Solving Rank-Deficient Linear Least-Squares Problems*

Thomas F. Coleman[1] and Chunguang Sun[2]

Abstract

Numerical solution of linear least-squares problems is a key computational task in science and engineering. Effective algorithms have been developed for the linear least-squares problems in which the underlying matrices have full rank and are well-conditioned. However, there are few efficient and robust approaches to solving the linear least-squares problems in which the underlying matrices are rank-deficient and sparse. In this paper, we propose a new method for solving rank-deficient linear least-squares problems. Our proposed method is mathematically equivalent to an existing method but has several practical advantages over the existing method. Furthermore, our proposed method is applicable to solving both dense and sparse rank-deficient linear least-squares problems. Our experimental results demonstrate the practical potential of our proposed method.

Section 1: Introduction

Let $A \in R^{m \times n}, b \in R^m$. The numerical solution of the linear least-squares problem

$$\min_{x \in R^n} \|Ax - b\|_2 \tag{1.1}$$

lies at the heart of many computational problems frequently arising in scientific, engineering, and economic disciplines. Efficient algorithms are available when the matrix A has full rank and is well-conditioned. However, when the matrix is ill-conditioned or rank-deficient, numerical solution of (1.1) often requires some variant of rank-revealing QR factorization (RRQR) or singular value decomposition (SVD). The resulting solution process is relatively expensive. Furthermore, the solution to (1.1) is not unique. Usually, the *minimum-norm* solution is sought.

In practical applications, the matrix A is often large and sparse. If A is also rank-deficient, there are few effective algorithms available for solving (1.1). Because of the paramount need for sparsity preservation, the algorithms involving RRQR or SVD developed for dense linear least squares problems are not suitable for sparse linear least squares problems.

---

*This work was supported by the National Sciences and Engineering Research Council of Canada. The views expressed herein are solely from the authors.

1. Department of Combinatorics and Optimization, University of Waterloo, 200 University Avenue West, Waterloo, ON, N2L 3G1, Canada (tfcoleman@uwaterloo.ca)

2. Faculty of Mathematics, University of Waterloo, 200 University Avenue West, Waterloo, ON, N2L 3G1, Canada (c23sun@uwaterloo.ca)

Effective approaches (George, et al., 1980)(Heath, 1982)(Heath, 1984)(Liu, 1986)(Sun, 1996) (Sun, 1997) have been developed to solve sparse linear least-squares problem when the underlying matrix has full rank and is well-conditioned. Issues in handling rank deficiency in solving sparse linear least-squares problems are considered in (Ng, 1991) and (Avron, et al., 2009).

In this paper, we propose a new method for solving rank-deficient linear least-squares problems. We show that our proposed method is mathematically equivalent to an existing method. If we view both our method and the existing method as generating a sequence of points (i.e. approximate solutions) approaching the true solution, the two methods are mathematically equivalent in the sense that they generate the same sequence of points in exact arithmetic. However, there are two major differences between the two methods. First, the theoretical underpinnings of the two methods are very different. In particular, the mathematical derivations of the two methods are drastically different. Second, the computations at each iterative step are organized differently. Because of these differences, our proposed method offers practical advantages over the existing method in terms of algorithmic efficiency and applicability to handling both dense and sparse rank-deficient linear least-squares problems.

We focus our attention upon the design and analysis of our new method for solving dense and rank-deficient linear least-squares problems in this paper. We outline our approach to the solution of sparse and rank-deficient linear least-squares problems. However, detailed results on the sparse problems will be presented in (Coleman, et al., 2010).

In Section 2 we propose a new algorithm for solving rank-deficient linear least-squares problems. We prove that our proposed algorithm is mathematically equivalent to an existing algorithm in Section 3. In Section 4, we discuss the selection of a crucial parameter used in our algorithm. In Section 5, we compare the practical performance of our algorithm with that of the existing algorithm. Finally, we discuss future work and summarize our results in Section 6.

Notations. Assume $k = rank(A)$. Let $A = U\Sigma V^T$ be the SVD of A. Assuming $\Sigma_1$is the leading submatrix of Σ corresponding to the k positive singular values, the compact SVD of A can be written as $A = U_1\Sigma_1 V_1^T$, where

$$U_1 = [u_1, u_2, \dots, u_k],\ V_1 = [v_1, v_2, \dots, v_k],$$

and

$$\Sigma_1 = \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_k \end{pmatrix},$$

where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k > 0$.

The *minimum-norm* solution to the above linear least-squares problem (1.1) is given by

$$\hat{x} = A^{+}b = V_1 \Sigma_1^{-1} U_1^T b,$$

where $A^{+}$is the pseudo-inverse of A. Let $\hat{r} = b - A\hat{x}$.

Section 2: A New Method for Handling Rank Deficiency

For $\lambda > 0$ define

$$x(\lambda) = (A^T A + \lambda I)^{-1} A^T b \tag{2.1}$$

Using the compact SVD, (2.1) can be written

$$x(\lambda) = V_1 (\Sigma_1^2 + \lambda I)^{-1} (\Sigma_1 U_1^T b) \tag{2.2}$$

From (2.2) it is clear that $x(\lambda) \to \hat{x} = V_1 \Sigma_1^{-1} U_1^T b$ as $\lambda \to 0$.

An *approximate minimum-norm* least-squares solution to (1.1) is therefore given by (2.2) or, equivalently, the least-squares solution to the following full-rank problem:

$$\min_{x \in R^n} \left\| \begin{pmatrix} A \\ \lambda^{\frac{1}{2}} I \end{pmatrix} x - \begin{pmatrix} b \\ 0 \end{pmatrix} \right\|_2 \tag{2.3}$$

While there are several numerical approaches to (2.3), the stable QR factorization method can be used, perhaps with the use of a permutation matrix Π to limit fill in the upper triangular matrix R_π (in the case where matrix *A* is sparse):

$$Q^T \begin{pmatrix} A \\ \lambda^{\frac{1}{2}} I \end{pmatrix} \Pi = \begin{pmatrix} R_\Pi \\ 0 \end{pmatrix} \tag{2.4}$$

One general approach to solving the rank-deficient linear least-squares problem is to choose a positive value for λ, solve (2.3) by using the factorization in (2.4) to obtain $x(\lambda)$, and then refine $x(\lambda)$ to get (or approximate) $x(0) = \hat{x}$. The question we address next is how to effectively do this refinement. We discuss our approach to choosing the parameter λ in Section 4.

Let $M = A^T A$, $d = A^T b$. The solution to the full-rank problem (2.3) is the same as the solution to the following system of semi-normal equations:

$$(A^T A + \lambda I)x = A^T b$$

or

$$(M + \lambda I)x = d \tag{2.5}$$

which obviously yields the solution

$$x = (M + \lambda I)^{-1} d \tag{2.6}$$

Differentiating (2.5) with respect to λ yields

$$x + \left(M + \lambda I\right) x^{'} = 0 \tag{2.7}$$

or $x^{'} = -(M + \lambda I)^{-1} x = -(M + \lambda I)^{-2} d.$

Differentiating (2.7) with respect to λ yields

$$2x^{'} + \left(M + \lambda I\right) x^{''} = 0 \tag{2.8}$$

or, $x^{''} = -2(M + \lambda I)^{-1} x^{'} = 2(M + \lambda I)^{-3} d$.

Generally,

$$x^{(k)} = -k(M + \lambda I)^{-1} x^{(k-1)}. \tag{2.9}$$

By Taylor's theorem,

$$x\left(0\right) = x + \sum_{k=1}^{\infty} (-1)^k s_k \lambda^k \tag{2.10}$$

where $x \triangleq x(\lambda),\ s_k \triangleq \frac{1}{k!} x^{(k)}(\lambda),\ (k = 1, 2, \ldots).$

Note that by (2.9)

$$s_k = \frac{-1}{(k-1)!}(M + \lambda I)^{-1} x^{(k-1)} = -(M + \lambda I)^{-1} s_{k-1}, \quad (k = 2, 3, \ldots) \tag{2.11}$$

where $s_1 = x^{'} = -(M + \lambda I)^{-1} x.$

An important computational observation is that given s_{k-1} and $\Pi^{\mathrm{T}}(\mathrm{M}+\lambda\mathrm{I})\Pi = \mathrm{R}_{\Pi}^{\mathrm{T}}\mathrm{R}_{\Pi}$, vector s_k can be computed with two triangular solves, and thus in time $O\big(nnz(R_{\Pi})\big)$, where $nnz(R_{\Pi})$ is the number of nonzeroes in the matrix R_{Π}.

Computationally, our algorithm can be described as follows:

1) $x_0 = x(\lambda)$
2) $s_0 = x_0$
3) $for\ i = 1, 2, 3, \ldots$
   $s_i = -(M + \lambda I)^{-1} s_{i-1}$
   $x_i = x_{i-1} + (-1)^i s_i \lambda^i$

At the i-th refinement step, $s_i = -(M + \lambda I)^{-1} s_{i-1}$ is accomplished by solving $R^T R s_i = -s_{i-1}$, where $M + \lambda I = R^T R$. In practice, this version of our algorithm is numerically unstable since the product $s_i \lambda^i$ is formed at each refinement step. Typically, $||s_i||_2$ is very large and λ^i very small.

To overcome the numerical instability, we revise our algorithm as follows:

1) $x_0 = x(\lambda)$
2) $t_0 = x_0$
3) $for\ i = 1, 2, 3, \ldots$
   $t = \lambda t_{i-1}$
   $t_i = (M + \lambda I)^{-1} t$
   $x_i = x_{i-1} + t_i$

It is straightforward to show that

$$t_i = (-1)^i s_i \lambda^i, \qquad i = 1, 2, 3, \ldots$$

via mathematical induction. Therefore, our revised algorithm is equivalent to our original algorithm. However, our revised algorithm avoids forming the product $s_i \lambda^i$ at each refinement step. Therefore, the issue of numerical instability in the original algorithm has been resolved.

Now we discuss the implementation issues of our algorithm. For a given $\lambda > 0$, step 1) of our algorithm is accomplished by solving the full-rank linear least-squares problem (2.3). Let

$$C = \begin{pmatrix} A \\ \lambda^{\frac{1}{2}} I \end{pmatrix} \tag{2.12}$$

and

$$f = \begin{pmatrix} b \\ 0 \end{pmatrix}. \tag{2.13}$$

Then problem (2.3) can be written as

$$\min_{x \in R^n} \|Cx - f\|_2 \tag{2.14}$$

where C is an $(m+n)$-by-n matrix and f is an $(m+n)$-vector. First, we compute the "economy size" QR factorization:

$$C = QR, \tag{2.15}$$

where Q is an $(m+n)$-by-n matrix with orthonormal columns and R is an n-by-n upper triangular matrix. Let

$$Q = \begin{pmatrix} Q_I \\ Q_{II} \end{pmatrix} \tag{2.16}$$

where Q_I is an m-by-n matrix and Q_{II} is an n-by-n matrix. Second, the solution to (2.3) (i.e. (2.14)), denoted by x_0 or $x(\lambda)$, is obtained by performing a matrix-vector multiply followed by a triangular solve as follows:

$$x_0 = x(\lambda) = R^{-1}(Q_I^T b). \tag{2.17}$$

Mathematically, we have

$$(M + \lambda I) = C^T C = R^T R. \tag{2.18}$$

Hence, at the i-th refinement step, $t_i = (M + \lambda I)^{-1} t$ is accomplished by two triangular solves:

$$R^T R t_i = t. \tag{2.19}$$

Clearly, each iterative refinement step of our algorithm requires precisely two triangular solves, one scalar-vector multiply, and the addition of two vectors.

It is important to note that the QR factorization (2.15) is computed only once. The overall complexity of our algorithm consists of the arithmetic work for solving (2.3) (which consists of the arithmetic work for the QR factorization (2.15) and the arithmetic work for (2.17)) and the arithmetic work for N iterative refinement steps, where N is the total number of iterative refinement steps performed. As shown in Section 5, N is typically less than or around 10.

Section 3: Mathematical Equivalence of Our Algorithm with an Existing Algorithm

In this section we show that our algorithm is mathematically equivalent to an existing algorithm in the sense that both algorithms generate the same sequence of points (i.e. approximate solutions) approaching the true solution.

Riley (Riley, 1956) proposed the following iterative scheme (referred to as Iterated Regularization scheme or **IR scheme** for short) for solving linear least-squares problems with full column rank. Let $x^{(0)}$ be an arbitrary vector, solve

$$\left(A^T A + \lambda I\right) x^{(q+1)} = A^T b + \lambda x^{(q)} \tag{3.1}$$

The sequence $x^{(q)}$ converges to $\hat{x}$ (the true solution) if $\lambda > 0$ since the spectral radius of $\lambda(A^T A + \lambda I)^{-1}$ is less than 1.

Golub (Golub, 1965) observed that the Riley's iterative scheme is equivalent to the following:

$$\begin{aligned} r^{(q)} &= b - Ax^{(q)} \\ \left(A^T A + \lambda I\right) e^{(q)} &= A^T r^{(q)} \\ x^{(q+1)} &= x^{(q)} + e^{(q)} \end{aligned} \tag{3.2}$$

The vector $e^{(q)}$ is the solution of the following linear least-squares problem:

$$\min_{e^{(q)} \in R^n} \| Ce^{(q)} - f^{(q)} \|_2 \tag{3.3}$$

where

$$C = \begin{pmatrix} A \\ \lambda^{\frac{1}{2}} I \end{pmatrix} \tag{3.4}$$

and

$$f^{(q)} = \begin{pmatrix} r^{(q)} \\ 0 \end{pmatrix} \tag{3.5}$$

Therefore, the IR scheme can be described as follows:

$x^{(0)} = 0$
$for\ q = 0, 1, 2, 3, \ldots$

a) $r^{(q)} = b - Ax^{(q)}$
b) Solve the linear least-squares problem (3.3).
c) $x^{(q+1)} = x^{(q)} + e^{(q)}$

Golub (Golub, 1965) has shown that

$$x^{(q)} = \mu_1^{(q)} v_1 + \mu_2^{(q)} v_2 + \cdots + \mu_k^{(q)} v_k$$

where

$$\mu_j^{(q)} = \left[1 - \left(\frac{\lambda}{\lambda + \sigma_j^2}\right)^q\right] \frac{u_i^T b}{\sigma_j}$$

for $j = 1, 2, \ldots, k$.

Therefore, as $q \to \infty$

$$x^{(q)} = \frac{u_1^T b}{\sigma_1} v_1 + \frac{u_2^T b}{\sigma_2} v_2 + \cdots + \frac{u_k^T b}{\sigma_k} v_k = \hat{x}$$

As indicated in (Golub, 1965), it is easy to show that

$$e^{(q+1)} = \lambda (A^T A + \lambda I)^{-1} e^{(q)}$$

and

$$\left\| e^{(q+1)} \right\|_2 \leq \frac{\lambda}{\lambda + \sigma_k^2} \left\| e^{(q)} \right\|_2 < \left\| e^{(q)} \right\|_2$$

Therefore, a good termination procedure is to stop the iterative process as soon as $\left\| e^{(q)} \right\|_2$ increases or doesn't change.

Now we prove that the IR scheme as formulated in (Golub, 1965) is equivalent to our algorithm. The IR scheme shows that

$$x^{(q)} = e^{(0)} + e^{(1)} + e^{(2)} \ldots + e^{(q)},$$

where $x^{(q)}$ is the solution vector computed by the IR scheme after q iterative steps. Clearly, $e^{(0)} = x_0 = x(\lambda)$. Therefore

$$x^{(q)} = x_0 + e^{(1)} + e^{(2)} \ldots + e^{(q)}.$$

Our algorithm shows that

$$x_q = x_0 + t_1 + t_2 \ldots + t_q$$

where x_q is the solution vector computed by our algorithm after q iterative steps. We prove

$$x^{(q)} = x_q \left(q = 1, 2, 3, \ldots\right) \tag{3.6}$$

by showing

$$e^{(q)} = t_q \left(q = 1,2,3,\ldots\right). \tag{3.7}$$

We can easily derive the following from our algorithm:

$$t_q = \lambda^q (M + \lambda I)^{-(q+1)} d \tag{3.8}$$

Let

$$G = \lambda (A^T A + \lambda I)^{-1}$$

and

$$h = (A^T A + \lambda I)^{-1} A^T b$$

It has been shown in (Golub, 1965) that

$$x^{(q+1)} = \left(G^q + G^{q-1} + \ldots + G + I\right) h \tag{3.9}$$

Therefore,

$$\begin{aligned} e^{(q)} &= x^{(q+1)} - x^{(q)} \\ &= (G^q + G^{q-1} + \cdots + G + I)h - (G^{q-1} + G^{q-2} + \cdots + G + I)h \\ &= G^q h \end{aligned}$$

Thus,

$$e^{(q)} = G^q h = [\lambda (A^T A + \lambda I)^{-1}]^q (A^T A + \lambda I)^{-1} A^T b = \lambda^q \left(M + \lambda I\right)^{-(q+1)} d \tag{3.10}$$

Combing (3.8) and (3.10), we obtain (3.7). Therefore, we have proved (3.6). In other words, the IR scheme is equivalent to our algorithm in the sense that both algorithms generate the same sequence of approximate solutions approaching the true solution in exact arithmetic. Clearly, the mathematical derivations of the two algorithms are very different. Our algorithm relies on the application of the Taylor series expansion while the existing algorithm is an iterated regularization method. Furthermore, the computations at each iterative step are different. In Section 5, we will demonstrate that our algorithm offers several practical advantages over the IR scheme.

Section 4: Selection of Lambda

Given that the singular values of A are $\sigma_1, \sigma_2, \ldots, \sigma_k, 0, \ldots, 0$. It is easy to show that the singular values of the extended matrix C as defined in (2.12) are as follows:

$$\sqrt{(\sigma_1^2 + \lambda)}, \sqrt{(\sigma_2^2 + \lambda)}, \ldots, \sqrt{(\sigma_k^2 + \lambda)}, \sqrt{\lambda}, \ldots, \sqrt{\lambda}$$

Therefore, the conditioning number of the extended matrix C

$$cond(C) = cond(R) = \sqrt{\frac{\sigma_1^2 + \lambda}{\lambda}},$$

where $C = QR$ is the QR factorization of C as defined in (2.15). Clearly, the conditioning number of R will be large if λ is small. This will have negative impact on the numerical accuracy of solving triangular systems involving R as required by both our algorithm and the existing algorithm. Hence, λ cannot be too small. On the other hand, the following discussion shows that λ cannot be too large either.

Let δ be a lower bound of the smallest non-zero singular value σ_k. Golub (Golub, 1965) suggested that λ be chosen so that

$$\frac{\lambda}{\lambda + \delta^2} < 0.1 \tag{4.1}$$

Clearly, a wide range of choices for λ and δ would satisfy (4.1). Our approach is to choose the greatest lower bound for σ_k, which is σ_k itself. Hence, we have

$$\frac{\lambda}{\lambda + \sigma_k^2} < 0.1 \tag{4.2}$$

Obviously, an infinite number of choices for λ would satisfy (4.2). In particular, we choose $\lambda = \beta\sigma_k^2$. Then

$$\frac{\beta\sigma_k^2}{\beta\sigma_k^2 + \sigma_k^2} = \frac{\beta}{\beta + 1} < 0.1 \tag{4.3}$$

Thus,

$$\beta < \frac{1}{9}$$

Our experimental results have shown that $\beta = 0.01$ produces satisfactory results. So our choice for λ is $0.01\sigma_k^2$. To distinguish different choices for λ in the following discussion, let

$$\lambda_{svd} = 0.01\sigma_k^2.$$

It is well-known that computing the SVD of a matrix is an expensive process. At least, it is considerably more expensive than computing the QR factorization of the same matrix. So our approach avoids computing the SVD of the given matrix. Hence, we have no direct knowledge of σ_k. In other words, we cannot compute λ_{svd} exactly. Our approach to choosing λ is to obtain an approximate value for λ_{svd}. Our approach is described as follows:

- Compute the QR factorization of A: $A = Q_1 R_1$, where Q_1 is m-by-m, and R_1 is m-by-n.

- Let W denote the set of absolute values of the nonzero diagonal elements of R_1. Let ω_{min} and ω_{max} denote the smallest and largest elements of W, respectively.

  Both $\lambda_1 = \hat{\beta}\omega_{min}^2$ and $\lambda_2 = \hat{\beta}\frac{\omega_{min}^2}{\omega_{max}^2}$, where $\hat{\beta} = 0.00025$ produce satisfactory results. Furthermore, we observe that

$$\lambda_{QR} = \frac{1}{2}(\lambda_1 + \lambda_2) = \hat{\beta}\frac{\omega_{min}^2}{\omega_{max}^2}\frac{(\omega_{max}^2 + 1)}{2}$$

  produces improved results over λ_1 or λ_2.

The following table shows λ_{svd} and λ_{QR} for six dense problems. Except for the problem 600x300, λ_{QR} is a good approximation for λ_{svd}.

Dense Problems	rank	λ_{svd}	λ_{QR}
600x300	238	2.2441e-11	6.0530e-14
1500x1000	812	8.2283e-16	5.1143e-16
2000x1500	1212	3.7025e-17	2.4798e-17
3000x2000	1632	1.6702e-21	4.0261e-20
3500x500	383	1.8902e-19	5.2419e-19
3500x1000	812	9.4053e-18	8.5434e-18

Section 5: Performance Comparison

First, we consider the implementation details of the IR scheme. Each iterative step of the IR scheme solves a linear least-squares problem with the same $(m+n)$-by-n matrix C as defined in (2.12). As in our algorithm, the (economy) QR factorization $C = QR$ defined in (2.15) is computed only once. Assume that the matrix Q is partitioned as in (2.16).

There are two possible implementations of the IR scheme:

- Option 1: The matrix Q is saved (actually, it is only necessary to save the submatrix Q_I). Then at the q-th iterative step, we solve the following triangular system:

$$Re^{(q)} = Q^T f^{(q)} = Q_I^T r^{(q)}$$

Compared with our algorithm, this implementation of the IR scheme performs two extra matrix-vector multiplications $Ax^{(q)}$ and $Q_I^T r^{(q)}$ per iterative step. But it solves only one $n \times n$ triangular system instead of two as in our algorithm.

If A is sparse, Q(or Q_I) is often dense. Therefore, saving Q(or Q_I) would not be practical for sparse linear least-squares problems. Evidently, this implementation of the IR scheme is not suitable for solving sparse linear least-squares problems.

- Option 2: The matrix Q(or Q_I) is not saved. Then at the q-th iterative step, we solve the following system of semi-normal equations (SNE):

$$R^T R e^{(q)} = A^T r^{(q)}$$

Compared with our algorithm, this option performs two extra matrix-vector multiplications $Ax^{(q)}$ and $A^T r^{(q)}$ per iterative step. In other words, our algorithm is more efficient than this implementation of the IR scheme.

In the following discussion, Option 1 and Option 2 will be referred to as the **Algorithm IR-Q** and the **Algorithm IR-SNE**, respectively.

All three algorithms solve exactly the same extended linear least-squares problem (2.3). The running time for solving (2.3) includes running times for forming the extended matrix C, computing the (economy) QR factorization of C, and solving the resulting triangular system $Rx_0 = Q_I^T b$. The following table shows the running times for solving (2.3) on six dense problems

Dense Problems	Running Times for Solving the Extended LS Problem (Seconds)
600x300	0.0550
1500x1000	0.7882
2000x1500	2.1226
3000x2000	4.9765
3500x500	0.5450
3500x1000	1.5034

Now we compare running times of the three algorithms for the iterative refinement process. We measure the running time for ten iterative steps for each of the three algorithms. The running times are summarized in the following table.

Dense Problems	Running Times for 10 Iterative Steps (Seconds)		
	IR-Q	IR-SNE	Our Algorithm
600x300	0.0031	0.0103	0.0084
1500x1000	0.0593	0.1685	0.1210
2000x1500	0.1137	0.3640	0.2813
3000x2000	0.2111	0.6392	0.4836
3500x500	0.0506	0.0799	**0.0229**
3500x1000	0.1077	0.2149	0.1166

Overall, IR-Q is the most efficient approach since it performs only one triangular solve per iterative step. However, when the matrix A is very skinny (that is $m \gg n$, e.g. 3500x500), it is not as efficient as our algorithm. The reason is that the forming the matrix-vector products $Ax^{(q)}$ and $Q_I^T r^{(q)}$ would be more expensive than a single triangular solve. The most severe drawback of Algorithm IR-Q lies in the fact that it is not suitable for solving sparse linear least-squares problems.

Clearly, our algorithm is more efficient than IR-SNE since IR-SNE performs two extra matrix-vector multiplications $Ax^{(q)}$ and $A^T r^{(q)}$ per iterative step. Furthermore, our algorithm becomes increasingly more efficient as the underlying matrix gets skinnier.

We used $\lambda = \lambda_{QR}$, discussed in the previous section, in our experiments. All three algorithms converge in about the same number of iterative steps. They converge under ten iterations for five of the six test problems. They converge in 12 iterations for the remaining test problem 3000-by-2000.

As suggested in (Golub, 1965), a good termination criterion is to measure the size of the update vector t_q at each step. The iterative process should stop as soon as $\|t_q\|_2$ doesn't change or increases. For both algorithm IR-Q and our algorithm, once they converge, the following relative change to x_q being computed

$$\frac{\|t_q\|_2}{\|x_q\|_2} \tag{5.1}$$

remains unchanged (at least the first five significant digits of the ratio (5.1) remain unchanged) for a considerable number of subsequent iterations. This is a clear signal for terminating the iterative process. In practice, as soon as the first five significant digits of the ratio (5.1) of two consecutive iterations agree, the iterative process terminates. This stopping criterion can be easily implemented. It is worth noting that algorithm IR-SNE doesn't have this convergent property.

IR-Q and our algorithm are virtually identical in terms of numerical properties. Using the stopping criterion outlined above, the number of steps required for convergence for both IR-Q and our algorithm are given in the following table:

Dense Problems	Number of Steps for Convergence (IR-Q and our algorithm)
600x300	3
1500x1000	6
2000x1500	5
3000x2000	12
3500x500	6
3500x1000	5

The algorithm IR-SNE also converges around the same number of steps as shown in the above table. However, the termination criterion for IR_SNE is not as easy to implement as the other two algorithms because of the lack of the convergent property discussed above.

Let

$$r^{(q)} = r_q \ = b - Ax^{(q)} = b - Ax_q \quad (q = 1, 2, 3, \dots)$$

Define

$$RE(x_q) = \frac{\|\hat{x} - x_q\|_2}{||\hat{x}||_2}$$

and

$$RE(r_q) = \frac{\|\hat{r} - r_q\|_2}{||\hat{r}||_2}$$

for $q \geq 0$. RE stands for Relative Error.

The following three tables show experimental results on the test problem 2000-by-1500. In the tables, column 1 represents iterative steps (from 0 to 10), column 2 shows $RE(x_q)$, and column 3 shows $RE(r_q)$. The last column shows $||t_q||_2/||x_q||_2$.

Table 1: Dense Problem 2000-by-1500 (Our Algorithm)

IT	$RE(x_q)$	$RE(r_q)$	$\|\|t_q\|\|_2/\|\|x_q\|\|_2$
0	6.3065e-003	2.4782e-004	0
1	4.2058e-005	1.6469e-006	6.3023e-003
2	4.3457e-006	1.1063e-008	4.1703e-005
3	5.7820e-006	2.8667e-010	1.4719e-006
4	7.2276e-006	2.4866e-010	1.4455e-006
5	8.6731e-006	2.5100e-010	1.4455e-006
6	1.0119e-005	2.5207e-010	1.4455e-006
7	1.1564e-005	2.5067e-010	1.4455e-006
8	1.3010e-005	2.5192e-010	1.4455e-006
9	1.4455e-005	2.5199e-010	1.4455e-006
10	1.5901e-005	2.5323e-010	1.4455e-006

Table 2: Dense Problem 2000-by-1500 (Algorithm IR-Q)

IT	$RE(x_q)$	$RE(r_q)$	$\|\|t_q\|\|_2/\|\|x_q\|\|_2$
0	6.3065e-003	2.4782e-004	0
1	4.2058e-005	1.6469e-006	6.2648e-003
2	4.3457e-006	1.1058e-008	4.1702e-005
3	5.7820e-006	2.9249e-010	1.4719e-006
4	7.2276e-006	2.4562e-010	1.4455e-006
5	8.6731e-006	2.4256e-010	1.4455e-006
6	1.0119e-005	2.4222e-010	1.4455e-006
7	1.1564e-005	2.4858e-010	1.4455e-006
8	1.3010e-005	2.4483e-010	1.4455e-006
9	1.4455e-005	2.4470e-010	1.4455e-006
10	1.5901e-005	2.4480e-010	1.4455e-006

Table 3: Dense Problem 2000-by-1500 (Algorithm IR-SNE)

IT	$RE(x_q)$	$RE(r_q)$	$\|\|t_q\|\|_2/\|\|x_q\|\|_2$
0	6.3065e-003	2.4782e-004	0
1	4.2098e-005	1.6469e-006	6.2648e-003
2	4.9380e-006	1.1022e-008	4.1746e-005
3	6.2523e-006	3.0371e-010	2.3706e-006
4	7.3532e-006	2.4815e-010	2.1699e-006
5	8.4908e-006	2.5428e-010	2.4117e-006
6	9.7705e-006	2.9390e-010	2.3501e-006
7	1.0947e-005	2.6434e-010	2.3575e-006
8	1.2334e-005	2.2960e-010	2.4873e-006
9	1.3579e-005	2.6337e-010	2.3336e-006
10	1.4774e-005	2.4588e-010	2.3539e-006

Section 6: Future Work and Concluding Remarks

Efficiency Enhancement

In our implementations, we have computed two separate QR decompositions -- the QR decomposition of the original matrix A to obtain λ_{QR} and the QR decomposition of the extended matrix C to obtain $x(\lambda)$. We plan to explore the approach in which the QR decomposition of C is not computed from scratch rather it is built upon the QR decomposition of A. The problem will then become how to compute the following QR decomposition efficiently

$$\begin{pmatrix} R_1 \\ \lambda^{\frac{1}{2}} I \end{pmatrix} = \hat{Q}R \tag{6.1}$$

where $A = Q_1 R_1$ is the QR decomposition of the matrix A. Mathematically,

$$C = \begin{pmatrix} A \\ \lambda^{\frac{1}{2}} I \end{pmatrix} = \begin{pmatrix} Q_1 R_1 \\ \lambda^{\frac{1}{2}} I \end{pmatrix} = \begin{pmatrix} Q_1 & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} R_1 \\ \lambda^{\frac{1}{2}} I \end{pmatrix} = \begin{pmatrix} Q_1 & 0 \\ 0 & I \end{pmatrix} \hat{Q}R = QR \tag{6.2}$$

In this context, the matrix $\sqrt{\lambda} I$ is considered an n-by-n upper triangular matrix. Depending on whether R_1 is stored row-by-row or column-by-column, the QR decomposition defined in (6.1) can be computed by a sequence of Givens rotations or a sequence of Householder reflections.

If R_1 is dense, we need $n(n+1)/2$ Givens rotations or n Householder reflections to perform the QR decomposition (6.1). If R_1 is sparse, choosing an appropriate storage scheme for R_1 is critical. Assume that R_1 is stored row-by-row, we need $nnz(R_1) = nnz(R)$ Givens rotations to compute the QR factorization. In this situation, Householder reflections would be very inefficient.

Note that the sparsity structure of R_1 is the same as that of R. We can determine the sparsity structure of R in a preliminary step. Therefore, we don't need to compute the sparsity structure of R again in carrying out (6.2). Furthermore, the sparsity structure R can fully accommodate the process of merging $\sqrt{\lambda} I$ into R_1 without the need to dynamically allocate additional storage for R.

We plan to investigate whether the proposed method of combining the two QR decompositions is more efficient than computing the two QR decompositions separately for both dense and sparse problems.

The following describes our overall approach to solving rank-deficient linear least-squares problems.

1. For a given m-by-n rank-deficient matrix A, compute the QR decomposition $A = Q_1 R_1$. The matrix Q_1 is not saved if A is sparse.

2. Calculate λ_{QR} as described in Section 4.

3. Compute the QR decomposition of the extended matrix C as defined in (2.12) from scratch or using the method of combining the two QR decompositions described above.

4. Compute $x_0 = x(\lambda)$ using the results of the QR decomposition of C.

5. Perform the iterative refinement process.

The application of our algorithm to the solution of sparse and rank-deficient linear least-squares problems is discussed in (Coleman, et al., 2010).

Sparse Linear Least-Squares Problems with Dense Rows

In practice, a sparse linear least-squares problem frequently contains a number of dense or nearly dense rows. The corresponding upper triangular factor R becomes nearly or completely full. Therefore, the straightforward application of the QR decomposition method is not a viable approach to the solution of the sparse least-squares problems with dense rows. Heath (Heath, 1982) proposed a method for handling dense rows. Through row permutation, a sparse linear least-squares problem with dense rows can be easily transformed into the following equivalent problem

$$\min_{x \in R^n} \left\| \begin{pmatrix} A \\ E \end{pmatrix} x - \begin{pmatrix} b \\ f \end{pmatrix} \right\|_2 \tag{6.3}$$

where A consists of sparse rows and E the dense rows of the original matrix, respectively. The solution to (6.3) is obtained by first computing the solution to the sparse linear least-squares problem $\min_x \|Ax - b\|_2$ as usual and then updating the solution by the rows in E. We refer the reader to (Heath, 1982) for details. The crucial assumption made in the method described in (Heath, 1982) is that A has full column rank. However, practical problems might not satisfy this assumption. If the matrix A is rank-deficient, our proposed method described in this paper can be applied to obtain a solution to the problem $\min_x \|Ax - b\|_2$. Handling dense rows in sparse linear least-squares problems is also considered in (Sun, 1995).

Concluding Remarks

In this paper, we have proposed a new method for handling rank deficiency in solving sparse linear least-squares problems. Our proposed method is mathematically equivalent to an existing method. We have shown that our method has several practical advantages over the existing method in terms of efficiency and applicability to rank-deficient sparse problems. Our experimental results show the practical promise of our approach. We have outlined several future directions of research to further enhance the efficiency of our algorithm and to apply our algorithm to the solution of sparse linear least-squares problems with dense rows.

Works Cited

Avron H., Ng E. and Toledo S. Using Perturbed QR Factorizations to Solve Linear Least-Squares Problems [Article] // SIAM J. Matrix Anal. Appl.. - 2009. - 2 : Vol. 31. - pp. 674-693.

Coleman T. F. and Sun C. Solving Rank-Deficient Sparse Linear Least Squares Problems [Report] / University of Waterloo. - Waterloo, Ontario, Canada : [s.n.], 2010. - Work in Progress.

George J. A. and Heath M. T. Solution of Sparse Linear Least Squares Problems using Givens Rotations [Article] // Linear Algebra Appl.. - 1980. - Vol. 34. - pp. 69-83.

Golub G. Numerical Methods for Solving Linear Least Squares Problems [Article] // Numerische Mathematik. - 1965. - Vol. 7. - pp. 206-216.

Heath M. T. Numerical Methods for Large Sparse Linear Least Squares Problems [Article] // SIAM J. Sci. Stat. Comput.. - 1984. - Vol. 26. - pp. 497-513.

Heath M. T. Some Extensions of an Algorithm for Sparse Linear Least Squares Problems [Article] // SIAM J. Sci. Stat. Comput.. - 1982. - Vol. 3. - pp. 223-237.

Liu J. W. H. On General Row Merging Schemes for Sparse Givens Transformations [Article] // SIAM J. Sci. Statist. Comput.. - 1986. - Vol. 7. - pp. 127-148.

Ng E. A scheme for handling rank deficiency in the solution of sparse linear least squares problems [Article] // SIAM J. Sci. Statist. Comput.. - 1991. - Vol. 12. - pp. 1173-1183.

Riley J. D. Solving Systems of Linear Equations with a Positive Definite, Symmetric, but Possibly Ill-Conditioned Matrix [Article] // Math. Tables Aids Comput.. - 1956. - Vol. 9. - pp. 96-101.

Sun C. Dealing with Dense Rows in the Solution of Sparse Linear Least Squares Problems [Report] / Advanced Computing Research Institute, Cornell Theory Center ; Cornell University. - Ithaca, NY : [s.n.], 1995. - CTC95TR227.

Sun C. Parallel Solution of Sparse Linear Least Squares Problems on Distributed-Memory Multiprocessors [Article] // Parallel Computing. - 1997. - pp. 2075-2093.

Sun C. Parallel Sparse Orthogonal Factorization on Distributed-Memory Multiprocessors [Article] // SIAM J. Sci. Comput.. - 1996. - Vol. 17. - pp. 666-685.