Electronic Transactions on Numerical Analysis.
Volume 8, 1999, pp. 26-35.

ISSN 1068-9613.

ETNA
Kent State University
etna@mcs.kent.edu

PRECONDITIONERS FOR LEAST SQUARES PROBLEMS BY LU FACTORIZATION*

A. BJÖRCK† AND J. Y. YUAN‡

Abstract. Iterative methods are often suitable for solving least-squares problems $\min \|Ax - b\|_2$, where $A \in \mathbf{R}^{m\times n}$ is large and sparse. The use of the conjugate gradient method with a nonsingular square submatrix $A_1 \in \mathbf{R}^{n\times n}$ of A as preconditioner was first suggested by Läuchli in 1961. This conjugate gradient method has recently been extended by Yuan to generalized least-squares problems.

In this paper we consider the problem of finding a suitable submatrix A_1 and its LU factorization for a sparse rectangular matrix A. We give three algorithms based on the sparse LU factorization algorithm by Gilbert and Peierls.

Numerical results are given, which indicate that our preconditioners can be effective.

Key words. Linear least squares, preconditioner, conjugate gradient method, LU factorization.

AMS subject classifications. 65F10, 65F20.

1. Introduction. Consider the linear least-squares problem

$$\min_{x\in\mathbf{R}^n} \|Ax - b\|_2, \quad (A \in \mathbf{R}^{m\times n},\ \ b \in \mathbf{R}^m,\ \ m \geq n) \tag{1.1}$$

where $\|\cdot\|_2$ denotes the Euclidean vector norm. For large scale sparse problems iterative solution methods are often preferable to direct methods. To improve the rate of convergence of an iterative method it can be applied to the preconditioned problem

$$\min_{y\in\mathbf{R}^n} \|AS^{-1}y - b\|_2, \quad y = Sx, \tag{1.2}$$

where the nonsingular matrix $S \in \mathbf{R}^{n\times n}$ is a preconditioner.

Here we will consider a class of preconditioners based on a partitioning of A,

$$PA = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}, \tag{1.3}$$

where P is a permutation matrix and $A_1 \in \mathbf{R}^{n\times n}$ a nonsingular submatrix. Such preconditioners first seem to have been suggested by Läuchli [11], and later investigated by Chen [4]. For applications see also [6, 7, 8, 12, 13, 15], and the survey in [2]. The detailed construction of preconditioners of this form for sparse problems seems to have been little studied in the literature.

A powerful class of iterative methods is obtained by applying the conjugate gradient method to the normal equations of the preconditioned problem (1.2),

$$S^{-T}A^TAS^{-1}y = S^{-T}A^Tb. \tag{1.4}$$

Note that for numerical stability care is needed in the implementation of this method. The stability in finite precision of the conjugate gradient and Lanczos methods for least squares problems is discussed in [3].

* Received November 30, 1997. Accepted for publication December 11, 1998. Recommended by D. Calvetti December 11, 1998. The work of the second author was partially supported by CNPq, Brazil.
† Department of Mathematics, Linköping University, S-581 83, Linköping, Sweden (`akbjo@math.liu.se`)
‡ Department of Mathematics, Universidade Federal do Paraná, Curitiba, Paraná, Brazil (`jin@gauss.mat.ufpr.br`)

Yuan [18] recently presented block iterative methods, which use A_1 as preconditioner, for the generalized least-squares problem

$$\min_{x\in\mathbf{R}^n} (Ax-b)^T W^{-1}(Ax-b), \tag{1.5}$$

where W is symmetric and positive definite. The solution to this problem satisfies the normal equations $A^TW^{-1}Ax = A^TW^{-1}b$. Introducing the scaled residual vector $r = W^{-1}(b-Ax)$ these can be written in augmented form as

$$\left\{\begin{array}{ll} Wr + Ax & = b \\ A^T r & = 0 \end{array}\right. . \tag{1.6}$$

An outline of the paper is as follows. In Section 2 we give two basic conjugate gradient methods using A_1 in (1.3) as preconditioner, and give bounds for the rate of convergence. We also show how one of these can be adapted to solve the generalized least squares problem (1.5). An algorithm for selecting n linearly independent rows from A to form A_1, and performing a sparse LU factorization of A_1 is outlined in Section 3. In order to save multiplication and storage, we use a modification of the algorithm of Gilbert and Peierls [9]. In order to ensure the numerical stability of this algorithm, we consider using partial pivoting.

In Section 4 the LU factorization algorithms are tested on some sparse rectangular matrices from the Harwell-Boeing sparse matrix collection [5]. Numerical comparisons between the conjugate gradient method and the preconditioned conjugate gradient method with the preconditioner A_1 given by our algorithms are also given for the generalized least squares problem (5) where A and W are Hilbert matrices. From the comparison results, the preconditioned conjugate gradient method is much better than the conjugate gradient method.

2. CG methods preconditioned by LU factorizations. Iterative methods using A_1 in (1.3) as preconditioner often have very good convergence properties. However, the row permutation matrix P must be chosen so that the n rows of A_1 are linearly independent and AA_1^{-1} is well conditioned.

In this section we assume for simplicity that the permutation P in (1.3) has been carried out in advance, and that we have computed the matrix $C = A_2A_1^{-1} \in \mathbf{R}^{(m-n)\times n}$, where

$$AA_1^{-1} = \begin{pmatrix} I_n \\ A_2A_1^{-1} \end{pmatrix} = \begin{pmatrix} I \\ C \end{pmatrix}. \tag{2.1}$$

Then the preconditioned problem can be written in the form

$$\min_y \left\| \begin{pmatrix} I_n \\ C \end{pmatrix} y - \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \right\|, \quad y = A_1x, \quad C = A_2A_1^{-1}, \tag{2.2}$$

where b has been partitioned conformingly with A. The normal equations for this problem are

$$(I_n + C^TC)y = b_1 + C_2^Tb_2, \tag{2.3}$$

and x can then be retrieved from $A_1x = y$. Läuchli [11] proposed to apply the conjugate gradient algorithm to this system of equations. This leads to the following algorithm:

LU PRECONDITIONED CGLSI. Set

$$r_1^{(0)} = b_1, \quad r_2^{(0)} = b_2, \quad p^{(0)} = s^{(0)} = b_1 + C^T b_2, \quad \gamma_0 = \|s^{(0)}\|_2^2,$$

and for $k = 0, 1, 2, \ldots$ while $\gamma_k > tol$ compute

$$\begin{aligned}
&q^{(k)} = Cp^{(k)},\\
&\alpha_k = \gamma_k / (\|p^{(k)}\|_2^2 + \|q^{(k)}\|_2^2),\\
&r_1^{(k+1)} = r_1^{(k)} - \alpha_k p^{(k)},\\
&r_2^{(k+1)} = r_2^{(k)} - \alpha_k q^{(k)},\\
&s^{(k+1)} = r_1^{(k+1)} + C^T r_2^{(k+1)},\\
&\gamma_{k+1} = \|s^{(k+1)}\|_2^2,\\
&\beta_k = \gamma_{k+1}/\gamma_k,\\
&p^{(k+1)} = s^{(k+1)} + \beta_k p^{(k)}.
\end{aligned}$$

Solve x from $A_1 x = b_1 - r_1$.

It is well known (see Björck [2, Chap. 7.4]) that for the conjugate gradient method applied to the preconditioned normal equations (1.4) the error is reduced according to

$$\|A(x - x_k)\|_2 \leq 2\left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^k \|A(x - x_0)\|_2$$

where x_0 is the initial solution and $\kappa = \kappa(S^{-T} A^T A S^{-1})$ the condition number of the preconditioned normal matrix.

The convergence of the conjugate gradient method applied to (2.3) has been studied by Freund [8]. The eigenvalues of $(I_n + C^T C)$ are

$$\lambda_i = 1 + \sigma_i^2(C), \qquad i = 1, \ldots, n, \tag{2.4}$$

where $\sigma_i(C)$ are the singular values of C. It follows that

$$\kappa(AA_1^{-1}) = \frac{\sqrt{1 + \sigma_1^2(C)}}{\sqrt{1 + \sigma_n^2(C)}} \leq \sqrt{1 + \alpha^2},$$

where

$$\alpha = \sigma_1(C) = \|C\|_2 = \|A_2 A_1^{-1}\|_2 \leq \frac{\sigma_{\max}(A_2)}{\sigma_{\min}(A_1)}. \tag{2.5}$$

Hence, the standard error bounds based on Chebyshev polynomials for this CG method is (see Freund [8])

$$\frac{\|A(x - x_k)\|_2}{\|A(x - x_0)\|_2} < 2\frac{\rho^{2k}}{1 + \rho^{4k}} \qquad \rho = \frac{\alpha}{1 + \sqrt{1 + \alpha^2}}. \tag{2.6}$$

To get fast convergence we want to have α small. According to (2.5) this is achieved if the blocking (1.3) can be chosen so that $\|A_2\|_2$ is small and A_1 well-conditioned. Since C has at most $p = \min\{m - n, n\}$ distinct singular values, the matrix $(I + C^T C)$ will have at most $\min\{p + 1, n\}$ distinct eigenvalues. Hence, in exact arithmetic, CGLSI will converge

ETNA
Kent State University
etna@mcs.kent.edu

in at most $\min\{p+1, n\}$ steps. Therefore, in particular, we can expect rapid convergence if $p \ll n$.

An alternative is obtained by eliminating r_1 from the relations

$$r_1 = b_1 - y = -C^T r_2, \qquad r_2 = b_2 - Cy.$$

Then we get a system of dimension $(m-n) \times (m-n)$ for r_2

$$(CC^T + I_{m-n}) r_2 = b_2 - Cb_1, \tag{2.7}$$

and x can be retrieved from

$$A_1 x = b_1 + C^T r_2. \tag{2.8}$$

The system (2.7) can be interpreted as the normal equations for

$$\min_{r_2} \left\| \begin{pmatrix} -C^T \\ I_{m-n} \end{pmatrix} r_2 - \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \right\|. \tag{2.9}$$

If $(m-n)$ is small the system (2.7) can be cheaply solved by Cholesky factorization, A QR decomposition of the matrix $(-C \ \ I_{m-n})^T$ of size $n \times (m-n)$ also can be used to solve for r_2. Note that the special form of this matrix allows for further economizing in the QR decomposition, see [2, Sec. 2.7.2].

The conjugate gradient method can be adapted to solve the system (2.7). The resulting algorithm is as follows:

LU PRECONDITIONED CGLSII. Set

$$v_1^{(0)} = b_1, \quad v_2^{(0)} = b_2, \quad p^{(0)} = s^{(0)} = b_2 - Cb_1, \quad \gamma_0 = \|s^{(0)}\|_2^2,$$

and for $k = 0, 1, 2, \ldots$ while $\gamma_k > tol$ compute

$$\begin{aligned}
&q^{(k)} = -C^T p^{(k)}, \\
&\alpha_k = \gamma_k / (\|p^{(k)}\|_2^2 + \|q^{(k)}\|_2^2), \\
&v_1^{(k+1)} = v_1^{(k)} - \alpha_k q^{(k)}, \\
&v_2^{(k+1)} = v_2^{(k)} - \alpha_k p^{(k)}, \\
&s^{(k+1)} = v_2^{(k+1)} - C v_1^{(k+1)}, \\
&\gamma_{k+1} = \|s^{(k+1)}\|_2^2, \\
&\beta_k = \gamma_{k+1} / \gamma_k, \\
&p^{(k+1)} = s^{(k+1)} + \beta_k p^{(k)}.
\end{aligned}$$

Solve $A_1 x = b_1 - C^T(b_2 - v_2)$ for x.

Algorithm CGLSII requires about the same storage and work as CGLSI. Since the eigenvalues of $(I + CC^T)$ are the same as those of $(I + C^T C)$ the convergence rate will also be the same. Yuan [18] has shown that an advantage of CGLSII is that it can more easily be adapted to solve generalized least squares problems. For the generalized problem (1.5) the normal equations (2.3) become

$$(\, I_n \quad C^T \,)\, W^{-1} \begin{pmatrix} I_n \\ -C \end{pmatrix} y = (\, I_n \quad C^T \,)\, W^{-1} b, \quad A_1 x = y. \tag{2.10}$$

On the other hand the equations (2.7) and (2.8) generalize to

$$(2.11)\qquad \begin{pmatrix} -C & I_{m-n} \end{pmatrix} W \begin{pmatrix} -C^T \\ I_{m-n} \end{pmatrix} r_2 = b_2 - Cb_1,$$

$$(2.12)\qquad A_1 x = b_1 - \begin{pmatrix} I_n & 0 \end{pmatrix} W \begin{pmatrix} -C^T \\ I_{m-n} \end{pmatrix} r_2.$$

Here the symmetric system (2.11) can be solved by the conjugate gradient method without the need of inverting or factorizing the full covariance matrix W. Yuan shows in [18] that the standard error bounds based on Chebyshev polynomials for this CG method is

$$(2.13)\qquad \frac{\|A(x-x_k)\|_{W^{-1}}}{\|A(x-x_0)\|_{W^{-1}}} < 2\frac{\rho^{2k}}{1+\rho^{4k}}, \qquad \rho^2 = \frac{(1+\alpha^2)\beta - 1}{(1+\sqrt{(1+\alpha^2)\beta})^2}.$$

where $\beta = \kappa(W)$ is the spectral condition number of W, and

$$\|x\|_{W^{-1}} = (x^T W^{-1} x)^{1/2}.$$

(For $\beta = 1$ this estimate reduces to (2.6).)

THEOREM 2.1. *Let λ_n and γ_n be smallest eigenvalues of A^TA and $A_1^TA_1$ respectively. Then*

$$\kappa(I + C^TC) \le \kappa(A^TA)\frac{\lambda_n}{\gamma_n}.$$

Proof. Let λ_1 be the largest eigenvalue of A^TA, A_1x_1 eigenvector of $I+C^TC$ associated with σ_1, the largest eigenvalue of $I+C^TC$, i.e., $\lambda_1 = \max_{\|x\|_2=1} x^TA^TAx$ and $\sigma_1 = \max_{\|x\|_2=1} x^T(I+C^TC)x$. Then,

$$\lambda_1 \ge x_1^TA_1^T(I+C^TC)A_1x_1 = \sigma_1 x_1^TA_1^TA_1x_1 \ge \sigma_1\gamma_n.$$

Assume that $\sigma_n = \min_{\|x\|_2=1} x^T(I+C^TC)x$ is the smallest eigenvalue of $I+C^TC$. Hence, $\sigma_n \ge 1$. It follows that

$$\kappa(I+C^TC) \le \sigma_1 \le \frac{\lambda_1}{\gamma_n} = \kappa(A^TA)\frac{\lambda_n}{\gamma_n}.$$

□

REMARK 1. *If x_1 is not eigenvalue of A^TA associated with λ_1, then*

$$\kappa(I+C^TC) < \kappa(A^TA)\frac{\lambda_n}{\gamma_n}.$$

Thus we must choose the preconditioner A_1 as well as possible such that $\gamma_n \approx \lambda_n$ to guarantee $\kappa(I+C^TC) \le \kappa(A^TA)$.

3. LU-decomposition algorithms. Läuchli [11] used Gauss-Jordan elimination to explicitly compute the matrix $C = A_2A_1^{-1}$ in (2.1). Then at each iteration step in the preconditioned conjugate gradient methods matrix vector products with C and C^T are computed. This approach is suitable only when $(m-n)$ is not too large compared to n.

When A has many more rows than columns it is preferable to factorize just A_1 rather than A and not form the matrix $C = A_2A_1^{-1}$ explicitly. If C is not formed, we need to perform in each iterative step two matrix vector multiplications with A_2 and A_2^T, and solve two linear systems of the form

$$A_1 y = c \text{ and } A_1^T z = d. \tag{3.1}$$

To do this efficiently we need to compute the LU factorization of A_1. We could compute the LU factorization of the complete matrix A and then extract the LU factorization of A. However, this would require extra computation and storage. Further we would need an auxiliary array to save a copy of A_2.

We will use the sparse LU-decomposition algorithm of Gilbert and Peierls [9], modified so that we get a LU decomposition of A_1 only. This algorithm performs Gaussian elimination in column-wise order, and breaks the computation of each column into a symbolic and a numerical stage. The jth column of L and U are computed by solving a sparse triangular system. This approach allows partial pivoting by rows to be performed in time proportional to arithmetic operations.

The algorithm of Gilbert and Peierls can be modified to work for an $n \times m$ matrix with $m \geq n$. For our least squares problems the matrix A is $m \times n$, with $m \geq n$. Therefore we apply Gilbert and Peierls' algorithm to the transposed matrix A^T to obtain a factorization $A_1 = U^T L^T$ of an $n \times n$ nonsingular submatrix. To reduce fill-in we first sort the rows of A by increasing nonzero count.

We use similar notations as Gilbert and Peierls in [3] for description of the basic algorithm. Thus j is the index of the row of L and U being computed. $a_j = (a_{j1}, \ldots, a_{j,j-1})^T$, $a'_j = (a_{jj}, \ldots, a_{jn})^T$,

$$L_j = \begin{pmatrix} l_{11} & \ldots & 0 \\ \vdots & \ddots & \vdots \\ l_{j-1,1} & \ldots & l_{j-1,j-1} \end{pmatrix} \quad L'_j = \begin{pmatrix} l_{j1} & \ldots & l_{j-1,j} \\ \vdots & \ddots & \vdots \\ l_{1,n} & \ldots & l_{j-1,n} \end{pmatrix},$$

and $l'_j = (l_{jj}, \ldots, l_{nj})^T$, $u_j = (u_{1j}, \ldots, u_{j-1,j})^T$ where $l_{jj} = 1$. Also we use $c'_j = (c_{jj}, \ldots, c_{nj})^T$ as an intermediate result.

In the basic algorithm SP1, if $|u_{ii}| < \epsilon$, where ϵ is small tolerance, we consider the ith column in A^T as linearly dependent on the first $i-1$ columns. Such a column is put out of all further computations in the factorization, and interchanged with a later column. We repeat this process until $i = n$ or more than $m - n$ linearly dependent columns have been found. The output of Algorithm SP1 is $A_1 = U^T L^T$ where L is unit lower triangular.

ALGORITHM SP1

1. *Set initial values* $l = m$, $index(i) = i$, $i = 1, \ldots, m$;
2. *For* $j = 1$, *to* n *do*

   Compute column j *of* U *and* L;

   2.1 solve $L_j u_j = a_j$ *for* u_j; (a_j^T *is the* j*th row in* A)

   2.2 $c_j = a'_j - L'_j u_j$;

   2.3 if $|c_{jj}| \geq \epsilon$ *then go to 2.5*

   else

   interchange the j*th column and the* l*th column of* A^T,

   and update index and l;

   2.4 if $l \leq j$ *then* $rank(A) < n$ *and stop else goto 2.1*

   2.5 $u_{jj} = c_{jj}$; $\quad l'_j = c_j / u_{jj}$.

The upper triangular solve in step 2.1 proceeds as follows (see [9]: for each k with $u_{kj} \neq 0$ (in the topological order determined in the previous step) do

$$u_j = u_j - u_{kj} \, (\, l_{1k} \quad \ldots \quad l_{j-1,k} \,)^T \, .$$

Algorithm SP1 computes elements in the LU-decomposition of A_1 in a column-wise fashion. Now we shall consider the LU decomposition of A_1 as $U^T LT$ where U is unit upper triangular.

ALGORITHM SP2

1. *Set initial values* $l = m$, $index(i) = i$, $i = 1, \ldots, m$;
2. *For* $j = 1, \ldots, n$ *do*

   2.1 solve $U_j l_j = a_j$ *for* l_j;

   2.2 $c_j = a'_j - U'_j l_j$;

   2.3 if $|c_{jj}| \geq \epsilon$ *then go to 2.5*

   else

   interchange the j*th row and the* l*th row of* A,

   and update index and l;

   2.4 if $l \leq j$ *then* $rank(A) < n$ *and stop else goto 2.1*

   2.5 $l_{jj} = c_{jj}$; $\quad u'_j = c_j / l_{jj}$.

   2.6 $l = m$

Algorithm SP3 performs partial pivoting, and is obtained by making the following changes in Algorithm SP2:

ALGORITHM SP3

Before step 2.3 insert

2.3a $k = arg\max_{j \leq i \leq n} |c_{ji}|$ *and* $s1 = c_{jk}$;

Change 2.6 to:

2.6 if $k = j$ *goto 2.7 else do*

exchange c_{jj}, $index1(j)$ *and* $U'_{ji}, (i = 1, \ldots, j-1)$
respectively with c_{jk}, $index1(k)$ *and* $U'_{jk}, (i = 1, \ldots, j-1)$.

2.7 $l_{jj} = c_{jj}$

$u'_j = c_j / l_{jj}$.

We finally mention an alternative approach suggested by Saunders [16] is to use Gaussian elimination with row interchanges to compute a stable, sparse factorization $PA = LU$, where $L \in \mathbf{R}^{m \times n}$ unit lower trapezoidal, U upper triangular, and use U as preconditioner. The rational for this choice is that any ill-conditioning in A is usually reflected in U, and L tends to be well conditioned. A preliminary pass through the rows of A can be made to select a triangular subset with maximal diagonal elements. The matrix L is not saved, and subsequent use of the operator AU^{-1} involves back-substitution with U and multiplication with A. This approach has the advantage that often there is very little fill in U, and hence U is likely to contain less non-zeros than A.

4. Numerical experiments. All programs were written in FORTRAN 77 and executed in double precision. The computations were performed on a SUN Workstation running UNIX at Information Laboratory, the Federal University of Paraná, Curitiba, Brazil.

The three LU factorization algorithms in Section 3 were tested on some sparse rectangular matrices from the Harwell-Boeing sparse matrix collection by Duff, Grimes and Lewis

ETNA
Kent State University
etna@mcs.kent.edu

[5]. The numerical results are given in Tables 1–3 for the case $\epsilon = 0$. The CPU time is the time in seconds for the factorization and m and n are the numbers of rows and columns of sparse matrix A. NZ is the number of nonzero elements in matrix A and $NZ2$ the number of nonzero elements in the submatrix A_2. NZL and NZU are the numbers of nonzero elements of the triangular factors L and U in the LU decomposition of A_1.

We also did numerical comparisons between the conjugate gradient method and the preconditioned conjugate gradient method with the preconditioner A_1 selected by algorithms in the previous section for the generalized least squares problems (GLSP) (5) where A and W are Hilbert matrices and b generated by random number, with convergence tolerance $\epsilon = 0.0000001$. The comparison results were given in Table 4.

Algorithm SP1 gives almost the same number of nonzero elements in the LU decomposition as SP2, but needs much more CPU time. Algorithms SP2 and SP3 are faster than SP1 by a factor of 5–10, but have more fill-in. We have also considered $\epsilon \neq 0$, e.g., $\epsilon = 10^{-10}$ and 10^{-8}. All three algorithms worked for the matrices tested, and gave an LU decomposition of a nonsingular submatrix A_1. All results have shown that for all algorithms the bigger ϵ we give, the less CPU time is used and the more nonzero elements in the LU decomposition of A_1 result.

It follows from Table 4 that the preconditioned conjugate gradient method with selected preconditioner A_1 is much better than the conjugate gradient method in the number of iterations, CPU time and also precision of solutions.

REFERENCES

[1] J. L. Barlow, N. K. Nichols and R. J. Plemmons, *Iterative methods for equality-constrained least squares problems*, SIAM J. Sci. Statist. Comput., 9 (1988), 892–906.

[2] A. Björck, *Numerical Methods for Least Squares Problems*, in Frontiers in Applied Mathematics, SIAM, 1996.

[3] A. Björck, T. Elfving and Z. Strakoš, *Stability of conjugate gradient and Lanczos methods for linear least squares problems*, SIAM. J. Matrix Anal. Appl., 19 (1998), 720–736.

[4] Y. T. Chen, *Iterative methods for linear least squares problems*, Technical Report CS-75-04, University of Waterloo, Canada, 1975.

[5] I. S. Duff, R. G. Grimes and J. G. Lewis, *User's guide for Harwell-Boeing sparse matrix test problems collection*, Technical Report RAL-92-086, Rutherford Appleton Laboratory, 1992.

[6] D. J. Evans and C. Li, *Numerical aspects of the generalized cg-method applied to least squares problems*, Computing, 41 (1989), 171–178.

[7] ———, *The theoretical aspects of the gcg-method applied to least-squares problems*, Inter. J. Comput. Math., 35 (1990), 207–229.

[8] R. Freund, *A note on two block SOR methods for sparse least squares problems*, Linear Algebra Appl., 88/89 (1987), 211–221.

[9] J. R. Gilbert and T. Peierls, *Sparse partial pivoting in time proportional to arithmetic operations*, SIAM J. Sci. Statist. Comput., 9 (1988), 862–874.

[10] A. Jennings and M. A. Ajiz, *Incomplete methods for solving $A^TAx = b$*, SIAM J. Sci. Statist. Comput., 5 (1984), 978–987.

[11] P. Läuchli, *Jordan-Elimination und Ausgleichung nach kleinsten Quadraten*, Numer. Math., 3 (1961), 226–240.

[12] T. L. Markham, M. Neumann and R. J. Plemmons, *Convergence of a direct-iterative method for large-scale least-squares problems*, Linear Algebra Appl., 69 (1985), 155–167.

[13] W. Niethammer, J. de Pillis and R. S. Varga, *Convergence of block iterative methods applied to sparse least-squares problems*, Linear Algebra Appl., 58 (1984), 327–341.

[14] C. C. Paige, *Fast numerically stable computations for generalized least squares problems*, SIAM J. Numer. Anal., 16 (1979), 165–171.

[15] E. P. Papadopoulou, Y. G. Saridakis and T. S. Papatheodorou, *Block AOR iterative schemes for large-scale least-squares problems*, SIAM J. Numer. Anal., 26 (1989), 637–660.

[16] M. A. Saunders, *Sparse least squares by conjugate gradients: a comparison of preconditioning methods*, in Proceedings of Computer Science and Statistics: Twelfth Annual Conference on the Interface, Waterloo, Canada, 1979.

[17] J.-Y. YUAN, *The convergence of the 2-block SAOR method for the least-squares problem*, Appl. Numer. Math., 11 (1993), 429–441.
[18] ———, *Iterative Methods for the Generalized Least-Squares Problem*, Ph.D. thesis, Instituto de Matemática Pura e Aplicada, Rio de Janeiro, Brazil, 1993.
[19] ———, *Numerical methods for generalized least-squares problems*, J. Comp. Appl. Math., 66 (1996), 571–584.
[20] J.-Y. YUAN AND A.-N. IUSEM, *Preconditioned conjugate gradient method for generalized least squares problems*, J. Comp. Appl. Math., 71 (1996), 287–297.

TABLE 4.1
Algorithm SP1 with $\epsilon = 0$

Matrix	m	n	NA	NA2	NAL	NAU	CPU
ILLC1033	1033	320	4732	3254	1100	1774	8.96875
WELL1033	1033	320	4732	3257	1243	1752	8.17578
ILLC1850	1850	712	8758	5378	3603	4927	144.891
WELL1850	1850	712	8758	5388	4152	5301	153.930

TABLE 4.2
Algorithm SP2 with $\epsilon = 0$

Matrix	m	n	NA	NA2	NAL	NAU	CPU
ILLC1033	1033	320	4732	3214	1623	654	1.82031
WELL1033	1033	320	4732	3214	1802	844	1.02344
ILLC1850	1850	712	8758	5377	8652	2834	25.9297
WELL1850	1850	712	8758	5377	9073	2627	11.6280

TABLE 4.3
Algorithm SP3 with $\epsilon = 0$

Matrix	m	n	NA	NA2	NAL	NAU	CPU
ILLC1033	1033	320	4732	3253	2865	2450	1.92188
WELL1033	1033	320	4732	3256	3100	2976	1.80078
ILLC1850	1850	712	8758	5379	13054	12579	11.4609
WELL1850	1850	712	8758	5388	13058	12581	12.7108

TABLE 4.4
Comparison between CG and PCG for GLSP

		CG			PCG		
m	n	IT	CPU	e	IT	CPU	e
9	8	21	0.18662	0.2794D-08	1	0.0055	0.1434D-07
9	7	14	0.19877	0.6519D-08	4	0.0055	0.5560D-07
9	6	10	0.22156	0.6519D-08	7	0.0396	0.3817D-07
9	5	7	0.15960	0.3725D-08	1	0.0055	0.1368D-06
8	8	20	0.21428	0.1397D-07	0	0.0055	0.1760D-08
8	7	14	0.22700	0.2352D-07	1	0.0055	0.1499D-07
8	6	11	0.14108	0.6985D-09	4	0.0391	0.7903D-08
8	5	7	0.21004	0.9313D-08	6	0.0395	0.7047D-08