HiQR: An efficient algorithm for high dimensional quadratic regression with penalties

Cheng Wang[a,*], Haozhe Chen[a], Binyan Jiang[b]

[a] *School of Mathematical Sciences, MOE-LSC, Shanghai Jiao Tong University, Shanghai 200240, China.*
[b] *Department of Applied Mathematics, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong.*

Abstract

This paper investigates the efficient solution of penalized quadratic regressions in high-dimensional settings. We propose a novel and efficient algorithm for ridge-penalized quadratic regression that leverages the matrix structures of the regression with interactions. Building on this formulation, we develop an alternating direction method of multipliers (ADMM) framework for penalized quadratic regression with general penalties, including both single and hybrid penalty functions. Our approach greatly simplifies the calculations to basic matrix-based operations, making it appealing in terms of both memory storage and computational complexity.

Keywords: ADMM, LASSO, quadratic regression, ridge regression

1. Introduction

Quadratic regression, which extends linear regression by accounting for interactions between covariates, has found widespread applications across various disciplines. However, as the complexity of the interactions increases quadratically with the number of variables, parameter estimation becomes increasingly challenging for problems with large or even moderate dimensionality. A surge of methodologies have been developed in the past decade to tackle the high dimensionality challenge under different structural assumptions; see for example **???????** and **?**, among others.

*Corresponding author
Email addresses: `chengwang@sjtu.edu.cn` (Cheng Wang), `bob150115chz@sjtu.edu.cn` (Haozhe Chen), `by.jiang@polyu.edu.hk` (Binyan Jiang)

Given the observations $(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R},\ i = 1, \ldots, n$, we consider a general penalized quadratic regression model expressed as

$$\operatorname*{arg\,min}_{\mathbf{B}=\mathbf{B}^\top, \mathbf{B}\in\mathbb{R}^{p\times p}} \frac{1}{2n}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^\top \mathbf{B}\mathbf{x}_i)^2 + f(\mathbf{B}), \tag{1}$$

where $\mathbf{B} = (B_{jk})_{p\times p}$ denotes a symmetric matrix of parameters, and $f(\cdot)$ is a convex penalty function. Typically, the first element of $\mathbf{x}_i$ is a constant 1, allowing for the capture of the intercept, linear effect, and interaction effect through $B_{1,1,}$, $\{B_{1,i}, i = 2, \ldots, p\}$, and $\{B_{i,j}, 2 \le i \le j \le p\}$, respectively.

Without the penalty $f(\mathbf{B})$, the mean squared error is:

$$\frac{1}{2n}\sum_{i=1}^{n}\left(y_i - B_{11} - \sum_{j=2}^{p}(B_{1j} + B_{j1})x_{ij} - \sum_{j,k=2}^{p} B_{jk}x_{ij}x_{ik}\right)^2.$$

The penalty term $f(\mathbf{B})$ is introduced to impose different structures on the parameter matrix $\mathbf{B}$ depending on the application scenario. For instance, in gene-gene interaction detection where the number of genes is typically large and the interactions related to the response are sparse, the ℓ_1 penalty $f(\mathbf{B}) = \lambda\|\mathbf{B}\|_1$ is often used to induce sparsity in $\mathbf{B}$. The resulting model is called the all-pairs LASSO by **?**. In addition to sparsity, researchers have also considered heredity, where the existence of the interaction effect $B_{j,k}$ depends on the existence of its parental linear effects $B_{1,j}, B_{1,k}$. Specifically, we have:

$$\begin{aligned} \text{strong heredity: } B_{j,k} \neq 0 &\Rightarrow B_{1j} \neq 0 \text{ and } B_{1k} \neq 0, \\ \text{weak heredity: } B_{j,k} \neq 0 &\Rightarrow B_{1j} \neq 0 \text{ or } B_{1k} \neq 0. \end{aligned}$$

Several penalty functions are proposed in the literature to enforce these heredity structures, including those proposed by **?**, **?**, **?**, **?**, **?**, **?**, and **?**, among others. In addition to sparsity and heredity, we can also introduce the nuclear norm penalty to impose a low rank structure in $\mathbf{B}$, and hybrid penalties to impose more than one structure. Further details will be provided in Section 3.

A naive approach to solving the penalized quadratic regression model (1) is to use vectorization. We define

$$\mathbf{z}_i = \mathbf{x}_i \otimes \mathbf{x}_i \in \mathbb{R}^{p^2\times 1},$$

where $\otimes$ denotes the Kronecker product, and write

$$\mathbf{b} = \operatorname{vec}(\mathbf{B}) \in \mathbb{R}^{p^2\times 1},$$

where vec$(\cdot)$ denotes the vectorization of a matrix. We can then obtain the following equivalent form of (1):

$$\arg\min_{\mathbf{b}} \frac{1}{2n} \sum_{i=1}^{n} (y_i - \mathbf{z}_i^\top \mathbf{b})^2 + f(\mathbf{b}).$$

Therefore, the penalized quadratic regression problem (1) can be reformulated as a penalized linear model with $p(p+1)/2$ features. From a theoretical perspective, we can use this formulation together with the classical theory for high-dimensional regularized M-estimators (**?**, Chapter 9). Detailed theoretical analyses of the consistency of the penalized quadratic regression model can be found in **?** and the references therein. However, from a computational perspective, many algorithms do not scale well with a large p, since the number of parameters scales quadratically with the dimension p. Moreover, storing the design matrix and computer memory can also be expensive when vectorization is applied to the interaction variables. For example, computing an all-pairs LASSO with $n = 1000$ and $p = 1000$ on a personal computer can cause the well-known algorithm *glmnet* (**?**) to break down due to out-of-memory errors. Specifically, the feature matrix of order $10^3 \times 10^6$ has a memory size of about 8GB.

To address the computational challenges associated with high-dimensional penalized quadratic regression, several two-stage methods have been proposed in the literature (**?????**, e.g.,). These methods are computationally efficient and have been proven to be consistent under some structural assumptions, which can reduce the computational complexity via a feature selection procedure in the first stage. However, in this paper, we do not consider any of these structures, and our main goal is to develop efficient algorithms for solving the general penalized quadratic regression model (1) directly. Intuitively, penalized quadratic regression is different from a common linear regression with $O(p^2)$ features because the data has a specific structure for interactions. In this work, we leverage this structure in the algorithm and design an efficient framework for the general penalized quadratic regression problem. In previous work, **?** and **?** also developed efficient formulas for the matrix parameter under a factor model. However, their procedures greatly rely on the distributional assumptions and cannot be extended to general cases. In contrast, our approach does not require any distributional assumptions and can be applied to a wide range of high-dimensional data.

In this work, we study the original optimization problem (1) and design the algorithm from the viewpoint of matrix forms. To the best of our knowledge, this is the first algorithm for penalized quadratic regression that does not use vectorization and avoids any matrix operation of the $n \times p^2$ feature matrix. Our contributions are summarized as follows:

1. For ridge regression, we obtain an efficient solution for quadratic regression with a computational complexity of $O(np^2 + n^3)$.
2. To solve the general penalized quadratic regression problem for single non-smooth penalty and hybrid penalty functions, we propose an alternating direction method of multipliers (ADMM) algorithm. The algorithm is fully formulated with matrix forms, using only $p \times p$, $n \times p$, or $n \times n$ matrices, and has explicit formulas for the solutions in each iteration.
3. We have developed an R package for penalized quadratic regression. Compared to other existing solvers/packages, our algorithm is much more robust since we do not impose any structural assumptions such as heredity. Our algorithm is also appealing in both memory storage and computational cost, and can handle datasets with very high dimensions. This makes our package a useful tool for researchers and practitioners who need to analyze high-dimensional data using penalized quadratic regression.

The rest of the paper is organized as follows. In Section 2, we start with ridge-penalized quadratic regression and derive an efficient closed-form formula for the solution. In Section 3, we design an efficient ADMM algorithm for both single non-smooth penalty and hybrid penalty functions. We conduct simulations in Section 4 to illustrate the proposed algorithm. The developed R package "HiQR" is available on GitHub at `https://github.com/cescwang85/HiQR`.

2. Ridge regression

To facilitate the discussion, we introduce some notations first. For a real $p \times q$ matrix $\mathbf{A} = (A_{k,l})_{p\times q}$, we define:

$$\|\mathbf{A}\|_\infty \overset{\text{def}}{=} \max_{1\le k\le p, 1\le l\le q} |A_{k,l}|, \ \|\mathbf{A}\|_1 \overset{\text{def}}{=} \sum_{k=1}^{p}\sum_{l=1}^{q} |A_{k,l}|, \ \|\mathbf{A}\|_2^2 \overset{\text{def}}{=} \sum_{k=1}^{p}\sum_{l=1}^{q} |A_{k,l}|^2.$$

Denoting the singular values of $\mathbf{A}$ as $\sigma_1 \ge \cdots \sigma_p \ge 0$, the nuclear norm of $\mathbf{A}$ is defined as

$$\|\mathbf{A}\|_* = \sum_{i=1}^{p} \sigma_i.$$

We first consider the ridge regression for the quadratic regression, i.e.,

$$\text{Ridge QR:} \quad \mathop{\arg\min}_{\mathbf{B}=\mathbf{B}^\top, \mathbf{B}\in\mathbb{R}^{p\times p}} \frac{1}{2n}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^\top \mathbf{B}\mathbf{x}_i)^2 + \frac{\lambda}{2}\|\mathbf{B}\|_2^2. \tag{2}$$

where $\lambda > 0$ is a tuning parameter. Since the object function is convex in $\mathbf{B}$, the solution can be obtained by solving the following equation:

$$\frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i^\top \mathbf{B}\mathbf{x}_i - y_i)\mathbf{x}_i\mathbf{x}_i^\top + \lambda\mathbf{B} = \mathbf{0}_{p\times p}. \tag{3}$$

Denote $\mathbf{D} = \frac{1}{n}\sum_{i=1}^{n} y_i\mathbf{x}_i\mathbf{x}_i^\top$. Equation (3) can be equivalently written as:

$$\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i\mathbf{x}_i^\top \mathbf{B}\mathbf{x}_i\mathbf{x}_i^\top + \lambda\mathbf{B} = \mathbf{D}.$$

By applying vectorization to the above equation, we have:

$$\frac{1}{n}\sum_{i=1}^{n}\left\{(\mathbf{x}_i\mathbf{x}_i^\top)\otimes(\mathbf{x}_i\mathbf{x}_i^\top)\right\}\operatorname{vec}(\mathbf{B}) + \lambda\cdot\operatorname{vec}(\mathbf{B}) = \operatorname{vec}(\mathbf{D}),$$

and then the solution can be seen as:

$$\operatorname{vec}(\mathbf{B}) = \left\{\mathbb{X}\mathbb{X}^\top + \lambda\mathbf{I}_{p^2}\right\}^{-1}\operatorname{vec}(\mathbf{D}) = \left\{\mathbb{X}\mathbb{X}^\top + \lambda\mathbf{I}_{p^2}\right\}^{-1}\mathbb{X}\mathbf{Y}, \tag{4}$$

where

$$\mathbb{X} = \frac{1}{\sqrt{n}}(\mathbf{x}_1\otimes\mathbf{x}_1, \cdots, \mathbf{x}_n\otimes\mathbf{x}_n).$$

Note that $\mathbb{X}\mathbb{X}^\top$ is a $p^2\times p^2$ matrix, which can lead to a high computational complexity of $O(p^6)$ for direct calculation of its inverse. Moreover, storing such a large matrix when p is large is also impractical. Therefore, the naive algorithm that computes (4) directly is usually not applicable for high-dimensional quadratic regression.

Note that the rank of $\mathbb{X}$ is $\min\{n, p^2\}$, which can be much smaller than p^2 when $n \ll p$. To exploit the low-rank structure of $\mathbb{X}$, we can use the Woodbury matrix identity, which allows us to compute $(\mathbb{X}\mathbb{X}^\top + \lambda\mathbf{I}_{p^2})^{-1}$ more efficiently. Specifically, by applying the Woodbury identity, we have:

$$(\mathbb{X}\mathbb{X}^\top + \lambda\mathbf{I}_{p^2})^{-1} = \lambda^{-1}\mathbf{I}_{p^2} - \lambda^{-1}\mathbb{X}(\lambda\mathbf{I}_n + \mathbb{X}^\top\mathbb{X})^{-1}\mathbb{X}^\top. \tag{5}$$

The computational complexity is now been reduced to $O(n^2p^2 + n^3)$, where the n^2p^2 term is due to matrix multiplication and n^3 is the complexity of matrix inverse. The Woodbury identity has been widely used in many other algorithms, and it is sometimes referred to as the "shortcut-trick" for high-dimensional data ((**?**, section 4.2.4); **?**).

Another efficient technique to further reduce the computational cost is the implementation of the singular value decomposition(SVD) to $\mathbb{X}$ (**?**). Specifically, let $\mathbb{X} = \mathbf{U}\Lambda\mathbf{V}^\top$ be the thin SVD of $\mathbb{X}$. Together with (5), the solution (4) can be expressed as:

$$(\mathbb{X}\mathbb{X}^\top + \lambda\mathbf{I}_{p^2})^{-1}\mathbb{X}\mathbf{Y} = \mathbf{U}(\Lambda^2 + \lambda\mathbf{I})^{-1}\Lambda\mathbf{V}^\top\mathbf{Y}. \tag{6}$$

Here, the complexity of SVD is $O(n^2p^2)$, which can significantly reduce the computational complexity compared to the naive algorithm that computes (4) directly. However, for some large-scale problems, the reduction in computational complexity may still be insignificant.

In what follows, we will further exploit the special structure of the parameter matrix in quadratic regression and reduce the computational complexity to $O(np^2)$. Note that from (5) and the first equation of (4), we have:

$$\mathrm{vec}(\mathbf{B}) = \left\{\lambda^{-1}\mathbf{I}_{p^2} - \lambda^{-1}\mathbb{X}(\lambda\mathbf{I}_n + \mathbb{X}^\top\mathbb{X})^{-1}\mathbb{X}^\top\right\}\mathrm{vec}(\mathbf{D}).$$

Firstly, note that

$$\begin{aligned}\mathbb{X}^\top\mathbb{X} &= \begin{pmatrix}\frac{1}{\sqrt{n}}(\mathbf{x}_1 \otimes \mathbf{x}_1)^\top \\ \vdots \\ \frac{1}{\sqrt{n}}(\mathbf{x}_n \otimes \mathbf{x}_n)^\top\end{pmatrix}\left(\frac{1}{\sqrt{n}}\mathbf{x}_1 \otimes \mathbf{x}_1, \cdots, \frac{1}{\sqrt{n}}\mathbf{x}_n \otimes \mathbf{x}_n\right) \\ &= \frac{1}{n}\left((\mathbf{x}_i^\top\mathbf{x}_j)^2\right)_{n\times n} = n^{-1}(\mathbf{X}\mathbf{X}^\top) \circ (\mathbf{X}\mathbf{X}^\top),\end{aligned} \tag{7}$$

where $\circ$ is the Hadamard product and the complexity of the last equation is of order $O(n^2p)$. Secondly, note that

$$\begin{aligned}\mathbb{X}^\top\mathrm{vec}(\mathbf{D}) &= \begin{pmatrix}\frac{1}{\sqrt{n}}(\mathbf{x}_1 \otimes \mathbf{x}_1)^\top \\ \vdots \\ \frac{1}{\sqrt{n}}(\mathbf{x}_n \otimes \mathbf{x}_n)^\top\end{pmatrix}\mathrm{vec}(\mathbf{D}) = \begin{pmatrix}\frac{1}{\sqrt{n}}\mathbf{x}_1^\top\mathbf{D}\mathbf{x}_1 \\ \vdots \\ \frac{1}{\sqrt{n}}\mathbf{x}_n^\top\mathbf{D}\mathbf{x}_n\end{pmatrix} \\ &= \frac{1}{\sqrt{n}} \cdot \mathrm{diag}\left(\mathbf{X}\mathbf{D}\mathbf{X}^\top\right),\end{aligned} \tag{8}$$

where in the last equation the complexity is also reduced to $O(np^2)$. Lastly, denoting

$$\mathbf{w} = \frac{1}{\sqrt{n}}(\lambda \mathbf{I}_n + \mathbb{X}^\top \mathbb{X})^{-1}\mathbb{X}^\top \mathrm{vec}(\mathbf{D}) \in \mathbb{R}^n,$$

we have:

$$\begin{aligned}\mathbb{X}(\lambda \mathbf{I}_n + \mathbb{X}^\top \mathbb{X})^{-1}\mathbb{X}^\top \mathrm{vec}(\mathbf{D}) =& \sum_{k=1}^{n} w_k \mathbf{x}_k \otimes \mathbf{x}_k \\ =&\mathrm{vec}\left(\sum_{i=1}^{n} w_k \mathbf{x}_k \mathbf{x}_k^\top\right) = \mathrm{vec}\left(\mathbf{X}^\top \mathrm{diag}(\mathbf{w})\mathbf{X}\right),\end{aligned} \tag{9}$$

where the complexity of the last equation is also $O(np^2)$.

Your paragraph looks good! Here's a possible minor revision to improve the flow:

By combining equations (7)-(9), we can obtain a computationally efficient form for the explicit solution of the ridge-penalized quadratic regression (2). We summarize the results in the following proposition.

Proposition 2.1. *For a given tuning parameter $\lambda > 0$, the solution of the ridge-penalized quadratic regression problem* (2) *is given as:*

$$\widehat{\mathbf{B}} = \lambda^{-1}\mathbf{D} - \lambda^{-1}\mathbf{X}^\top \mathit{diag}\{\mathbf{w}\}\mathbf{X}, \tag{10}$$

where

$$\begin{aligned}\mathbf{X} =&(\mathbf{x}_1, \dots, \mathbf{x}_n), \quad \mathbf{D} = \frac{1}{n}\sum_{i=1}^{n} y_i \mathbf{x}_i \mathbf{x}_i^\top = \frac{1}{n}\mathbf{X}^\top \mathit{diag}(y_1, \cdots, y_n)\mathbf{X}, \\ \mathbf{w} =& \left\{\lambda \mathbf{I}_n + n^{-1}(\mathbf{X}\mathbf{X}^\top) \circ (\mathbf{X}\mathbf{X}^\top)\right\}^{-1} \mathit{diag}\left(\frac{1}{n}\mathbf{X}\mathbf{D}\mathbf{X}^\top\right).\end{aligned}$$

The computational complexity for calculating the close-form solution (10) is $O(np^2 + n^3)$, which is much more efficient than the forms given in (4) and (6) under the high-dimensional setting where $n \ll p$. In addition, the memory cost of the solution is also lower because it only requires components in the form of either an $n \times n$ matrix, an $n \times p$ matrix, or a $p \times p$ matrix. In next section, we will further extend our results obtained in this section to solve quadratic regression with other non-smooth penalties.

3. Non-smooth penalty and beyond

In this section, we consider the case where the penalty $f(\cdot)$ in the penalized quadratic regression (1) is possibly non-smooth. For example, we can consider setting $f(\mathbf{B}) = \lambda\|\mathbf{B}\|_1$ as in the all-pairs-LASSO, or $f(\mathbf{B}) = \lambda\|\mathbf{B}\|_*$ as in reduced rank regression. For high-dimensional quadratic regression, it is also attractive to introduce additional penalties to impose different structures simultaneously. For instance, we can combine the ℓ_1 norm and the nuclear norm to get a sparse and low-rank solution, i.e., $f(\mathbf{B}) = \lambda_1\|\mathbf{B}\|_1 + \lambda_2\|\mathbf{B}\|_*$. In the literature, several hybrid penalty functions are proposed for quadratic regression, and we summarize these hybrid penalties as follows.

- $\ell_1 + \ell_2$:

$$f(\mathbf{B}) = \lambda_1\|\mathbf{B}\|_1 + \lambda_2\sum_{k=2}^{p}\|\mathbf{B}_{\cdot,k}\|_2 + \lambda_2\sum_{k=2}^{p}\|\mathbf{B}_{k,\cdot}\|_2.$$

  See **?** and **?** for more details.

- $\ell_1 + \ell_\infty$:

$$f(\mathbf{B}) = \lambda_1\|\mathbf{B}\|_1 + \lambda_2\sum_{k=2}^{p}\|\mathbf{B}_{\cdot,k}\|_\infty + \lambda_2\sum_{k=2}^{p}\|\mathbf{B}_{k,\cdot}\|_\infty.$$

  See **?**.

- $\ell_1 + \ell_1/\ell_\infty$:

$$\begin{aligned} f(\mathbf{B}) &= \lambda_1\|\mathbf{B}\|_1 + \lambda_2\sum_{k=2}^{p}\max\{|\mathbf{B}_{1,k}|, \|\mathbf{B}_{-1,k}\|_1\} \\ &\quad +\lambda_2\sum_{k=2}^{p}\max\{|\mathbf{B}_{k,1}|, \|\mathbf{B}_{k,-1}\|_1\} \end{aligned}$$

  See **?** and **?**.

- $\ell_1 + \ell_*$:

$$f(\mathbf{B}) = \lambda_1\|\mathbf{B}\|_1 + \lambda_2\|\mathbf{B}\|_*.$$

  See **?** and the references therein.

We remark that all of these penalties are formulated in a symmetric pattern, i.e., $f(\mathbf{B}) = f(\mathbf{B}^\top)$. Thus, the final solution will be a symmetric matrix. Utilizing the efficient formulation we obtained in Proposition 2.1, we now introduce an ADMM algorithm for solving the general penalized quadratic regression problem (1).

3.1. ADMM algorithm

Writing the squared loss function

$$f_0(\mathbf{B}) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^\top \mathbf{B}\mathbf{x}_i)^2,$$

we study the generic problem

$$\min f_0(\mathbf{B}) + f_1(\mathbf{B}) + \cdots + f_N(\mathbf{B}),$$

where $f_k(\cdot), k = 1, \ldots, N$ are penalty functions. Introducing the local variables $\mathbf{B}_i \in \mathbb{R}^{p\times p}$, the problem can be equivalently rewritten as the following *global consensus problem* (**?**, Section 7)

$$\min \sum_{i=0}^{N} f_i(\mathbf{B}_i), \text{ subject to } \mathbf{B}_i - \mathbf{B} = \mathbf{0},\ i = 0, 1, \ldots, N. \tag{11}$$

The augmented Lagrangian of (11) is

$$L(\mathbf{B}_0, \ldots, \mathbf{B}_N, \mathbf{B}, \mathbf{U}_0, \ldots, \mathbf{U}_N) = \sum_{i=0}^{N}\left\{f_i(\mathbf{B}_i) + \frac{\rho}{2}\|\mathbf{B}_i - \mathbf{B} + \mathbf{U}_i\|_2^2\right\}.$$

For a given solution $\mathbf{B}_i^k, i = 0, \ldots, N$ in the kth iteration, the $(k+1)$th iteration of the ADMM algorithm is given as follow:

- Step 1: $\mathbf{B}_i^{k+1} = \arg\min_{\mathbf{B}_i}\left\{f_i(\mathbf{B}_i) + \frac{\rho}{2}\|\mathbf{B}_i - \mathbf{B}^k + \mathbf{U}_i^k\|_2^2\right\}, i = 0, \cdots, N$;
- Step 2: $\mathbf{B}^{k+1} = \frac{1}{N+1}\sum_{i=0}^{N}\left\{\mathbf{B}_i^{k+1} + \mathbf{U}_i^k\right\}$;
- Step 3: $\mathbf{U}_i^{k+1} = \mathbf{U}_i^k + \mathbf{B}_i^{k+1} - \mathbf{B}^{k+1},\ i = 0, \cdots, N$.

If we start with $\sum \mathbf{U}_i^1 = \mathbf{0}$, it can be shown that $\sum \mathbf{U}_i^k = \mathbf{0}$ for every $k > 1$ and so Step 2 will simply be an average operator, i.e.,

$$\mathbf{B}^{k+1} = \frac{1}{N+1}\sum_{i=0}^{N}\mathbf{B}_i^{k+1}.$$

As we can see, the computational complexity of the algorithm is usually dominated by the first step.

In general, for a convex function $f(\cdot)$, the proximal operator (**?**) is defined as:

$$\mathrm{proc}_{f,\rho}(\mathbf{A}) \overset{\mathrm{def}}{=} \arg\min_{\mathbf{B}} f(\mathbf{B}) + \frac{\rho}{2}\|\mathbf{B}-\mathbf{A}\|_2^2. \tag{12}$$

Thus, given $\mathbf{B}^k$ and the $\mathbf{U}_i^k$'s, Step 1 is a proximal operator for the sum of the squared loss function $f_0(\cdot)$ and the penalty functions $f_i(\cdot),\ i=1,\dots,N$. In Proposition 2.1 we have derived an efficient form for the proximal operator of the squared loss $f_0(\cdot)$ at $\mathbf{A}=\mathbf{0}$. For a general $\mathbf{A}$ in (12), the efficient solution can be obtained by setting $\lambda=\rho/2$ and updating $\mathbf{D}$ as $n^{-1}\sum_{i=1}^n y_i\mathbf{x}_i\mathbf{x}_i^\top + \mathbf{A}$ in Proposition 2.1. In next subsection, we provide the proximal operator for each penalty function.

3.2. Proximal operator

For most penalty functions, the proximal projection has an explicit solution, and we summarize these operators in this section. With some abuse of notation, let $\mathbf{B}$ be a parameter matrix with dimension $p\times q$. For the ℓ_1 norm, writing $\mathbf{A}=(A_{ij})_{p\times q}$, we have

$$\arg\min_{\mathbf{B}} \lambda\|\mathbf{B}\|_1 + \frac{1}{2}\|\mathbf{B}-\mathbf{A}\|_2^2 = (\mathrm{sign}(A_{ij})(|A_{ij}|-\lambda)_+)_{p\times q} \overset{\mathrm{def}}{=} \mathrm{soft}(\mathbf{A},\lambda),$$

where $x_+=\max(0,x)$. For the nuclear norm, denoting the singular value decomposition of $\mathbf{A}$ as

$$\mathbf{A} = \sum_{i=1}^{\min(p,q)} \sigma_i\mathbf{u}_i\mathbf{v}_i^\top,$$

we have

$$\arg\min_{\mathbf{B}} \lambda\|\mathbf{B}\|_* + \frac{1}{2}\|\mathbf{B}-\mathbf{A}\|_2^2 = \sum_{i=1}^{\min(p,q)} (\sigma_i-\lambda)_+\mathbf{u}_i\mathbf{v}_i^\top.$$

For other penalties imposed on the columns or the rows of $\mathbf{B}$, we present the solutions in the form of vectors for brevity. Considering the proximal operator,

$$\widehat{\mathbf{b}} = \arg\min_{\mathbf{b}} f(\mathbf{b}) + \frac{1}{2}\|\mathbf{b}-\mathbf{a}\|_2^2,\ \mathbf{a},\mathbf{b}\in\mathbb{R}^q,$$

we have the following solution.

- ℓ_2 norm–Group LASSO (**?**):

$$f(\mathbf{b}) = \lambda\|\mathbf{b}\|_2, \ \widehat{\mathbf{b}} = \left(1 - \frac{\lambda}{\|\mathbf{a}\|_2}\right)_+ \cdot \mathbf{a}.$$

- ℓ_∞ norm penalty (**?**):

$$f(\mathbf{b}) = \lambda\|\mathbf{b}\|_\infty.$$

  When $\lambda \geq \|\mathbf{a}\|_1$, we have $\hat{\mathbf{b}} = \mathbf{0}$. Otherwise, the solution is

$$\widehat{\mathbf{b}} = \mathbf{a} - \mathrm{soft}(\mathbf{a}, \lambda_1),$$

  where $\lambda_1 \geq 0$ satisfies the equation

$$\sum_{i=1}^{q}(|a_i| - \lambda_1)I(|a_i| > \lambda_1) = \lambda.$$

  The details of the derivation can be found in Section 5.4 of **?**.

- Hybrid ℓ_1/ℓ_∞ norm penalty (**?**):

$$f(\mathbf{b}) = \lambda \max\left(|b_1|, \sum_{i=2}^{q}|b_i|\right).$$

  The solution is

$$\widehat{\mathbf{y}} = \left(\mathrm{soft}(a_1, \lambda_1), \mathrm{soft}(\mathbf{a}_{-1}, \lambda - \lambda_1)\right),$$

  where

$$\lambda_1 = \operatorname*{arg\,min}_{t\in[0,\lambda]} \|\mathrm{soft}(a_1, t)\|_2^2 + \|\mathrm{soft}(\mathbf{a}_{-1}, \lambda - t)\|_2^2.$$

  In particular, when $\lambda \geq |a_1| + \|\mathbf{a}_{-1}\|_\infty$, $\widehat{\mathbf{b}} = \mathbf{0}$. Further details can be found in **?**.

With these explicit proximal operators, we can get the unified algorithm as follows.

Algorithm 1 HiQR: High dimensional Quadratic Regression.

```
Initialization:
 1: Input the observations $(\mathbf{x}_i, y_i),\ i = 1, \cdots, n$;
 2: Set the loss function $f_0(\cdot)$ and the penalty functions $f_1(\cdot), \cdots\ f_N(\cdot)$;
 3: Start from $k = 0$, $\mathbf{B}_i^0 = \mathbf{U}_i^0 = \mathbf{0}_{p\times p}$.
Iteration:
 4: Update $\mathbf{B}_i^{k+1} = \text{proc}_{f_i,\rho}(\mathbf{B} - \mathbf{U}_i),\ i = 0, \cdots, N$.
 5: Update $\mathbf{B}^{k+1} = \frac{1}{N+1}\sum_{i=0}^{N}\mathbf{B}_i^{k+1}$.
 6: Update $\mathbf{U}_i^{k+1} = \mathbf{U}_i^k + \mathbf{B}_i^{k+1} - \mathbf{B}^{k+1}\ i = 0, \cdots, N$.
 7: Repeat steps 4-6 until convergence.
Output: Return $\mathbf{B}$.
```

Algorithm 1 is simple and efficient owing to the fact that each step of the iteration has a closed form, and we have greatly utilized the matrix structure of the problem to obtain a closed-form solution for the proximal operator of the squared loss for quadratic regression. The algorithm is fully matrix-based, where we update $p \times p$ matrices in each step without any unnecessary matrix operations such as vectorization and Kronecker product. This can greatly reduce the memory and computational burden when handling high-dimensional data.

4. Simulations

To illustrate the efficiency of the proposed algorithm, we consider a toy example:

$$Y = 2X_1 - 2X_5 + 2X_{10} + 3X_1X_5 - 2.5X_5^2 + 4X_5X_{10} + \epsilon.$$

For all the simulations, we generate $\mathbf{x}_1, \cdots, \mathbf{x}_n$ independently from $N(\mathbf{0}, \mathbf{\Sigma})$, where $\mathbf{\Sigma} = (0.5^{|k-l|})_{p\times p}$, and the error term ϵ from $N(0, 1)$. We fix the sample size $n = 1000$, and vary the data dimension p from small to large. The code is implemented on an Apple M1 chip with 8-core CPUs and 8G RAM, and the R version used is 4.2.3 with vecLib BLAS.

4.1. Ridge regression

In this part, we compare four algorithms for computing the ridge-penalized quadratic regression, namely, the naive inverse (4), the Woodbury trick (5), the SVD method (6), and the proposed HiQR. We fix $\lambda = 10$, and the computation times are recorded in seconds based on 10 replications.

From Table 1, we can observe that our HiQR algorithm greatly outperforms other algorithms in terms of computation efficiency. Additionally,

Table 1: Average computation time (standard deviation) of different algorithms for ridge regression ($\lambda = 10$) over 10 replications. Time is recorded in seconds.

	p=100	p=200	p=400	p=800	p=1200
Naive	9.808 (0.113)	NA	NA	NA	NA
Woodbury	0.129 (0.005)	0.557 (0.043)	2.214 (0.095)	21.551 (2.892)	NA
SVD	0.547 (0.014)	2.685 (0.042)	13.755 (0.167)	72.081 (1.798)	NA
HiQR	0.019 (0.001)	0.021 (0.001)	0.023 (0.002)	0.047 (0.005)	0.051 (0.005)

*NA is produced due to out of memory in R.

the results are roughly consist with their native computational complexity, e.g., $O(p^6)$, $O(n^2p^2 + n^3)$, $O(n^2p^2)$ and $O(np^2 + n^3)$. As we can see, the vectorization methods all fail to handle the $p = 1200$ case due to memory shortage, while our method is still efficient, as we only need to handle the storage of $n \times p$ and $p \times p$ matrices.

4.2. Single penalty function

In this part, we investigate the performance of the proposed HiQR for a single penalty, i.e., $f(\mathbf{B}) = \lambda\|\mathbf{B}\|_1$. As a comparison, we also implement the all-pairs LASSO of vectorized features using two state-of-the-art algorithms, e.g., "glmnet" (**?**) and "ncvreg" (**?**). Table 2 reports the computation times of these three algorithms for a solution path with 50 λs based on 10 replications. From Table 2, we can see that while the proposed HiQR is comparable to the two well-established algorithms, both "glmnet" and "ncvreg" fail to generate solutions when $p = 1200$ due to out-of-memory errors.

Table 2: Average computation time (standard deviation) of three packages for obtaining a solution path for all-paris LASSO over 10 replications. The same set of 50 λs has been used for the three different packages, and time is recorded in seconds.

	p=100	p=200	p=400	p=800	p=1000
glmnet	0.235(0.205)	0.745(0.127)	3.342(0.268)	21.167(1.207)	NA
ncvreg	0.373(0.063)	1.421(0.129)	6.136(0.102)	35.162(2.004)	NA
HiQR	0.324(0.087)	0.987(0.171)	5.475(1.308)	26.215(7.535)	31.319(17.579)

We remark that both "glmnet" and "ncvreg" are accelerated by using strong rules; see **?** and **?** for more details. Strong rules screen out a large number of features to substantially improve computational efficiency. However, as **?** has pointed out, the price is that "the strong rules are not foolproof and can mistakenly discard active predictors, that is, ones that have nonzero coefficients in the solution.” As a comparison, our algorithm can be as efficient as “glmnet” and “ncvreg” without the need for the same type of acceleration.

4.3. Hybrid penalty functions

In this part, we report the performance of HiQR for hybrid penalty functions. Specifically, we conduct simulations for the $\ell_1 + \ell_2$, $\ell_1 + \ell_\infty$, $\ell_1 + \ell_1/\ell_\infty$, and $\ell_1 + \ell_*$ penalties. The two parameters λ_1 and λ_2 are determined by λ and $\alpha \in (0, 1)$, that is,

$$\lambda_1 = \lambda \cdot \alpha \cdot \lambda_{1,max}, \ \lambda_2 = \lambda \cdot (1 - \alpha) \cdot \lambda_{2,max},$$

where $\lambda_{1,max}$ ($\lambda_{2,max}$) is set to be the smallest tuning value corresponding to a zero estimation when λ_2 (λ_1) is set to be 0. We apply HiQR over a 10×50 grid of (α, λ) values, and Table 3 presents the average computation times for the whole procedure. We can see that the proposed algorithm scales very well to high-dimensional quadratic regression.

Table 3: Average computation times (standard deviation) of “HiQR” under hybrid penalties with 500 tuning pairs over 10 replications. Time is recorded in seconds.

	$\ell_1 + \ell_2$	$\ell_1 + \ell_\infty$	$\ell_1 + \ell_1/\ell_\infty$	$\ell_1 + \ell_*$
$p = 50$	8.947(0.735)	5.113(0.495)	7.654(0.519)	14.107(1.038)
$p = 100$	25.963(2.952)	10.616(1.194)	21.891(1.964)	46.136(2.897)
$p = 200$	132.924(6.018)	28.047(2.152)	105.835(6.322)	204.892(11.087)

4.4. Model performance

Lastly, we evaluate different penalties on different models. In particular, we consider

$$\text{Model 1: } Y = 2X_1 - 2X_5 + 2X_{10} + 3X_1X_5 - 2.5X_5^2 + 4X_5X_{10} + \epsilon, \quad (13)$$

$$\text{Model 2: } Y = -2X_5 + 3X_1X_5 - 2.5X_5^2 + 4X_5X_{10} + \epsilon, \quad (14)$$

$$\text{Model 3: } Y = 3X_1X_5 - 2.5X_5^2 + 4X_5X_{10} + \epsilon, \quad (15)$$

where the true parameters of $\mathbf{B}[(1, 2, 6, 11), (1, 2, 6, 11)]$ are

$$\begin{pmatrix} 0 & 1 & -1 & 1 \\ 1 & 0 & 1.5 & 0 \\ -1 & 1.5 & -2.5 & 2 \\ 1 & 0 & 2 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 1.5 & 0 \\ -1 & 1.5 & -2.5 & 2 \\ 0 & 0 & 2 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1.5 & 0 \\ 0 & 1.5 & -2.5 & 2 \\ 0 & 0 & 2 & 0 \end{pmatrix}.$$

respectively. In particular, Model 1 has a strong hierarchical structure, Model 2 has a weak hierarchical structure, and Model 3 is a model with only interactions.

Due to the efficiency of HiQR, we studied a high-dimensional case where $p = 200$ and $n = 500$. It is noted that the model has about 2×10^4 parameters. We implemented the penalized quadratic regression with 50 αs and 50 λs, resulting in a solution path for 2500 grids. To measure each estimation $\widehat{\mathbf{B}}$, we adopted the critical success index (CSI), which can evaluate the support recovery rate and the model size simultaneously. For the true $\mathbf{B}$ and an estimation $\widehat{\mathbf{B}}$, the CSI is defined as follows:

$$\mathrm{CSI}(\mathbf{B}, \widehat{\mathbf{B}}) = \frac{\#\{(i,j) : B_{ij} \neq 0 \text{ and } \hat{B}_{ij} \neq 0\}}{\#\{(i,j) : B_{ij} \neq 0 \text{ or } \hat{B}_{ij} \neq 0\}}.$$

Figure 1 presents the results for different models and different penalties. From these solution paths, we can see that these methods can detect the true signals if the tuning parameters are set suitably. Although tuning parameters selection is beyond the scope of the current work, our results indicate that the proposed "HiQR" algorithm is capable of training a model with 2×10^4 parameters and 2500 tuning parameters efficiently, and is able to generate estimations with satisfactory support recovery (c.f. grids in black in Figure 1).

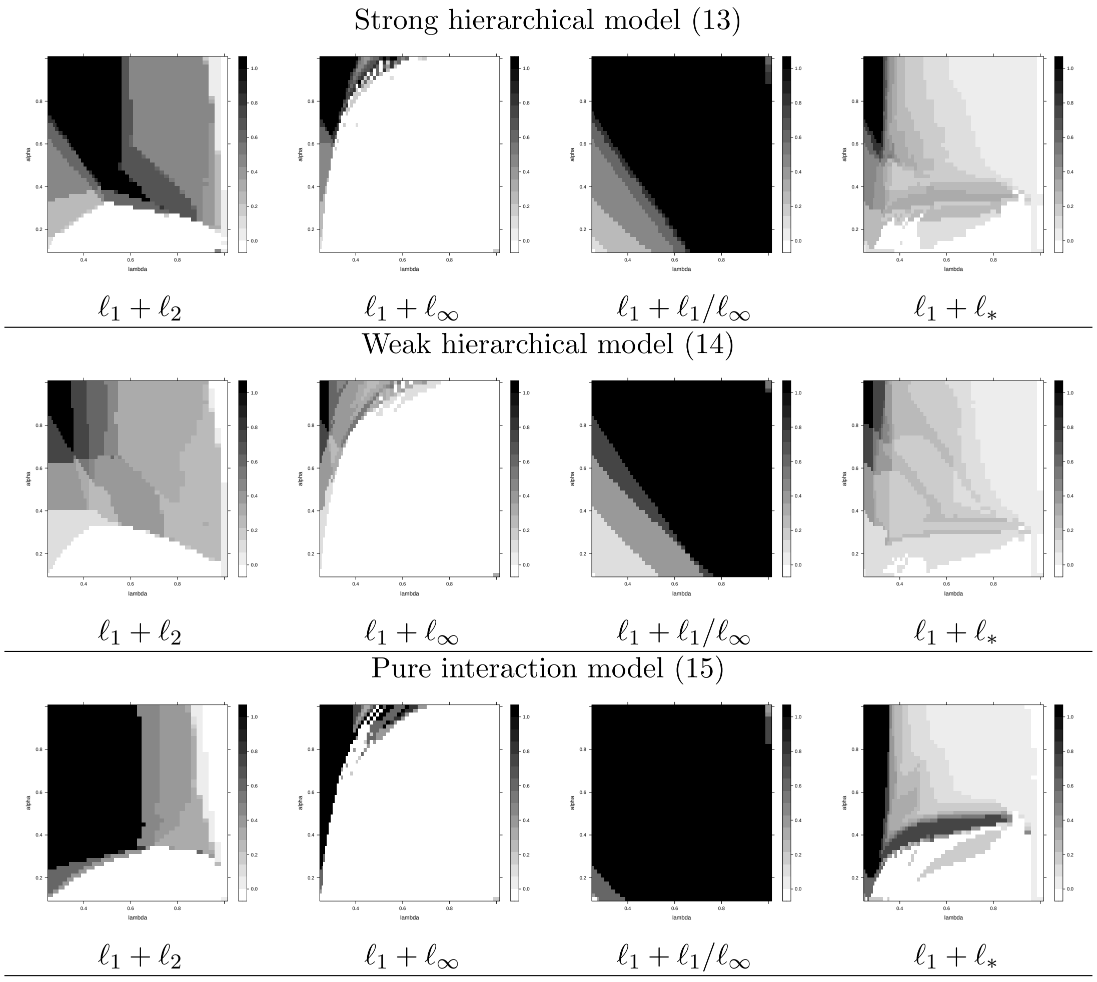

Figure 1: The critical success index for different models and different penalties with 2500 tuning parameters.