Homotopy algorithm for $l_1$-norm minimisation problems

Jisheng Dai$^{1,5}$, Weichao Xu$^2$, Jin Zhang$^3$, Chunqi Chang$^4$

$^1$School of Electrical and Information Engineering, Jiangsu University, Zhenjiang 212013, People’s Republic of China
$^2$Department of Automatic Control, School of Automation, Guangdong University of Technology, Guangzhou 510006, People’s Republic of China
$^3$Department of Electrical and Electronic Engineering, Hefei Electronic Engineering Institute, Hefei 230027, People’s Republic of China
$^4$School of Electronic and Information Engineering, Soochow University, Suzhou 215006, People’s Republic of China
$^5$National Mobile Communications Research Laboratory, Southeast University, Nanjing 210096, People’s Republic of China

E-mail: jsdai@ujs.edu.cn

**Abstract:** This paper proposes a novel approach to handle the singularity problem in the Homotopy algorithm. Following a state-of-the-art ridge-adding-based method, the authors introduce a random ridge term to each element of the measure matrix to avoid the occurrence of singularities. Then, the authors give the sufficient condition of avoiding singularities, and prove that adding random ridge is greatly useful to deal with the singularity problem, even the random ridge tends to zero. Although the main procedures in the method coincide with the state-of-the-art method, all the derivations and theoretical analyses are different because of distinct forms of optimisation problems. Thus, this work is by no means a trivial task. Moreover, the authors note that the most computationally expensive step in the proposed method is the inversion of active matrices, whose time complexity relative to that of other required operations is proportional to the active set’s cardinality. This motivates us to develop an efficient QR update algorithm based on the new ridge-adding-based method, which is incorporated in Givens QR factorisation and Gaussian elimination. It turns out that, with such new QR update algorithm, the previous ratio of time complexities can be reduced to a constant order. As a consequence, the new method can eliminate the bottleneck of matrix inversion in the improved Homotopy algorithm.

**1 Introduction**

Recently, $l_1$-norm minimisation has been proved to be an effective tool for solving the underdetermined system of linear equations, and has attracted tremendous attention in the signal processing and optimisation communities [1, 2]. In a typical underdetermined linear regression problem, we are given a measurement vector $y \in \mathbb{R}^{P \times 1}$ generated from an unknown signal of interest $x_0 \in \mathbb{R}^{N \times 1}$ by a linear transformation $y = Ax_0 + n$, where $n$ is a noise vector and $A \in \mathbb{R}^{P \times N}$ is a known measurement matrix. In order to obtain a vector $x_l$ that approximates the interested signal $x_0$, we compute a minimum $l_1$-norm solution, which is given by the following mathematical formula

\[
(P_1) \min_{x} \|x\|_1 \text{ subject to } \|y - Ax\|_2^2 \leq \epsilon
\]  

(1)

or, equivalently, in a form of the so-called LASSO problem [3]

\[
(P_2) \min_{x} \|y - Ax\|_2^2 \text{ subject to } \|x\|_1 \leq t
\]  

(2)

where $\epsilon$ and $t$ are certain pair constants. Moreover, it is convenient to consider the unconstrained optimisation problem instead

\[
(P_3) \min_{x} \frac{1}{2} \|y - Ax\|_2^2 + \lambda \|x\|_1
\]  

(3)

that is, a form of $l_1$-penalised least-squares. Indeed, $(P_1)$, $(P_2)$ and $(P_3)$ are equivalent under an appropriate correspondence of parameters [1].

The idea of $l_1$-norm minimisation to encourage sparsity is now well established, and its applications have been proposed in the context of feature selection [3], face recognition [2], sparse portfolio optimisation [4], trend filtering [5], array signal processing [6–8] and compressed sensing [9–11]. Among several efficient and robustness algorithms for solving the $l_1$-norm minimisation problem, the Homotopy algorithm [1, 12] is the most interesting and powerful one for $(P_3)$ in a sparse setting. The Homotopy algorithm starts out at $\lambda = \infty$ with the initial solution $x = 0$. As $\lambda$ decreases, the algorithm computes the solution for every value of $\lambda$ in a piecewise linear manner, and successively builds a sparse solution by adding or removing elements from its active set. As long as sufficient sparsity is present, it will reach the solution in a few steps.
The Homotopy algorithm provides such an important active-set method for solving the quadratic programming (QP) that it is extended to other related problems, such as support vector machine (SVM) [13], support vector regression (SVR) [14], compressed sensing [12] and multiple-input and multiple-output precoder optimisation [15]. Nevertheless, as previously reported in the literature, these active-set algorithms (including the Homotopy algorithm), which rely on the active set to identify the search direction in each iteration, suffer from a lack of provision for singularities. It will result in algorithmic instability when the measurement matrix contains duplicate columns, nearly duplicate columns, or columns that are linearly dependent in kernel space. Such measurement matrices are quite common among many real-world data. Note that the singularity problem in SVMpath has been pointed out and solved in [16, 17]. However, to the best of our knowledge, the singularity problem in the Homotopy algorithm has still not been addressed.

In this paper, we will propose a novel approach to handle the singularity problem in the Homotopy algorithm. Inspired by the ridge-adding-based method proposed in [17], we similarly introduce a random ridge term to each element of the measure matrix to avoid the occurrence of singularities in the Homotopy algorithm. Such ridge-adding-based method is to ensure that only one index is added into or removed from the active set, rather than directly modifying a singular matrix to guarantee the existence of the inverse matrix. Therefore the new proposed approach, like the one in [17], is distinctly different from the conventional ridge-adding methods. Although the main procedures in our method coincide with the ones in [17], all the derivations and theoretical analyses are quite different because of the distinct corresponding QP forms. Thus, this work is by no means a trivial task. It can be considered as a useful extension of the ridge-adding-based method proposed in [17].

The other contribution of this paper is to propose a much more efficient QR update Homotopy algorithm based on the new ridge-adding-based method. We note that, in the proposed method, the most computationally expensive step is the inversion of active matrices, whose time complexity relative to that of other required operations is proportional to the active set’s cardinality. Hence, a considerable reduction in computational complexity can be achieved if we are able to handle the matrix inversion in a much more efficient way. Together with the procedure of random ridge adding (which guarantees that the active matrix is changed only slightly), we are able to employ an efficient QR update algorithm, which is incorporated in Givens QR factorisation and Gaussian elimination. It turns out that, with such new QR update algorithm, the previous ratio of time complexities can be reduced to a constant order. As a consequence, the new efficient method can eliminate the bottleneck of matrix inversion in the improved Homotopy algorithm.

The rest part of this paper is organised as follows. Section 2 briefly reviews the conventional Homotopy algorithm and discusses the singularity problem. Section 3 presents our improved method for handling singularities. Section 4 proposes an efficient QR update Homotopy algorithm. Numerical experiments and conclusions are presented in Sections 5 and 6, respectively.

2 Review of the Homotopy algorithm

The general principle underlying the Homotopy algorithm is to fit the entire path solution to \((P_l)\) for every value of the regularisation parameter \(\lambda\) by utilising the piecewise linear structure [12]. In constructing a decreasing sequence of \(\lambda\), it only has to identify those ‘breakpoints’ corresponding to the occurrence of one event: adding/dropping elements into/from its active set. The solution path finding is followed by maintaining the optimality conditions of \((P_l)\) for each point. Denote the objective function of \((P_l)\) as \(f_{\lambda}(x)\), and define active set \(\mathcal{E}\) and its complementary set \(\mathcal{R}\) as

\[
\begin{align*}
\{ x_i \neq 0, \text{ if } i \in \mathcal{E} \\
\{ x_i = 0, \text{ if } i \in \mathcal{R} 
\end{align*}
\]

Then, from the Karush-Kuhn-Tucker (KKT) optimality conditions, we can derive the following relationship

\[
\frac{\partial f_{\lambda}(x)}{\partial x} = -A^T(y-Ax) + \lambda u = 0
\]

where \(u\) is the subgradient of \(\|x\|_1\), whose elements are given by

\[
\begin{align*}
\{ u_i = \text{sgn}(x_i), \text{ if } i \in \mathcal{E} \\
\{ u_i \in [-1, 1], \text{ if } i \in \mathcal{R} 
\end{align*}
\]

The (4) can be rewritten equivalently as the following two conditions

\[
|A_{\mathcal{E}}^T(y-Ax)| \leq \lambda \cdot 1
\]

\[
A_{\mathcal{E}}^T(y-Ax) = \lambda \cdot \text{sgn}(x_{\mathcal{E}})
\]

where \(A_{\mathcal{E}}\) denotes a matrix with columns corresponding to indexes in \(\mathcal{E}\). Likewise for \(A_{\mathcal{R}}\). The KKT optimality conditions guarantee that if \(x\) satisfies (5) and (6), it must be an optimal solution to \((P_l)\).

Taking a derivative on both sides of (6) with \(\lambda\), we have

\[
A_{\mathcal{E}}^T A_{\mathcal{E}} x'_{\mathcal{E}} = -\text{sgn}(x_{\mathcal{E}})
\]

where \(x' \triangleq (dx/d\lambda)\). With a sufficiently small change of \(\lambda\), each index \(i \in \mathcal{R}\) remains in its original set. Hence, \(x'_{i} = 0, i \in \mathcal{R}\). Therefore (7) leads to

\[
A_{\mathcal{E}}^T A_{\mathcal{E}} x'_{\mathcal{E}} = -\text{sgn}(x_{\mathcal{E}})
\]

The Homotopy algorithm assumes that the matrix \(A_{\mathcal{E}}^T A_{\mathcal{E}}\) is of full rank, and computes the search direction

\[
x'_{\mathcal{E}} = -(A_{\mathcal{E}}^T A_{\mathcal{E}})^{-1} \text{sgn}(x_{\mathcal{E}})
\]

\[
x'_{\mathcal{R}} = 0
\]

Then, we are able to update \(\lambda\) and \(x\) as follows

\[
\lambda^{(l+1)} = \lambda^{(l)} + \Delta \lambda
\]

\[
x^{(l+1)} = x^{(l)} + \Delta \lambda \cdot x'
\]

where the superscript \(l\) attached to a variable (or a set) is used to index its value after the \(l\)th event (described below) has occurred. We just have to monitor the occurrence of any of these possible events:
• Adding event: An index \( i \in \mathcal{E}^{(0)} \) moves into \( \mathcal{R}^{(0)} \), that is, one point in \( i \in \mathcal{R}^{(0)} \) attains \( A_i^T(y - Ax^{(0)} + \Delta x) = (A^0 + \Delta A) \), where \( A_i \) denotes the \( i \)-th column of \( A \). The corresponding value of \( \Delta \lambda \) for each \( i \) is determined by

\[
\Delta \lambda = \frac{A_i^T(y - Ax^{(0)})}{A_i^TA_i} \pm 1 \tag{13}
\]

• Dropping event: An index \( i \in \mathcal{R}^{(0)} \) moves into \( \mathcal{E}^{(0)} \), that is, one point in \( i \in \mathcal{E}^{(0)} \) hits 0. The corresponding value of \( \Delta \lambda \) for each \( i \) can be easily established by

\[
\Delta \lambda = -x_i^{(0)}/x_i' \tag{14}
\]

For the sake of constructing a monotonic descending sequence of \( \lambda^{(0)} \), candidates of \( \Delta \lambda \) must be always negative. The smallest value of \( |\Delta \lambda| \) among all the negative candidates of \( \Delta \lambda \) is selected to determine the breakpoint, and then update \( \lambda^{(l + 1)} \in \mathcal{R} \) and \( \mathcal{E} \) accordingly. The algorithm is terminated when \( \|y - Ax\|^2 = \epsilon \) or \( \|x\|_1 = t \), and the solution to \((P_1)\) or \((P_2)\) is reached. Note that a proper initialisation is required to trigger the path finding procedure. The Homotopy algorithm starts out at a large \( \lambda^{(0)} \) with the initial solution \( x^{(0)} = 0 \), that is, \( \mathcal{R}^{(0)} = \{1, 2, \ldots, n\} \) and \( \mathcal{E}^{(0)} = [] \).

As the Homotopy algorithm is an active-set method, it might suffer from a lack of provision for potential singularities when solving for the search direction at each iteration. This is because of rank deficiency of the matrix \( A_i^TA_i \). To obtain the search direction \( x'_e \) in (9), the Homotopy algorithm has to assume that \( A_i^TA_i \) is of full rank. Unfortunately, this assumption is not always valid, especially in the measurement matrix having duplicate columns, or columns that are linearly dependent. Such measurement matrices are quite common among many real-world data, and become more prominent in large-scale applications [16, 17]. As shown by examples provided in [16, 17], active-set methods will halt because of the failure in calculating the inverse of active matrices.

### 3 Handle the singularity problem

In this section, we will present our method to handle the singularity problem in the Homotopy algorithm. Inspired by the ridge-adding-based method proposed in [17], we first establish a sufficient condition of avoiding singularities based on a rigorous mathematical argument. Then, we prove that the active set never chooses a column that makes the new active matrix low-ranked. Although the main procedures in our method coincide with the ones in [17], all the derivations and theoretical analyses are quite different because of the different corresponding QP forms. Thus, this work is by no means a trivial task.

Firstly, we claim the sufficient condition of avoiding singularities as

**Lemma 1:** If there is only one index being added into or removed from the active set with the change of \( \lambda \) at one time, the matrices \( A_{i_{l=0}}^TA_{i_{l=1}}, l = 1, 2, \ldots \) retain full rank.

**Proof:** See Appendix 1.

From the review of the Homotopy algorithm, we know that if the smallest value of \( \Delta \lambda \), determined by (13) and (14), is non-unique, two or more than two indexes are added into or removed from the active set simultaneously. Thus, in order to guarantee the sufficient condition given in Lemma 1, we may add a random ridge term into each element of the matrix \( A \), for example

\[
\hat{A} = A + \Gamma
\]

The element of \( \Gamma \), denoted as \( \Gamma_{i_{l=0}} \), is exactly the random ridge, which might be generated from a zero-mean normal distribution with a small common variance. With the modified matrix \( \hat{A} \), \((P_3)\) turns to

\[
\min x \frac{1}{2} \|y - \hat{A}x\|_2^2 + \lambda \|x\|_1 \tag{16}
\]

and the KKT conditions become

\[
[\hat{A}^T \Gamma \ (y - \hat{A}x)] \leq \lambda I \tag{17}
\]

\[
\hat{A}^T(y - \hat{A}x) = \lambda \cdot \text{sgn}(x_c) \tag{18}
\]

Owing to the randomness of \( \Gamma_{i_{l=0}} \), the index corresponding to the smallest value of \( |\Delta \lambda| \) will be unique almost surely (with probability one). Hence, only one index corresponding to the smallest value of \( |\Delta \lambda| \) is chosen, and it will be either removed from the active set (if the smallest \( |\Delta \lambda| \) comes from (14)) or added into the active set (if the smallest \( |\Delta \lambda| \) comes from (13)).

It is worth noting that ridge adding is a common technique that is employed to dealing with singular matrix inversions and optimisations. The most obvious way to remove singularities is to manipulate the singular active matrices directly. However, its added ridge cannot be chosen as small as possible, because the new constructed matrix is nearly singular, and the method will be out of work if the ridge tends to zero. On the other hand, if the added ridge is large, the effect of the added ridge will gradually accumulate along the solution path as more singular events occur. Unlike the above naive method, our method adopts a distinctly different strategy which ensures that only one index is added into or removed from the active set through ridge addition. In Lemma 2, we will show that our method does work well when the random ridge tends to zero. Hence, the added ridge in our situation can be chosen as small as possible. In this case, the new generated solution path might be almost identical to the original one. Specifically, if we choose sufficiently small random ridges, for example, \( O(10^{-x}) \), problem (16) can be rewritten as

\[
\min x \frac{1}{2} \|y - Ax\|_2^2 + \lambda \|x\|_1 + O(10^{-x}) \tag{19}
\]

Clearly, the distance between the optimal solutions to \((P_3)\) and (19) is of order \( O(10^{-x}) \) for every value of the regularisation parameter \( \lambda \).

**Lemma 2:** If \( \hat{A}_{i_{l=0}} \) is of full column rank, \( \hat{A}_{i_{l=1}} \) will be of full column rank for any sufficiently small random ridge almost surely (with probability one).

**Proof:** See Appendix 2.
Summarily, we have the following results:

(1) The sufficient condition of avoiding singularities is only one index being added into or removed from the active set (Lemma 1);
(2) Adding random ridge can guarantee the sufficient condition holding;
(3) The traditional shortcoming of adding random ridge (i.e., the new matrix with random added ridge is nearly singular if the ridge tends to zero) is excluded from our situation almost surely (Lemma 2).

Hence, adding random ridge is greatly useful to deal with the singular problem in the Homotopy algorithm. The singularity problem with active set will not occur almost surely, even the random ridge tends to zero. In order to maintain a reasonable accuracy of $l_i$-minimisation, we may choose very small random ridge in practice, for example, $O(10^{-8})$.

4 Computational speedup

The computational complex of the inversion in (8) is of order $O(|E|^2)$, which is the most computationally expensive step in the Homotopy algorithm when $|E|$ is large. For some $B \in \mathbb{R}^{m \times m}$, $x \in \mathbb{R}^m$, $y \in \mathbb{R}^m$ such that $Bx = y$, we note that an alternative to solving the equality $Bx = y$ is to perform the QR factorisation of $B$ (e.g., $B = QR$) so that

$$Rx = Q^Ty$$

Then, $x$ can be efficiently obtained by Gaussian elimination. Moreover, it is well known that if $B$ is changed slightly, insertion/deletion update formula can be used to return $Q$ and $R$ without recomputing the QR factorisation from scratch [16]. Since we have ensured that only one index is added into or removed from the active set by adding random ridge in Section 3, $A_{t+1}$ is changed only by an addition/deletion of a row/column. This motivates us to find an efficient way to handle the inversion so as to further reduce the computational cost of the Homotopy algorithm.

For easy presentation, we first introduce the Givens matrix. For any $t \in \mathbb{R}^m$, a Givens matrix, $G(t_i, t_j) \in \mathbb{R}^{m \times m}$, is of the form

$$G(t_i, t_j) = \begin{bmatrix}
I_{l-1} & c & s \\
-s & c & 0 \\
0 & 0 & I_{m-l}
\end{bmatrix}$$

where $c = \left(t_i \sqrt{t_i^2 + t_j^2}\right)$ and $s = \left(-t_j \sqrt{t_i^2 + t_j^2}\right)$, and is therefore orthogonal. Then for $G(i, j)^T = \mathcal{E}$

$$z_k = \begin{cases}
ct_i - st_j & k = i \\
0 & k = j \\
t_k & k \neq i, j
\end{cases}$$

With the assistance of the Givens transformation, we now proceed to establish QR update formulas for the Homotopy algorithm. To shorten notation, we abuse notation by referring to $\hat{A}$ simply as $A$. Let the QR factorisation of $A_{t+1}^T A_{t+1}$ be $Q^{(t)} R^{(t)}$, which can be implemented by means of the Givens transformation. When the next active matrix

$A_{t+1}^T A_{t+1}$ is changed only slightly, we wish to find an efficient way to update the QR factorisation rather than recalculating the QR factorisation from scratch. Base on whether the index is added into or removed from the active set $E_t$, the following two formulas can be obtained.

(1) Insertion Update Formula: Without loss of generality, let $E_t = \{1, 2, \ldots, m\}$ and $(m + 1)$ be the new index entering $E_t$. We can write

$$A_{t+1}^T A_{t+1} = \begin{bmatrix}
A_{1:m}^T A_{1:m} & A_{1:m}^T a_{m+1} \\
a_{m+1}^T A_{1:m} & a_{m+1}^T a_{m+1}
\end{bmatrix}$$

$$= \begin{bmatrix}
Q^{(t)} & (Q^{(t)})^T A_{1:m} a_{m+1} \\
0 & (Q^{(t)})^T a_{m+1} a_{m+1}
\end{bmatrix}$$

$$= \begin{bmatrix}
Q^{(t)} & (Q^{(t)})^T A_{1:m} a_{m+1} \\
0 & (Q^{(t)})^T a_{m+1} a_{m+1}
\end{bmatrix}$$

$$H^{(t+1)}$$

where $P$ is a permutation matrix that permutes the last row to be the first one. Since $R^{(t)}$ is an upper triangular matrix, $H^{(t)}$ is upper Hessenberg. Therefore we can reduce $H^{(t)}$ to an upper triangular matrix with $m$ Givens matrices, $G(\hat{H}_i, H_i^T)$, $i = 1, 2, \ldots, m$, so that

$$A_{t+1}^T A_{t+1} = \begin{bmatrix}
Q^{(t)} & \cdots & \cdots & Q^{(t)} \\
0 & \cdots & \cdots & 0
\end{bmatrix}$$

$$= \begin{bmatrix}
G(H_i, H_i^T) & \cdots & \cdots & G(H_i, H_i^T) \\
0 & \cdots & \cdots & 0
\end{bmatrix}$$

$$R^{(t+1)}$$

Hence, we obtain the insertion update formula for $Q^{(t+1)}$ and $R^{(t+1)}$.

(2) Deletion update formula: Without loss of generality, let $E_t = \{1, 2, \ldots, m\}$ and the $i$th ($1 \leq i \leq m$) index leave $E_t$. With the definition

$$F_i = \begin{bmatrix}
I_{l-1} & 0_{(i-1) \times 1} \\
0_{(m-l) \times i} & I_{m-l}
\end{bmatrix}$$

we can write

$$A_{t+1}^T A_{t+1} = F_i A_{1:m}^T A_{1:m} F_i^T = F_i Q^{(t)} R^{(t)} F_i^T$$

(27)
where the last quality follows from the fact that the norm of the most computationally costly steps is (28) and (32). The most computationally costly step is (26); whereas in the deletion update formula, $P_{G_i}^T A_{G_i(j+1)}^T = F_i Q^j \left[ I_{l-1} \ G_1 \ldots I_{l-1} \ G_{m-j} \right]$

\[
\tilde{q} \begin{bmatrix} H_1 \\ H_2 \\ 0 \end{bmatrix} G_{m-j}^{-1} \tilde{G}_i H_3 \\
\tilde{q}
\]

\[
= F_i P \tilde{Q} \tilde{R}
\]

(29)

where $P_i \in \mathbb{R}^{m \times m}$ is a permutation matrix that satisfies $F_i P_i^T = \begin{bmatrix} 0_{(m-1)\times 1} \\ I_{m-1} \end{bmatrix} (= F_i)$. If $q^j \in \mathbb{R}^m$ is the first row of $P_i \tilde{Q}$, we can zero $q(2:m)$ with $m-1$ Givens matrices, $G(q_i, q_{i+1})$, $i = 1,2,\ldots,m-1$, so that

\[
P_i \tilde{Q} G(q_{m-1}, q_m) \cdots G(q_1, q_2) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0_{(m-1)\times 1}
\end{bmatrix} \tilde{Q}
\]

where the last quality follows from the fact that the norm of $[1 \ \tilde{q}^T]$ equals one (because the matrix in (30) is orthogonal). And we also have

\[
G(q_1, q_2)^T \cdots G(q_{m-1}, q_m)^T \tilde{R} \triangleq \begin{bmatrix} v^T \\ \tilde{R} \end{bmatrix}
\]

(31)

which is upper Hessenberg, so $\tilde{R}$ is upper triangular. Finally, we obtain the QR factorization of $A_{G_i(j+1)}^T A_{G_i(j+1)}^T$ as

\[
\begin{align}
A_{G_i(j+1)}^T A_{G_i(j+1)}^T &= F_i P_i G(q_{m-1}, q_m) \cdots G(q_1, q_2) \\
&= F_i \begin{bmatrix} 1 & 0_{(m-1)\times 1}
\end{bmatrix} \tilde{Q} \tilde{R}
\end{align}
\]

(32)

(33)

which gives the deletion update formula for $Q^{(j+1)}$ and $R^{(j+1)}$, that is, $Q^{(j+1)} = \tilde{Q}$ and $R^{(j+1)} = \tilde{R}$.

In the insertion update formula, the most computationally costly step is (26); whereas in the deletion update formula, the most computationally costly steps are (28) and (32). Since every Givens transformation requires two multiplications and two additions only, the computational complexity of each update formula is $O(|E|)$. Moreover, the Gaussian elimination for $R P_j = Q^{j+1}$ is also of computational complexity $O(|E|)$. Thus, our method is able to reduce the computational cost up to one order (from $O(|E|^2)$ to $O(|E|)$). This reduction is consistent with the QR update SVMpath algorithm in [16].

Another alternative to reducing the computational cost is mentioned in [2], which uses an update rule to compute inverses directly [MATLAB code is available at http://www.eecs.berkeley.edu/˜eyang/software/11benchmark/]. However, it also suffers from a lack of provision for singularities. Moreover, some iterations might need multiple one-rank updates, if two or more than two indexes are added into or removed from the active set simultaneously. In this case, more computational cost is required. Adding random ridge will help to deal with singularities, and obtain the same computational complexity as our method.

5 Simulation results

In this section, we will run some simulations to illustrate the correctness and performance of our proposed method.

To verify the validity of the sufficient condition of avoiding singularities and illustrate that the ridge-adding method works well under singular cases, two kinds of singularity are tested: (1) duplicate singularity; (2) linearly dependent singularity. Simulation 1 considers a measurement matrix $A \in \mathbb{R}^{5 \times 9}$ with duplicate columns, which is in the form of (see equation at the bottom of the page) $y = [−0.7, 0.1, −1.1, −1.9, 0.5]^T$. Table 1 shows the whole path finding steps with the candidate $\lambda$ which are obtained from the determination of breakpoints in the case of duplicate singularity. As we see in Table 1a) when $\lambda = 0.9737$, duplicate column 2 and 6 enter the set $E$ synchronously. Then, the matrix $A_{1,2,6}$ becomes singular, and the path finding procedure is disturbed. In order to handle the duplicate singularity, we modify each element of $A$ by adding a random ridge term $\mathcal{N}(0, 10^{-5})$. Table 1b) illustrates that the tiny modification ensures only one index moving from one set into another with the change of $\lambda$, and singular case is avoided, which is just consistent with our conclusion in Lemmas 1 and 2. Moreover, from those simulation results, it is great to see that our method always achieves the optimal solution for any value of $\lambda$, since the path finding objective cost of $(P_2)$ is the same as that computed directly by using CVX, a package for specifying and solving convex programs [18, 19].

Simulation 2 considers a measurement matrix $A \in \mathbb{R}^{3 \times 6}$ with linearly dependent columns, which is in the form of

\[
\begin{bmatrix}
−0.1, & 0.2, & −0.5, & −0.5, & 0.2, & 0.8 \\
0.0, & 0.4, & 0.0, & −0.2, & −0.3, & 0.0 \\
0.5, & 0.4, & −1.0, & −0.6, & −0.6, & 0.1
\end{bmatrix}
\]

\[
y = [1.0, −1.5, 1.0]^T
\]

Table 2 shows the whole path finding steps with the candidate $\lambda$ which are obtained from the determination of breakpoints in the case of linearly dependent singularity. The singular problem with path finding occurs at $\lambda = 0.5$, which corresponds to indexes 1 and 6 entering into the active set $E$ synchronously. In this case, the set $E$ contains 1, 3, and 6, and $A_{1,3,6}$ are linearly dependent. Hence, the path finding procedure is disturbed because of the singularity of $A_{1,3,6}$. From Table 2b), as expected, singular case is avoided by adding a random ridge.
term $N(0, 10^{-6})$, and the two methods, the modified method and the CVX method, achieve the same objective cost.

To testify the performance improvement under practical data sets, we use seven real data sets, obtained from the University of California at Irvine (UCI) repository [20], in the rest simulations. The Homotopy algorithm and our proposed method are used to testify the performance improvement under practical data sets, obtained from the determination of ‘dropping event’ or ‘adding event’ in the case of linearly dependent singularity.

Table 3 show the results achieved by our method and Homotopy for linear and RBF kernels, respectively. The quantities $|E|_{\text{max}}$, $fl_{\text{max}}$, flops$_{\text{INV}}$, flops$_{\text{UD}}$ and flops$_{\text{rest}}$ refer to the maximal cardinality of the set $E$, the total number of events, the total number of flops required by the inversion, the total number of flops required by the proposed QR update and Gaussian elimination operations, and the total number of flops required by other operations, respectively.

As shown in Table 3, $|E|_{\text{max}}$ is generally small for linear kernel. Compared to flops$_{\text{rest}}$, the cost of calculating inversions (flops$_{\text{INV}}$) is negligible. Hence, our method has no

### Table 1

Whole path finding steps with the candidate $\lambda$s which are obtained from the determination of ‘dropping event’ or ‘adding event’ in the case of duplicate singularity

<table>
<thead>
<tr>
<th>$l$th</th>
<th>$\lambda(l)$</th>
<th>Activated index</th>
<th>Event</th>
<th>Singular</th>
<th>CVX Cost</th>
<th>Homotopy cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3.4000</td>
<td>4</td>
<td>$R \rightarrow E$</td>
<td>no</td>
<td>2.7850</td>
<td>2.7850</td>
</tr>
<tr>
<td>2</td>
<td>3.0124</td>
<td>9</td>
<td>$R \rightarrow E$</td>
<td>no</td>
<td>2.7790</td>
<td>2.7790</td>
</tr>
<tr>
<td>3</td>
<td>1.9746</td>
<td>3</td>
<td>$R \rightarrow E$</td>
<td>no</td>
<td>2.6428</td>
<td>2.6428</td>
</tr>
<tr>
<td>4</td>
<td>1.8999</td>
<td>5</td>
<td>$R \rightarrow E$</td>
<td>no</td>
<td>2.3201</td>
<td>2.3201</td>
</tr>
<tr>
<td>5</td>
<td>0.9737</td>
<td>2, 6</td>
<td>$R \rightarrow E$</td>
<td>yes</td>
<td>2.1677</td>
<td>—</td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3.4000</td>
<td>4</td>
<td>$R \rightarrow E$</td>
<td>no</td>
<td>2.7850</td>
<td>2.7850</td>
</tr>
<tr>
<td>2</td>
<td>3.0124</td>
<td>9</td>
<td>$R \rightarrow E$</td>
<td>no</td>
<td>2.7790</td>
<td>2.7790</td>
</tr>
<tr>
<td>3</td>
<td>1.9746</td>
<td>3</td>
<td>$R \rightarrow E$</td>
<td>no</td>
<td>2.6428</td>
<td>2.6428</td>
</tr>
<tr>
<td>4</td>
<td>1.8999</td>
<td>5</td>
<td>$R \rightarrow E$</td>
<td>no</td>
<td>2.3201</td>
<td>2.3201</td>
</tr>
<tr>
<td>5</td>
<td>0.9737</td>
<td>6</td>
<td>$R \rightarrow E$</td>
<td>no</td>
<td>2.1677</td>
<td>2.1677</td>
</tr>
<tr>
<td>6</td>
<td>0.8958</td>
<td>3</td>
<td>$E \rightarrow R$</td>
<td>no</td>
<td>2.0952</td>
<td>2.0952</td>
</tr>
<tr>
<td>7</td>
<td>0.7078</td>
<td>1</td>
<td>$R \rightarrow E$</td>
<td>no</td>
<td>1.8675</td>
<td>1.8675</td>
</tr>
<tr>
<td>8</td>
<td>0.1238</td>
<td>4</td>
<td>$E \rightarrow R$</td>
<td>no</td>
<td>0.4556</td>
<td>0.4556</td>
</tr>
<tr>
<td>9</td>
<td>0.0453</td>
<td>3</td>
<td>$R \rightarrow E$</td>
<td>no</td>
<td>0.1737</td>
<td>0.1737</td>
</tr>
<tr>
<td>10</td>
<td>0.0000</td>
<td>4</td>
<td>$R \rightarrow E$</td>
<td>—</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

### Table 2

Whole path finding steps with the candidate $\lambda$s which are obtained from the determination of ‘dropping event’ or ‘adding event’ in the case of linearly dependent singularity

<table>
<thead>
<tr>
<th>$l$th</th>
<th>$\lambda(l)$</th>
<th>Activated index</th>
<th>Event</th>
<th>Singular</th>
<th>CVX Cost</th>
<th>Homotopy cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1.5000</td>
<td>3</td>
<td>$R \rightarrow E$</td>
<td>no</td>
<td>2.1250</td>
<td>2.1250</td>
</tr>
<tr>
<td>2</td>
<td>0.5000</td>
<td>1, 6</td>
<td>$R \rightarrow E$</td>
<td>yes</td>
<td>1.6279</td>
<td>—</td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1.5000</td>
<td>3</td>
<td>$R \rightarrow E$</td>
<td>no</td>
<td>2.1250</td>
<td>2.1250</td>
</tr>
<tr>
<td>2</td>
<td>0.5000</td>
<td>3</td>
<td>$R \rightarrow E$</td>
<td>no</td>
<td>1.7250</td>
<td>1.7250</td>
</tr>
<tr>
<td>3</td>
<td>0.4500</td>
<td>5</td>
<td>$R \rightarrow E$</td>
<td>no</td>
<td>1.6830</td>
<td>1.6830</td>
</tr>
<tr>
<td>4</td>
<td>0.4390</td>
<td>6</td>
<td>$R \rightarrow E$</td>
<td>no</td>
<td>1.6726</td>
<td>1.6726</td>
</tr>
<tr>
<td>5</td>
<td>0.4034</td>
<td>2</td>
<td>$R \rightarrow E$</td>
<td>no</td>
<td>1.6314</td>
<td>1.6314</td>
</tr>
<tr>
<td>6</td>
<td>0.0000</td>
<td>4</td>
<td>$R \rightarrow E$</td>
<td>—</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
</tbody>
</table>
advantage in this case. When RBF kernel is used, the data feature pattern is mapped into a very high dimensional space. Therefore $|\mathcal{E}|_{\text{max}}$ is generally large and calculating the inversion of active matrices is computationally expensive. As shown in Table 4, our method reduces significantly the amount of computational cost. For example, when ‘Sonar’ is used, the flops ratio is reduced from 54.5862 to 4.2814.

We conclude that the new method can reduce the overhead of the inversion from being the bottle neck of the Homotopy algorithm to being negligible.

### 6 Conclusions

This paper proposed a novel method of adding random ridges to fill the gap in handling the singularity problem of the Homotopy algorithm. We gave the sufficient condition of avoiding singularities that is only one index being added into or removed from the active set (Lemma 1), and then illustrated that random ridge adding is greatly useful to guarantee the sufficient condition holding. It showed that the singularity problem with the active set would not occur almost surely, even the random ridge tends to zero (Lemma 2). Theoretical analyses and experimental results demonstrated that the new approach worked well in occurrence of singular active matrix. Moreover, as only one index is added into or removed from the active set by adding random ridge, the active matrix $A_{(\ell)}$ is changed only by an addition/deletion of a row/column. Hence, we employed an efficient QR update algorithm, which is incorporated in Givens QR factorisation and Gaussian elimination, to significantly reduce the computational cost of the Homotopy algorithm. Simulation results also verified the efficiency of the proposed method.

### 7 Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grants 61102054 and 61271380, and the Open Research Fund of National Mobile Communications Research Laboratory, Southeast University under Grant 2013D08, China’s Strategic Basic Research Program 973 under Grant 2012CB720702, the Jiangsu Planned Projects for Postdoctoral Research Funds under Grant 1302004A, the Project funded by China Postdoctoral Science Foundation under Grant 2014M560403, and the Guangdong Natural Science Foundation under Grants S20121001009870 and S1414050001981.

### 8 References

2. Yang, A., Ganesh, A., Sastry, S., Ma, Y.: ‘Fast $l_1$-minimization algorithms and an application in robust face recognition: A review’. Technical Report UCB/EECS-2010-13, EECS Department, University of California, Berkeley, February 2010

### 9 Appendix

#### 9.1 Appendix 1: Proof of Lemma 1

**Proof:** Note that proving full rank of the matrix $A_{(\ell)}^T A_{(\ell)}$ is equivalent to showing that the matrix $A_{(\ell)}$ is full column rank. Hence, we focus on the column rank of the matrix $A_{(\ell)}$ in the following part. This proof is verified by an
induction argument. \( A_{\hat{g}(1)} \) is sure to be full column rank, since \( E^{(1)} \) only contains one index. Updating \( \lambda^{(1)} \) with a sufficiently small step \( \delta \) (e.g. \( \lambda^{(1)} = \lambda^{(1)} + \delta \)), the corresponding active set \( E^{(1)} = E^{(1)}_1 \), and \( \hat{x}_i^{(1)} \neq 0, \forall i \in E^{(1)}_1 \). We start by assuming that \( A_{\hat{g}_0} \) is of full column rank with \( \hat{x}_i^{(1)} \neq 0, \forall i \in E^{(1)}_1 \), and then we are going to show that \( A_{\hat{g}(1)} \) retains full rank with \( \hat{x}_i^{(1)} \neq 0, \forall i \in E^{(1)}_1 \).

Updating the value of \( \lambda^{(1)} \) until one event happens. The following results are derived:

- If ‘drop event’ happens, there is an index leaving \( E^{(0)} \). As \( A_{\hat{g}(0)} \) is column full rank, the matrix \( A_{\hat{g}(1)} \) that consists of the columns of the original matrix is also column full rank as long as \( E^{(1)} \) is non-empty. If the empty case occurs, we can resort to the initial configuration method to calculate \( E^{(m+1)} \), and then obtain the corresponding \( A_{\hat{g}(m+1)} \), with a full rank. Thus, the empty case does not invalidate the proof.
- If ‘add event’ happens, there is an index entering \( E^{(0)} \). It is hard to determine the column rank of \( A_{\hat{g}(1)} \) directly. Alternately, the result can be achieved by contradiction. Without loss of generality, let \( E^{(0)} = \{1, 2, \ldots, m\} \) and \( (m + 1) \) be the new index entering into \( E^{(1)} \). Assume that there exists a non-zero vector \( v \) such that \( A_{\hat{g}(1)} v = \hat{A}_{m+1} \). To gain some insight, we now discuss possible values of \( \sum_{i=1}^{m} \text{sgn}(x_i) \cdot v_i \), respectively.

1. We first consider that \( \sum_{i=1}^{m} \text{sgn}(x_i^{(1)}), v_i \neq \text{sgn}(x_i^{(1)}) \). From the KKT conditions at \( \lambda = \lambda^{(1)} \), we have \( v^T A_{\hat{g}(1)} (y - Ax^{(1)}) = \lambda^{(1)} \sum_{i=1}^{m} \text{sgn}(x_i^{(1)}) \cdot v_i \). Thus

\[
A_{\hat{g}(1)}^T (y - Ax^{(1)}) = \lambda^{(1)} \cdot \text{sgn}(x_i^{(1)}) \cdot v_i
\]

which contradicts the KKT conditions at \( \lambda = \lambda^{(1)} \).

2. If \( \sum_{i=1}^{m} \text{sgn}(x_i^{(1)}) \cdot v_i = \text{sgn}(x_i^{(1)}) \), from the KKT conditions at \( \lambda = \lambda^{(1)} \), we obtain

\[
v^T A_{\hat{g}(1)} (y - Ax^{(1)}) = \lambda^{(1)} \sum_{i=1}^{m} \text{sgn}(x_i^{(1)}) \cdot v_i
\]

\[
= \lambda^{(1)} \sum_{i=1}^{m} \text{sgn}(x_i^{(1)}) \cdot v_i
\]

which means that the \( (m+1) \) th index has entered the active set at \( \lambda = \lambda^{(1)} \). It is a contradiction to the fact that \( A_{\hat{g}(1)} \) is of full column rank.

To complete the induction argument, it remains only to show that there exists \( \lambda \) (denoted as \( \lambda^{(1)} \)) such that the corresponding active set \( \hat{E} = E^{(1)} \) and \( \hat{x}_i \neq 0, \forall i \in \hat{E} \). We update \( \lambda^{(1)} \) with a small step \( \delta \) (i.e., \( \lambda = \lambda^{(1)} + \delta \)), then the corresponding optimal solution becomes

\[
\hat{x} = [x_1^{(1)} + e_1, x_2^{(1)} + e_2, \ldots, x_m^{(1)} + e_m, e_{m+1}, e_{m+2}, \ldots, e_n]^T
\]

With this result, we claim that \( \hat{x}_{m+1} \neq 0 \). Otherwise, the linear relationship in function \( g(\lambda) \) will remain unchanged, since \( \hat{x}_{m+1} \) keeps being 0. Then, we have

- If \( g_1(\lambda^{(1)} + \delta) = 0, g(\lambda) < g_1(\lambda^{(1)} + \delta) = 0 \);
- If \( g_2(\lambda^{(1)} + \delta) = 0, g(\lambda) > g_2(\lambda^{(1)} + \delta) = 0 \).

Both candidates contradict \( \hat{x}_{m+1} = 0 \). Hence, we prove the claim of \( \hat{x}_{m+1} \neq 0 \). Therefore the induction argument holds.

9.2 Appendix 2: Proof of Lemma 2

Proof: As the proof for the dropping event is trivial, we only show the proof for the adding event. The rest proof is achieved by contradiction. Without loss of generality, let \( E^{(1)} = \{1, 2, \ldots, m\} \), and \( (m + 1) \) be the new index entering into \( E^{(1)} \). Assume that there exists a non-zero vector \( v \) such that

\[
A_{\hat{g}(1)} v = \hat{A}_{m+1}
\]

and then from (15), we have

\[
\hat{A}_{m+1} v = \hat{A}_{m+1} + \eta
\]

where \( \eta \in \mathbb{R}^{p \times 1} \) is a vector determined by the random ridge (\( \eta \to 0 \)) when \( \Gamma \to 0 \). \( v \) is independent on the random ridge \( \Gamma \), and is uniquely determined by (38). The KKT condition (18) for the \( (m+1) \) th index can be written by

\[
A_{\hat{g}(1)} (y - Ax^{(1)}) = \lambda^{(1)} \cdot \text{sgn}(x_{m+1})
\]

Using (18) and (39), we have

\[
\lambda^{(1)} \cdot \sum_{i=1}^{m} \text{sgn}(x_i^{(1)}) \cdot v_i = v^T \hat{A}_{m+1}^T (y - Ax^{(1)})
\]

or, equivalently

\[
A_{\hat{g}(1)}^T (y - Ax^{(1)}) = \lambda^{(1)} \cdot \sum_{i=1}^{m} \text{sgn}(x_i^{(1)}) \cdot v_i - \eta^T (y - Ax^{(1)})
\]

For the purpose of easy exposition and explanation, let \( \kappa \triangleq \left( \sum_{i=1}^{m} \text{sgn}(x_i^{(1)}) \right) - \text{sgn}(x_{m+1}) \). Obviously, \( \kappa \) has a finite value, which is also independent on the random ridge \( \Gamma \). Then, based on whether \( \kappa \) is positive, negative or zero, the following three scenarios can happen.

- If \( \kappa > 0 \), (42) leads to

\[
\hat{A}_{m+1}^T (y - Ax^{(1)}) = \lambda^{(1)} \cdot \text{sgn}(x_{m+1}) + \kappa \lambda^{(1)}
\]

\[
- \eta^T (y - Ax^{(1)})
\]

\[
> \lambda^{(1)} \cdot \text{sgn}(x_{m+1})
\]
where the last inequality follows from the fact that $\kappa l(l+1)$ is finite and positive; whereas $\eta^T (y - \hat{A}x(l+1))$ tends to zero when the random ridge $\Gamma_y$ is sufficiently small.

- Similarly, if $\kappa < 0$, we obtain

$$
\hat{A}_{m+1}^T (y - \hat{A}x(l+1)) = \lambda^{(l+1)} \cdot \sgn(x^{(l+1)}_{m+1}) + \kappa \lambda^{(l+1)} \\
- \eta^T (y - \hat{A}x^{(l+1)}) \\
< \lambda^{(l+1)} \cdot \sgn(x^{(l+1)}_{m+1})
$$

(44)

where $\kappa \lambda^{(l+1)}$ is finite and negative.

- Finally, we consider that $\kappa = 0$. Since $\eta$ is random and $y - A^{(l+1)} x \neq 0$, it is with measure 1 that

$$
\hat{A}_{m+1}^T (y - \hat{A}x^{(l+1)}) = \lambda^{(l+1)} \cdot \sgn(x^{(l+1)}_{m+1}) \\
- \eta^T (y - \hat{A}x^{(l+1)}) \neq \lambda^{(l+1)} \cdot \sgn(x^{(l+1)}_{m+1})
$$

(45)

All equalities (43), (44) and (45) contradict to (40). Therefore the assumption in (38) cannot be true. It turns out that $A_{l+1}$ is of full column rank for any sufficiently small random ridge almost surely. $\square$