\boldsymbol{\beta}_{i\cdot}\|_1$ and $\delta_{\mathcal{Z}_0(\boldsymbol{z})}$, the variational inequalities (B20) and (B22) also hold for the solution of the weighted ℓ_1-DS in step 2.2 of Algorithm 4. This is also the reason why Corollary 3.1 holds true.

B.2.1 Global convergence

In this subsection, we prove the global convergence of the algorithms for Theorem 2 and Theorem 3. Next, based on the iterative steps of (17) and (21), we will deduce that the sequence $\{\boldsymbol{g}^k\}$ has the contraction property. Reviewing (B20) and (B22), these two inequalities serve as the foundation for the subsequent analysis. We can consolidate these two equations into a single form, namely

$$\theta(\boldsymbol{f}) - \theta(\boldsymbol{f}^{t+1}) + (\boldsymbol{g} - \boldsymbol{g}^{t+1})^\top F(\boldsymbol{g}^{t+1}) \geq (\boldsymbol{g} - \boldsymbol{g}^{t+1})^\top \boldsymbol{H}_*(\boldsymbol{g}^t - \boldsymbol{g}^{t+1}), \quad \forall \boldsymbol{g} \in \Omega, \quad \text{(B23)}$$

where $\boldsymbol{g} = (\boldsymbol{f}^\top, \boldsymbol{u}^\top)^\top$, and $\boldsymbol{H}_* = \boldsymbol{H}$ for the parallel PPA in (17) and $\boldsymbol{H}_* = \boldsymbol{H}_K$ for the improved parallel PPA in (21).

Proposition 2. *For the sequence $\{\boldsymbol{g}^t\}$ generated by the parallel PPA and the improved parallel PPA, we have*

$$\|\boldsymbol{g}^{t+1} - \boldsymbol{g}^*\|_{\boldsymbol{H}_*}^2 \leq \|\boldsymbol{g}^t - \boldsymbol{g}^*\|_{\boldsymbol{H}_*}^2 - \|\boldsymbol{g}^{t+1} - \boldsymbol{g}^t\|_{\boldsymbol{H}_*}^2, \quad \forall \boldsymbol{g}^* \in \Omega^*. \quad \text{(B24)}$$

Proof. Let $\boldsymbol{g}$ in (B23) be $\boldsymbol{g}^*$ and with $\boldsymbol{f}^*$ is a component of $\boldsymbol{g}^*$, then we have,

$$(\boldsymbol{g}^{t+1} - \boldsymbol{g}^*)^\top \boldsymbol{H}_*(\boldsymbol{g}^t - \boldsymbol{g}^{t+1}) \geq \theta(\boldsymbol{f}^{t+1}) - \theta(\boldsymbol{f}^*) + (\boldsymbol{g}^{t+1} - \boldsymbol{g}^*)^\top F(\boldsymbol{g}^{t+1}).$$

Since $\boldsymbol{g}^*$ represents the optimal point (saddle point) in (B15) and $\boldsymbol{g}^{k+1} \in \Omega$, in accordance with (B18), it becomes evident that the right-hand side of the aforementioned equation is non-negative. Consequently, we derive the following inequality:

$$(\boldsymbol{g}^{t+1} - \boldsymbol{g}^*)^\top \boldsymbol{H}_*(\boldsymbol{g}^t - \boldsymbol{g}^{t+1}) \geq 0. \quad \text{(B25)}$$

Let $\boldsymbol{a} = (\boldsymbol{g}^t - \boldsymbol{g}^*)$ and $\boldsymbol{b} = (\boldsymbol{g}^{t+1} - \boldsymbol{g}^*)$, according to Lemma 2 and (B25), we can get (B24). $\qquad\square$

The aforementioned contraction property plays a pivotal role in ensuring the convergence of sequences. The assertions summarized in the following corollary are straightforward, grounded in the fact (B24).

Corollary B.1. *Let $\boldsymbol{g}^t$ be the sequence generated by the parallel PPA and the improved parallel PPA. Then, we have*

1. $\lim_{t\to\infty} \|\boldsymbol{g}^t - \boldsymbol{g}^{t+1}\|_{\boldsymbol{H}_*} = 0$;
2. *the sequence $\{\boldsymbol{g}^t\}$ is bounded;*
3. *for any optimal solution $\boldsymbol{g}^*$, the sequence $\{\|\boldsymbol{g}^t - \boldsymbol{g}^*\|_{\boldsymbol{H}_*}\}$ is monotonically non-increasing.*

The proof of sequence convergence derived from (B24) (or Corollary B.1) has been extensively documented in the literature, including Theorem 2 of He and Yuan (2018) and Theorem 4.1 of He et al. (2022). For completeness, we will include the detailed proof here.

Proposition 3. *The sequence $\{\boldsymbol{g}^t\}$ generated by the parallel PPA and the improved parallel PPA converges to a point $\boldsymbol{g}^\infty$ that belongs to Ω^*, which is the set of all solutions to the variational inequality given in (B15).*

Proof. According to Corollary B.1, the sequence $\{\boldsymbol{g}^t\}$ is bounded and

$$\lim_{t\to\infty} \|\boldsymbol{g}^t - \boldsymbol{g}^{t+1}\|_{\boldsymbol{H}_*} = 0. \tag{B26}$$

Let $\boldsymbol{g}^\infty$ be a cluster point of $\{\boldsymbol{g}^t\}$, and $\{\boldsymbol{g}^{t_j}\}$ be a subsequence that converges to $\boldsymbol{g}^\infty$. It follows from (B23) that

$$\boldsymbol{g}^{t_j+1} \in \Omega,\ \theta(\boldsymbol{f}) - \theta(\boldsymbol{f}^{t_j+1}) + (\boldsymbol{g} - \boldsymbol{g}^{t_j+1})^\top F(\boldsymbol{g}^{t_j+1}) \geq (\boldsymbol{g} - \boldsymbol{g}^{t_j+1})^\top \boldsymbol{H}_*(\boldsymbol{g}^{t_j} - \boldsymbol{g}^{t_j+1}),\ \forall \boldsymbol{g} \in \Omega.$$

Note that the matrix $\boldsymbol{H}_*$ is a positive definite matrix, it follows from the continuity of $\theta(\boldsymbol{f})$ and $F(\boldsymbol{g})$ that

$$\boldsymbol{g}^\infty \in \Omega, \quad \theta(\boldsymbol{f}) - \theta(\boldsymbol{f}^\infty) + (\boldsymbol{g} - \boldsymbol{g}^\infty)^\top F(\boldsymbol{g}^\infty) \geq 0, \quad \forall \boldsymbol{h} \in \Omega.$$

The above variational inequality indicates that $\boldsymbol{g}^\infty$ is a solution point of (B15), that is $\boldsymbol{g}^\infty \in \Omega^*$. Finally, due to (B24), we have

$$\|\boldsymbol{g}^{k+1} - \boldsymbol{g}^\infty\|_{\boldsymbol{H}_*} \leq \|\boldsymbol{g}^k - \boldsymbol{g}^\infty\|_{\boldsymbol{H}_*}$$

which implies that the sequence $\{\boldsymbol{g}^k\}$ converges to $\boldsymbol{g}^\infty$. The proof is complete. $\square$

The proposition above indicates that $\boldsymbol{w}^\infty \in \Omega^*$, meaning that $\boldsymbol{w}^\infty$ and $\boldsymbol{w}^*$ are essentially the same, but simply represented differently. Proposition 3 is derived from the contraction inequality (B24), which directly leads to the global convergence of Theorem 2 and Theorem 3.

B.2.2 Linear convergence rate

Here, we demonstrate that a worst-case convergence rate of $\mathcal{O}(1/T)$ in a non-ergodic sense of Algorithm 2 and Algorithm 3, which have been stated in Theorem 2 and Theorem 3. To do this, we first need to prove the following proposition.

Proposition 4. *For the sequence $\{\boldsymbol{g}^t\}$ generated by the parallel PPA and the improved parallel PPA, we have*

$$\|\boldsymbol{g}^t - \boldsymbol{g}^{t+1}\|_{\boldsymbol{H}_*}^2 \leq \|\boldsymbol{g}^{t-1} - \boldsymbol{g}^t\|_{\boldsymbol{H}_*}^2, \tag{B27}$$

where $\boldsymbol{H}_* = \boldsymbol{H}$ for the parallel PPA in (17) and $\boldsymbol{H}_* = \boldsymbol{H}_K$ for the improved parallel PPA in (21).

Proof. First, by setting $\boldsymbol{g} = \boldsymbol{g}^t$ in (B23), we obtain

$$\theta(\boldsymbol{f}^t) - \theta(\boldsymbol{f}^{t+1}) + (\boldsymbol{g}^t - \boldsymbol{g}^{t+1})^\top F(\boldsymbol{g}^{t+1}) \geq (\boldsymbol{g}^t - \boldsymbol{g}^{t+1})^\top \boldsymbol{H}_*(\boldsymbol{g}^t - \boldsymbol{g}^{t+1}). \qquad \text{(B28)}$$

Note that (B23) is also true for $t := t + 1$. Thus, we also have

$$\theta(\boldsymbol{f}) - \theta(\boldsymbol{f}^t) + (\boldsymbol{g} - \boldsymbol{g}^t)^\top F(\boldsymbol{g}^t) \geq (\boldsymbol{g} - \boldsymbol{g}^t)^\top \boldsymbol{H}_*(\boldsymbol{g}^{t-1} - \boldsymbol{g}^t), \quad \forall \boldsymbol{g} \in \Omega, \qquad \text{(B29)}$$

Setting $\boldsymbol{g} = \boldsymbol{g}^{t+1}$ in the above inequality, we obtain

$$\theta(\boldsymbol{f}^{t+1}) - \theta(\boldsymbol{f}^t) + (\boldsymbol{g}^{t+1} - \boldsymbol{g}^t)^\top F(\boldsymbol{g}^t) \geq (\boldsymbol{g}^{t+1} - \boldsymbol{g}^t)^\top \boldsymbol{H}_*(\boldsymbol{g}^{t-1} - \boldsymbol{g}^t), \quad \forall \boldsymbol{g} \in \Omega. \qquad \text{(B30)}$$

By adding (B28) and (B30) , and utilizing the monotonicity of F in (B18), we obtain

$$(\boldsymbol{g}^t - \boldsymbol{g}^{t+1})^\top \boldsymbol{H}_* \left[(\boldsymbol{g}^{t-1} - \boldsymbol{g}^t) - (\boldsymbol{g}^t - \boldsymbol{g}^{t+1}) \right] \geq 0. \qquad \text{(B31)}$$

Let $\boldsymbol{a} = (\boldsymbol{g}^{t-1} - \boldsymbol{g}^t)$ and $\boldsymbol{b} = (\boldsymbol{g}^t - \boldsymbol{g}^{t+1})$, according to Lemma 2 and (B31), we can get

$$\|\boldsymbol{g}^t - \boldsymbol{g}^{t+1}\|_{\boldsymbol{H}_*}^2 \leq \|\boldsymbol{g}^{t-1} - \boldsymbol{g}^t\|_{\boldsymbol{H}_*}^2 - \|(\boldsymbol{g}^{t-1} - \boldsymbol{g}^t) - (\boldsymbol{g}^t - \boldsymbol{g}^{t+1})\|_{\boldsymbol{H}_*}^2$$

Since $\boldsymbol{H}_*$ is a positive-definite matrix, the assertion in (B27) follows immediately. $\quad\square$

Now, with (B24) and (B27), we can establish the worst-case $\mathcal{O}\left(1/T\right)$ convergence rate in a nonergodic sense for the parallel PPA and the improved parallel PPA (Linear convergence rate in Theorem 2 and Theorem 3).

Proposition 5. *Let $\{\boldsymbol{g}^t\}$ be the sequence generated by the parallel PPA and the improved parallel PPA. For any integer $T > 0$, we have*

$$\|\boldsymbol{g}^T - \boldsymbol{g}^{T+1}\|_{\boldsymbol{H}}^2 \leq \frac{1}{(T+1)} \|\boldsymbol{g}^0 - \boldsymbol{g}^*\|_{\boldsymbol{H}}^2. \qquad \text{(B32)}$$

Proof. First, it follows from (B24) that

$$\|\boldsymbol{g}^{t+1} - \boldsymbol{g}^t\|_{\boldsymbol{H}_*}^2 \leq \|\boldsymbol{g}^t - \boldsymbol{g}^*\|_{\boldsymbol{H}_*}^2 - \|\boldsymbol{g}^{t+1} - \boldsymbol{g}^*\|_{\boldsymbol{H}_*}^2, \quad \forall \boldsymbol{g}^* \in \Omega^*. \qquad \text{(B33)}$$

Summing over $t = 0, 1, \ldots, T$ yields the following

$$\sum_{t=0}^{T} \|\boldsymbol{g}^{t+1} - \boldsymbol{g}^t\|_{\boldsymbol{H}_*}^2 \leq \|\boldsymbol{g}^0 - \boldsymbol{g}^*\|_{\boldsymbol{H}_*}^2, \quad \forall \boldsymbol{g}^* \in \Omega^*. \qquad \text{(B34)}$$

Combining with the conclusion from (B27), we obtain

$$\|\boldsymbol{g}^{T+1} - \boldsymbol{g}^T\|_{\boldsymbol{H}_*}^2 \leq \frac{1}{T+1} \|\boldsymbol{g}^0 - \boldsymbol{g}^*\|_{\boldsymbol{H}_*}^2, \quad \forall \boldsymbol{g}^* \in \Omega^*. \qquad \text{(B35)}$$

$\square$

Appendix C   Supplementary experiments

In this section, we will supplement the additional experiments mentioned in Sections 4.1 and 4.2 regarding SCAD-DS and MCP-DS. Since both this paper and Wen et al. (2024) handle nonconvex regularizers using the LLA (Local Linear Approximation, Zou and Li (2008)) method, this implies that the iterative algorithm will require more iterations to solve the nonconvex DS model compared to the ℓ_1-DS model. Consequently, in this section, the upper limit for the number of iterations is set to 1000, rather than 500 in Section 4.

C.1   Supplementary experiments for Section 4.1

The generation of $\boldsymbol{X}$, $\boldsymbol{y}$, and $\boldsymbol{\beta}^*$, as well as the selection of initial values follow the same procedure as outlined in Section 4.1. The sole distinction lies in the focus of this subsection, which is the solution of nonconvex DS models. The outcomes of the numerical experiments are summarized in Table C1, highlighting that both parallel PPA methods exhibit competitiveness when compared to TADMM in addressing nonconvex DS models.

Table C1 Comparison of various algorithms for solving nonconvex DS models in nonparallel environments with data scale $(n, p) = (1000, 10,000)$.

Mehod	SCAD-DS						
	ρ	ℓ_1 error	ℓ_2 error	Model	FP	NI	Time(s)
	0.1	3.29(0.125)	0.77(0.012)	0.66(0.010)	7.32(0.88)	1000+(0.00)	2000+(0.0)
TADMM	0.5	3.22(0.118)	0.75(0.011)	0.62(0.010)	7.24(0.82)	1000+(0.00)	2000+(0.0)
	0.9	3.47(0.132)	0.85(0.015)	0.71(0.011)	8.11(0.91)	1000+(0.00)	2000+(0.0)
	0.1	1.09(0.009)	0.22(0.004)	0.15(0.002)	5.32(0.42)	479.6(39.7)	691.3(40.3)
PPA	0.5	1.15(0.012)	0.25(0.004)	0.11(0.002)	5.66(0.57)	483.7(44.3)	714.2(49.2)
	0.9	1.35(0.014)	0.28(0.005)	0.19(0.003)	6.16(0.61)	506.2(49.1)	732.1(51.2)
Method	MCP-DS						
	ρ	ℓ_1 error	ℓ_2 error	Model	FP	NI	Time(s)
	0.1	3.31(0.129)	0.82(0.014)	0.63(0.009)	6.79(0.72)	1000+(0.00)	2000+(0.0)
TADMM	0.5	3.19(0.116)	0.79(0.012)	0.59(0.008)	7.15(0.75)	1000+(0.00)	2000+(0.0)
	0.9	3.60(0.141)	0.84(0.014)	0.68(0.010)	7.44(0.81)	1000+(0.00)	2000+(0.0)
	0.1	1.11(0.010)	0.27(0.006)	0.17(0.003)	4.82(0.38)	483.5(39.9)	702.4(43.2)
PPA	0.5	1.18(0.012)	0.30(0.007)	0.15(0.002)	4.96(0.49)	488.2(42.2)	709.3(46.7)
	0.9	1.50(0.015)	0.31(0.007)	0.21(0.004)	5.23(0.53)	508.7(50.3)	736.0(53.5)

C.2   Supplementary experiments for Section 4.2

The generation of $\boldsymbol{X}$, $\boldsymbol{y}$, and $\boldsymbol{\beta}^*$, as well as the selection of initial values, follows the same procedure outlined in Section 4.2. The only distinction is that this subsection focuses on the solution of nonconvex DS models. The results of the numerical experiments are summarized in Table C2, demonstrating that both parallel PPA methods are competitive with TADMM in addressing nonconvex DS models. It is worth noting that both PPPA and IPPPA exhibit more stable solutions compared to TADMM as the number of matrix partitions increases. In particular, PPPA, due to its partition insensitivity, yields solutions that remain unchanged with variations in K.

Table C2 Comparison of parallel environment for solving nonconvex DS models with $s = 10$.

	Method	SCAD-DS				MCP-DS			
	K	AE	FP	Ite	Time(s)	AE	FP	Ite	Time(s)
TADMM	5	0.453(0.151)	81.3(8.51)	1000+(0.00)	1241.4(43.5)	0.461(0.160)	83.7(8.63)	1000+(0.00)	1238.5(44.1)
	10	0.528(0.165)	86.4(9.07)	1000+(0.00)	721.6(26.8)	0.493(0.168)	85.5(8.98)	1000+(0.00)	711.2(25.7)
	20	0.545(0.179)	92.3(10.3)	1000+(0.00)	434.8(15.7)	0.572(0.182)	93.4(10.5)	1000+(0.00)	456.8(14.9)
PPPA	5	**0.052(0.008)**	47.32(5.6)	**533.3(38.8)**	**299.2(25.4)**	**0.058(0.009)**	**39.87(4.9)**	**527.2(35.6)**	**293.5(24.2)**
	10	**0.052(0.008)**	47.32(5.6)	**533.3(38.8)**	**183.5(17.1)**	**0.058(0.009)**	**39.87(4.9)**	**527.2(35.6)**	**176.1(16.5)**
	20	**0.052(0.008)**	47.32(5.6)	**533.3(38.8)**	**137.0(11.2)**	**0.058(0.009)**	**39.87(4.9)**	**527.2(35.6)**	**124.8(10.7)**
IPPPA	5	0.061(0.009)	**47.24(5.8)**	541.1(39.2)	310.1(26.4)	0.063(0.010)	45.35(5.2)	543.5(40.3)	305.2(25.8)
	10	0.065(0.011)	49.62(6.1)	557.8(40.4)	192.7(16.3)	0.067(0.011)	47.27(6.0)	559.1(42.7)	187.0(17.3)
	20	0.069(0.012)	50.19(6.6)	572.5(42.8)	151.3(12.6)	0.072(0.012)	52.24(6.5)	567.8(43.8)	149.4(11.7)

C.3   Supplementary experiments for different types of noise

As suggested by a reviewer, we have added the numerical performance of the algorithm in solving DS under different types of noise other than Gaussian noise to demonstrate the algorithm's adaptability. Here, ϵ represents the error vector, where each component is independently and identically distributed, following a mixed normal distribution ($0.4\mathcal{N}(-3,4)+0.6\mathcal{N}(2,1)$), $t(2.5)$ distribution, and Cauchy distribution. The data scale is the same as Table C2, and $K = 10$. We have placed the numerical results in Table C3.

Table C3 Comparison of parallel environment for solving nonconvex DS models with different types of noise.

	Method	SCAD-DS				MCP-DS			
	ϵ	AE	FP	Ite	Time(s)	AE	FP	Ite	Time(s)
TADMM	Mixed normal	0.650(0.200)	105.5(12.0)	1000+(0.00)	480.0(18.0)	0.680(0.210)	107.0(12.5)	1000+(0.00)	505.0(17.0)
	$t(2.5)$	0.720(0.220)	112.8(13.2)	1000+(0.00)	515.6(19.8)	0.750(0.230)	114.5(13.8)	1000+(0.00)	538.5(18.7)
	Cauchy	1.623(0.421)	216.7(31.4)	1000+(0.00)	513.6(20.4)	1.639(0.399)	223.1(30.8)	1000+(0.00)	546.3(21.2)
PPPA	Mixed normal	**0.055(0.009)**	**50.15(6.0)**	**560.2(41.5)**	**145.2(12.0)**	**0.061(0.010)**	**42.32(5.2)**	**554.6(38.2)**	**132.7(11.4)**
	$t(2.5)$	**0.058(0.010)**	**53.00(6.4)**	**590.0(44.5)**	**153.5(12.8)**	**0.064(0.011)**	**44.80(5.6)**	**584.0(41.0)**	**140.8(12.2)**
	Cauchy	**0.090(0.015)**	80.00(9.5)	900.0(65.0)	**220.0(18.5)**	**0.100(0.017)**	70.00(8.2)	920.0(60.0)	**200.0(17.5)**
IPPPA	Mixed normal	0.075(0.013)	57.00(7.0)	650.0(46.5)	220.0(18.8)	0.077(0.013)	54.30(6.9)	645.0(49.1)	215.0(19.9)
	$t(2.5)$	0.076(0.013)	55.10(6.7)	621.4(44.4)	217.1(18.4)	0.077(0.013)	53.04(6.6)	622.7(47.0)	210.1(19.4)
	Cauchy	0.112(0.020)	**79.8(9.2)**	**860.3(62.1)**	230.1(25.5)	0.116(0.020)	77.12(8.9)	**870.20(55.2)**	285.2(26.8)

The numerical results in Table C3 indicate that when the error vector ϵ follows a sub-Gaussian distribution (both the mixed normal distribution and the $t(2.5)$ distribution are sub-Gaussian), the nonconvex DS methoda still exhibit acceptable numerical performance, though it is slightly inferior to the case of the Gaussian distribution presented in Table C2. When the error vector ϵ follows the Cauchy distribution, the numerical results deteriorate significantly. This is not due to the limitations of our algorithm, but rather because the statistical properties of the DS assume that the error distribution follows a Gaussian distribution or sub-Gaussian distribution (see Candès and Tao (2007b) and Wen et al. (2024)). These distributions possess finite moments and satisfy properties such as concentration inequalities. Under these

assumptions, it can be proven that the estimates of the Dantzig selector exhibit favorable properties, including consistency and convergence rates. Nevertheless, due to the heavy-tailed nature of the Cauchy distribution and the absence of finite moments, the original proofs of consistency and convergence rates become invalid. This is because these proofs rely on the moment conditions of the error terms and the concentration inequalities.

References

Becker, S., Candès, E.J., Grant, M.C.: Templates for convex cone problems with applications to sparse signal recovery. Mathematical Programming Computation **3**, 165–218 (2010)

Bickel, P.J.: Discussion: The Dantzig Selector: Statistical Estimation When p Is Much Larger than n. The Annals of Statistics **35**(6), 2352–2357 (2007)

Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed Optimization and Statistical Learning Via the Alternating Direction Method of Multipliers. Foundation and Trends in Machine Learning **3**(1), 1–122 (2010)

Bickel, P.J., Ritov, Y., Tsybakov, A.: Simultaneous analysis of lasso and dantzig selector. Annals of Statistics **37**, 1705–1732 (2008)

Beck, A., Teboulle, M.: A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. SIAM Journal on Imaging Sciences **2**(1), 183–202 (2009)

Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, ??? (2004)

Chatterjee, S., Chen, S., Banerjee, A.: Generalized Dantzig Selector: Application to the k-support norm. In: Neural Information Processing Systems, vol. 3, pp. 1934–1942 (2014)

Cai, X., Gu, G., He, B., Yuan, X.: A proximal point algorithm revisit on the alternating direction method of multipliers. Science China Mathematics **56**(10), 2179–2186 (2013)

Chen, C., He, B., Ye, Y., Yuan, X.: The direct extension of ADMM for multi-block convex minimization problems is not necessarily convergent. Mathematical Programming **155**(1-2), 57–79 (2016)

Cai, T.T., Lv, J.: Discussion: The Dantzig Selector: Statistical Estimation When p Is Much Larger than n. The Annals of Statistics **35**(6), 2365–2369 (2007)

Cai, Z., Li, C., Wen, J., Yang, S.: Asset splitting algorithm for ultrahigh dimensional portfolio selection and its theoretical property. Journal of Econometrics **239**(2), 105291 (2024)

Chambolle, A., Pock, T.: A First-Order Primal-Dual Algorithm for Convex Problems with Applications to Imaging. Journal of Mathematical Imaging and Vision **40**, 120–145 (2011)

Candès, E., Tao, T.: Rejoinder: The Dantzig Selector: Statistical Estimation When p Is Much Larger than n. The Annals of Statistics **35**(6), 2392–2404 (2007)

Candès, E., Tao, T.: The Dantzig selector: Statistical estimation when p is much larger than n. The Annals of Statistics **35**(6), 2313–2351 (2007)

Dicker, L., Lin, X.: Parallelism, uniqueness, and large-sample asymptotics for the Dantzig selector. Canadian Journal of Statistics **41**(1), 23–35 (2013)

Efron, B., Hastie, T., Tibshirani, R.: Discussion: The Dantzig selector: Statistical estimation when p is much larger than n. The Annals of Statistics **35**(6), 2358–2364 (2007)

Friedman, J.H., Hastie, T.J., Tibshirani, R.: A note on the group lasso and a sparse group lasso. arXiv: Statistics Theory (2010)

Fan, J., Li, R.: Variable Selection via Nonconcave Penalized Likelihood and its Oracle Properties. Journal of the American Statistical Association **96**(456), 1348–1360 (2001)

Fang, S., Liu, Y.-J., Xiong, X.: Efficient Sparse Hessian-Based Semismooth Newton Algorithms for Dantzig Selector. SIAM J. Sci. Comput. **43**, 4147–4171 (2021)

Fan, Y., Lin, N., Yin, X.: Penalized Quantile Regression for Distributed Big Data Using the Slack Variable Representation. Journal of Computational and Graphical Statistics **30**(3), 557–565 (2021)

Friedlander, M.P., Saunders, M.A.: Discussion: The Dantzig selector: Statistical estimation when p is much larger than n. The Annals of Statistics **35**(6), 2385–2391 (2007)

Gu, Y., Fan, J., Kong, L., Ma, S., Zou, H.: ADMM for High-Dimensional Sparse Penalized Quantile Regression. Technometrics **60**(3), 319–331 (2018)

Gu, G., He, B., Yuan, X.: Customized proximal point algorithms for linearly constrained convex minimization and saddle-point problems : a uniform approach. Computational Optimization and Applications **59**, 135–161 (2014)

Ge, H., Li, P.: The Dantzig selector: recovery of signal via $\ell_1 - \alpha\ell_2$ minimization. Inverse Problems **38**(1), 015006 (2021)

Golub, T.R., Slonim, D.K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J.P., Coller, H., Loh, M.L., Downing, J.R., Caligiuri, M.A., Bloomfield, C.D., Lander, E.S.: Molecular classification of cancer: Class discovery and class prediction by gene

expression monitoring. Science **286**(5439), 531–537 (1999)

He, H., Cai, X., Han, D.: A fast splitting method tailored for Dantzig selector. Computational Optimization and Applications **62**, 347–372 (2015)

He, B., Ma, F., Xu, S., Yuan, X.: A rank-two relaxed parallel splitting version of the augmented Lagrangian method with step size in (0,2) for separable convex programming. Math. Comput. **92**, 1633–1663 (2022)

He, B., Ma, F., Yuan, X.: Optimally Linearizing the Alternating Direction Method of Multipliers for Convex Programming. Computational Optimization and Applications **75**, 361–388 (2020)

He, H., Xu, H.-K.: Splitting methods for split feasibility problems with application to Dantzig selectors. Inverse Problems **33** (2017)

He, B., Yuan, X.: On the $O(1/n)$ Convergence Rate of the Douglas–Rachford Alternating Direction Method. SIAM Journal on Numerical Analysis **50**(2), 700–709 (2012)

He, B., Yuan, X.: A class of ADMM-based algorithms for three-block separable convex programming. Computational Optimization and Applications **70**, 791–826 (2018)

James, G.M., Radchenko, P., Lv, J.: DASSO: Connections Between the Dantzig Selector and Lasso. Journal of the Royal Statistical Society Series B: Statistical Methodology **71**(1), 127–142 (2008)

Li, Y., Dicker, L.H., Zhao, S.D.: The Dantzig Selector for Censored Linear Regression Models. Statistica Sinica **24 1**, 251–2568 (2014)

Lin, Z., Fang, C., Li, H.: Alternating Direction Method of Multipliers for Machine Learning. Springer, Singapore (2022)

Li, C., Li, R., Wen, J., Yang, S., Zhan, X.: Regularized Linear Programming Discriminant Rule with Folded Concave Penalty for Ultrahigh-Dimensional Data. Journal of Computational and Graphical Statistics **32**(3), 1074–1082 (2023)

Li, X., Mo, L., Yuan, X., Zhang, J.: Linearized Alternating Direction Method of Multipliers for Sparse Group and Fused LASSO Models. Computational Statistics & Data Analysis **79**, 203–221 (2014)

Lu, Z., Pong, T.K., Zhang, Y.: An alternating direction method for finding Dantzig selectors. Computational Statistics & Data Analysis **56**(12), 4037–4046 (2012)

Liang, R., Wu, X., Zhang, Z.: Linearized Alternating Direction Method of Multipliers for Elastic-net Support Vector Machines. Pattern Recognition **148**, 110134 (2024)

Liu, H., Zhang, J., Jiang, X., Liu, J.: The Group Dantzig Selector. In: International

Conference on Artificial Intelligence and Statistics, vol. 9, pp. 461–468 (2010)

Li, X., Zhao, T., Yuan, X., Liu, H.: The flare Package for High Dimensional Linear Regression and Precision Matrix Estimation in R. Journal of Machine Learning Research **16**(18), 553–557 (2015)

Mao, X., He, H., Xu, H.-K.: A partially proximal linearized alternating minimization method for finding Dantzig selectors. Computational and Applied Mathematics **40** (2021)

Meinshausen, N., Rocha, G., Yu, B.: Discussion: A Tale of Three Cousins: Lasso, L2Boosting and Dantzig. The Annals of Statistics **35**(6), 2373–2384 (2007)

Parikh, N., Boyd, S.P.: Proximal Algorithms. Now Foundations and Trends **1**, 127–239 (2013)

Park, S., He, X., Zhou, S.: Dantzig-type penalization for multiple quantile regression with high dimensional covariates. Statistica Sinica **27**, 1619–1638 (2017)

Prater, A., Shen, L., Suter, B.W.: Finding Dantzig selectors with a proximity operator based fixed-point algorithm. Computational Statistics & Data Analysis **90**, 36–46 (2015)

Ritov, Y.: Discussion: The Dantzig Selector: Statistical Estimation When p Is Much Larger than n. The Annals of Statistics **35**(6), 2370–2372 (2007)

Sun, D., Toh, K.-C., Yang, L.: A Convergent 3-Block SemiProximal Alternating Direction Method of Multipliers for Conic Programming with 4-Type Constraints. SIAM Journal on Optimization **25**(2), 882–915 (2015)

Tibshirani, R.: Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society, Series B **58**(1) (1996)

Wang, H.: Forward regression for ultra-high dimensional variable screening. Journal of the American Statistical Association **104**(488), 1512–1524 (2009)

Wu, X., Jiang, J., Zhang, Z.: Partition-Insensitive Parallel ADMM Algorithm for High-dimensional Linear Models. arXiv (2023)

Wu, X., Liang, R., Zhang, Z., Cui, Z.: A unified consensus-based parallel ADMM algorithm for high-dimensional regression with combined regularizations. Computational Statistics & Data Analysis **203**, 108081 (2025)

Wu, X., Liang, R., Zhang, Z., Cui, Z.: Multi-block linearized alternating direction method for sparse fused Lasso modeling problems. Applied Mathematical Modelling **137**, 115694 (2025)

Wu, X., Ming, H., Zhang, Z., Cui, Z.: Multi-block Alternating Direction Method of

Multipliers for Ultrahigh Dimensional Quantile Fused Regression. Computational Statistics & Data Analysis **192**, 107901 (2024)

Wang, X., Yuan, X.: The Linearized Alternating Direction Method of Multipliers for Dantzig Selector. Siam Journal on Scientific Computing **34**(5), 2792–2811 (2012)

Wen, J., Yang, S., Wang, C., Jiang, Y., Li, R.: Feature-splitting Algorithms for Ultrahigh Dimensional Quantile Regression. Journal of Econometrics, 105426 (2023)

Wen, J., Yang, S., Zhao, D.: Nonconvex Dantzig selector and its parallel computing algorithm. Statistics and Computing **34**, 1573–1375 (2024)

Zhang, C.: Nearly Unbiased Variable Selection Under Minimax Concave Penalty. Annals of Statistics **38**(2), 894–942 (2010)

Zou, H., Li, R.: One-step Sparse Estimates in Nonconcave Penalized Likelihood Models. The Annals of Statistics **36**(4), 1509–1533 (2008)

Zou, H.: The Adaptive Lasso and Its Oracle Properties. Journal of the American Statistical Association **101**, 1418–1429 (2006)