A BIDAGONALIZATION ALGORITHM FOR SPARSE LINEAR EQUATIONS AND L-ETC

OCT 78 C C PAIGE, M A SAUNDERS

UNCLASSIFIED

SOL-78-19
A BIDIODNALIZATION ALGORITHM FOR
SPARSE LINEAR EQUATIONS AND LEAST-SQUARES PROBLEMS.

by

Christopher C. Paige* and Michael A. Saunders

TECHNICAL REPORT SOL-78-19

Oct. 1978

NFS-MCS-76-2067

School of Computer Science, McGill University, Montreal, Canada.

*Applied Mathematics Division, Department of Scientific and Industrial Research, Wellington, New Zealand.

Research and reproduction of this report were partially supported by Office of Naval Research Contract N00014-75-C-0267; the National Science Foundation Grants MCS76-20019 and ENG77-06761; and the Department of Energy Contract EY-76-S-03-0326 PA #18.

Reproduction in whole or in part is permitted for any purposes of the United States Government. This document has been approved for public release and sale; its distribution is unlimited.
CONTENTS

1. INTRODUCTION ............................................. 2
   1.1 Notation ............................................... 3

2. MOTIVATION VIA THE LANCZOS PROCESS .................. 4

3. THE BIDIAgonALIZATION PROCEDURES ..................... 8
   3.1 Relationship between the bidiagonalizations .......... 10

4. ALGORITHM LSQR ........................................... 12
   4.1 Recurrence relations .................................. 14

5. ESTIMATION OF NORMS ..................................... 16
   5.1 Estimates of \|r_k\| and \|A^T r_k\| ....................... 16
   5.2 An estimate of \|x_k\| .................................. 17
   5.3 Estimation of \|A\|_F and cond(A) ................. 18
   5.4 Standard errors ....................................... 19

6. STOPPING CRITERIA ........................................ 22
   6.1 Compatible systems .................................. 23
   6.2 Incompatible systems ................................ 23
   6.3 Ill-conditioned systems ............................... 24
   6.4 Singular systems ..................................... 25

7. OTHER METHODS ............................................ 27
   7.1 CGLS .................................................. 27
   7.2 Craig's method for \Ax = b ......................... 28
   7.3 Extension of Craig's method ....................... 29
   7.4 LSCG and LSIOQ ..................................... 31
7.5 Chen's algorithm RRLS ... ... ... ... ... 32  
7.6 RRLSQR ... ... ... ... ... ... 33  
7.7 Storage and work ... ... ... ... ... 35  

8. NUMERICAL COMPARISONS ... ... ... ... ... ... 37  
8.1 Generation of test problems ... ... ... ... ... 37  
8.2 $Ax = b$; failure of CGLS ... ... ... ... ... 39  
8.3 $Ax = b$; failure of RRLS ... ... ... ... ... 39  
8.4 $\min \|Ax - b\|$; high accuracy by RRLS and RRLSQR ... 40  
8.5 $\min \|Ax - b\|$; normal behavior ... ... ... ... 42  
8.6 Some results using greater precision ... ... ... ... 42  

9. TRANSFORMATIONS ... ... ... ... ... ... ... ... 47  
9.1 Partitioning by columns ... ... ... ... ... ... 47  
9.2 Partitioning by rows ... ... ... ... ... ... ... ... 50  
9.3 Scaling and preconditioning ... ... ... ... ... ... 51  
9.4 Linear constraints ... ... ... ... ... ... ... ... 53  

10. AN APPLICATION IN GEOPHYSICS ... ... ... ... ... ... 56  
10.1 Original formation ... ... ... ... ... ... ... ... 56  
10.2 Preconditioning with $LU$ factors ... ... ... ... ... 57  
10.3 A problem-dependent transformation ... ... ... ... ... 59  
10.4 Comparison with $QR$ factorization ... ... ... ... ... 63  
10.5 A larger problem ... ... ... ... ... ... ... ... 63  

11. SUMMARY ... ... ... ... ... ... ... ... ... ... 65  

ACKNOWLEDGEMENTS ... ... ... ... ... ... ... ... ... 65  
REFERENCES ... ... ... ... ... ... ... ... ... ... ... 66
Abstract

A method is given for solving $Ax = b$ and $\min \|Ax - b\|_2$ where the matrix $A$ is large and sparse. The method is based on the bidiagonalization procedure of Golub and Kahan. It is analytically equivalent to the method of conjugate gradients (CG) but possesses more favorable numerical properties. The Fortran implementation of the method (subroutine LSQR) incorporates reliable stopping criteria and provides estimates of various quantities including standard errors for $x$ and the condition number of $A$. Numerical tests are described comparing LSQR with several other CG algorithms. Further results for a large practical problem illustrate the effect of pre-conditioning least-squares problems using a sparse LU factorization of $A$. 
1. INTRODUCTION

A numerical method is presented here for computing a solution $x$ to either of the following problems:

Unsymmetric equations: solve $Ax = b$

Linear least squares: minimize $\|Ax - b\|_2$

where $A$ is a real matrix of dimensions $m$ by $n$ and $b$ is a real vector. It will usually be true that $m \geq n$ and $\text{rank}(A) = n$, but these conditions are not essential. The method, to be called algorithm LSQR, is similar in style to the well-known method of conjugate gradients (CG) as applied to the least-squares problem (Hestenes and Stiefel [11]). The matrix $A$ is used only to compute products of the form $Av$ and $A^Tu$ for various vectors $v$ and $u$. Hence $A$ will normally be large and sparse or will be expressible as a product of matrices that are sparse or have special structure. A typical application is to the large least-squares problems arising from analysis of variance.

CG-like methods are iterative in nature. They are characterized by their need for only a few vectors of working storage and by their theoretical convergence within at most $n$ iterations (if exact arithmetic could be performed). In practice such methods may require far fewer or far more than $n$ iterations to reach an acceptable approximation to $x$. The methods are most useful when $A$ is well-conditioned and has many nearly equal singular values. These properties occur naturally in many applications. In other cases it is often possible to divide the solution procedure into a direct and an iterative part, such that the iterative part has a better conditioned matrix for which CG-like methods will converge more quickly. Some such transformation methods are considered here.
Algorithm LSQR is based on the bidiagonalization procedure of Golub and Kahan [10]. It generates a sequence of approximations \( \{x_k\} \) such that the residual norm \( \|r_k\|_2 \) decreases monotonically, where \( r_k = b - Ax_k \).

Analytically the sequence \( \{x_k\} \) is identical to the sequence generated by the standard CG algorithm and by several other published algorithms. However, LSQR is shown by example to be numerically more reliable in various circumstances than the other methods considered.

The Fortran implementation of LSQR is designed for practical application. It incorporates reliable stopping criteria and provides the user with computed estimates of the following quantities: \( x, r = b - Ax, A^T r, \|
\|_2, \|A\|_F, \) standard errors for \( x \), and the condition number of \( A \).

1.1 Notation

Matrices will be denoted by \( A, B, \ldots \), vectors by \( v, w, \ldots \), and scalars by \( \alpha, \beta, \ldots \). An exception is \( c \) and \( s \) which will denote the significant components of an elementary orthogonal matrix, such that \( c^2 + s^2 = 1 \). For a vector \( v \), \( \|v\| \) will always denote the Euclidean norm \( \|v\|_2 = (v^T v)^{1/2} \). For a matrix \( A \), \( \|A\| \) will usually mean the Frobenius norm, \( \|A\|_F = (\sum_{i,j} A_{ij}^2)^{1/2} \), and the condition number for an unsymmetric matrix \( A \) is defined by \( \text{cond}(A) = \|A\| \|A^+\| \) where \( A^+ \) denotes the pseudo-inverse of \( A \).

The relative precision of floating-point arithmetic will be \( \varepsilon \), the smallest machine-representable number such that \( 1 + \varepsilon > 1 \).
2. MOTIVATION VIA THE LANCZOS PROCESS

In this section we review the symmetric Lanczos process [13] and its use in solving symmetric linear equations $Mx = b$. Algorithm LSQR is then derived by applying the Lanczos process to a particular symmetric system. Although a more direct development is given in section 4, the present derivation may remain useful for a future error analysis of LSQR, since many of the rounding error properties of the Lanczos process are already known (Paige [19]).

Given a symmetric matrix $M$ and a starting vector $b$, the Lanczos process is a method for generating a sequence of vectors $\{v_i\}$ and scalars $\{\alpha_i\}, \{\beta_i\}$ such that $M$ is reduced to tridiagonal form. A reliable computational form of the method is as follows:

The Lanczos process (Reduction to tridiagonal form)

\begin{align}
(a) & \quad \beta_1 v_1 = b \\
(b) & \quad \omega_i = Mv_i - \beta_i v_{i-1} \\
& \quad \alpha_i = v_i^T \omega_i \quad i = 1, 2, \ldots \\
& \quad \beta_{i+1} v_{i+1} = \omega_i - \alpha_i v_i \\
\end{align}

where $v_0 \equiv 0$ and each $\beta_i \geq 0$ is chosen so that $\|v_i\| = 1$ ($i > 0$). Prior to termination at the first zero $\beta_{i+1}$, the situation after $k$ steps is summarized by

\begin{equation}
MV_k = V_k T_k + \beta_{k+1} v_{k+1} e_k^T
\end{equation}

where $T_k \equiv \text{tridiag}(\beta_{i}, \alpha_i, \beta_{i+1})$ and $V_k \equiv [v_1, v_2, \ldots, v_k]$. If exact arithmetic is used then $V_k^T V_k = I$ and so $\beta_{i+1}$ must vanish for $i = n$, if not before, but in any event equation (2.2) holds to within machine precision.
Now suppose we wish to solve the symmetric system \( Mx = b \). Multiplying (2.2) by an arbitrary \( k \)-vector \( y_k \) whose last element is \( \eta_k \) gives \( MV_k y_k = V_k^T T_k y_k + \beta_{k+1} v_{k+1}^T \eta_k \). Since \( V_k (\beta_1 e_1) = b \) by definition, it follows that if \( y_k \) and \( x_k \) are defined by the equations

\[
T_k y_k = \beta_1 e_1 \quad (2.3)
\]

\[
x_k = V_k y_k \quad (2.4)
\]

then we shall have \( Mx_k = b + \eta_k \beta_{k+1} v_{k+1} \) to working accuracy. Hence \( x_k \) may be taken as the exact solution to a perturbed system, and will solve the original system whenever \( \eta_k \beta_{k+1} \) is negligibly small.

The above arguments are not complete, but they provide at least some motivation for defining the sequence of vectors \( \{x_k\} \) according to equations (2.3) and (2.4). It is now possible to derive several iterative algorithms for solving \( Mx = b \), each characterized by the manner in which \( y_k \) is eliminated from (2.3) and (2.4) (since it is not practical to compute each \( y_k \) explicitly). In particular, the method of conjugate gradients is known to be equivalent to using the Cholesky factorization \( T_k = L_k D_k L_k^T \) and is reliable when \( M \) (and hence \( T_k \)) is positive definite, while algorithm SYMMLQ employs the orthogonal factorization \( T_k = L_k Q_k \) to retain stability for arbitrary symmetric \( M \). (See Paige and Saunders [21] for further details of these methods.)

We now turn to a particular symmetric (but indefinite) system, namely

\[
\begin{bmatrix}
I & A \\
A^T & \rho
\end{bmatrix}
\begin{bmatrix}
x \\
r
\end{bmatrix}
= \begin{bmatrix}
b \\
o
\end{bmatrix} \tag{2.5}
\]

which is well known to solve the least-squares problem, \( \min \|Ax - b\| \), for arbitrary \( A \) and \( b \). If the Lanczos process is applied to the partitioned
matrix and rhs vector in (2.5), the recurrence relations (2.1) simplify to give one of the bidiagonalization procedures to be discussed in the next section. Furthermore, after $2k+1$ iterations the tridiagonal system corresponding to (2.3) has the form

$$
T_{2k+1} y_{2k+1} = \begin{bmatrix}
1 & \alpha_1 \\
\alpha_1 & 0 & \beta_2 \\
& \beta_2 & 1 & \alpha_2 \\
& & \ddots & \vdots & \ddots \\
& & & \alpha_k & 0 & \beta_{k+1} \\
& & & & \beta_{k+1} & 1 \\
\end{bmatrix} \begin{bmatrix}
\tau_1 \\
\eta_1 \\
\tau_2 \\
\eta_2 \\
\vdots \\
\vdots \\
\eta_k \\
\tau_{k+1} \\
\end{bmatrix} = \beta_1 e_1 \quad (2.6)
$$

following appropriate change in notation.

System (2.6) is indefinite and could be dealt with as in algorithm SYMMLQ, i.e., using the orthogonal factorization $T_{2k+1} = L_{2k+1} Q_{2k+1}$. However, this factorization proves to be essentially the same as for a general tridiagonal matrix without any convenient simplification. Thus a specialized form of SYMMLQ could capitalize on the simplified form of the Lanczos process, but it would not take full advantage of the structure inherent in (2.6).

Instead we observe that a simple symmetric permutation of (2.6) gives an equivalent system of the form

$$
\begin{bmatrix}
I & B_k \\
B_k^T & y_k \\
\end{bmatrix} \begin{bmatrix}
t_{k+1} \\
y_{k} \\
\end{bmatrix} = \begin{bmatrix}
\beta_1 e_1 \\
0 \\
\end{bmatrix} \quad (2.7)
$$
where \( B_k \equiv \text{bidiag}(\beta, \alpha) \) is a lower bidiagonal matrix of order \( k+1 \) by \( k \).

Comparing (2.7) with (2.5) immediately reveals another least-squares problem, \( \min \| B_k y_k - B_1 e_1 \| \). This does have a convenient structure and can be solved reliably using an orthogonal factorization of \( B_k \), as in the least-squares algorithm of Golub [9].

To summarize: an application of the Lanczos process to the problem \( \min \| A x - b \| \) leads to a series of subproblems \( \min \| B_k y_k - B_1 e_1 \| \) which can be effectively dealt with using a conventional QR factorization of \( B_k \). This observation forms the basis for algorithm LSQR.
3. The Bidiagonalization Procedures

In the previous section it was noted that the recurrence relations (2.1) simplify when the Lanczos process is applied to the least-squares system (2.5). (In fact both $V_k$ and $T_k$ have special structure.) The result is one form of the bidiagonalization procedure of Golub and Kahan [10]. For future reference we shall state the procedure in two different forms and give some unexpected relationships between the forms. It will then be possible to derive algorithm LSQR directly and relate it to various other algorithms that have been proposed.

For brevity we shall call the procedures Bidiag 1 and 2. The notation used emphasizes the connections between them.

Bidiag 1 (Starting vector $b$; reduction to lower bidiagonal form)

(a) $\beta_1 u_1 = b$, $\alpha_1 v_1 = A^T u_1$.

(b) $\begin{bmatrix} \beta_{i+1} u_{i+1} = A v_i - \alpha_i u_i \\ \alpha_{i+1} v_{i+1} = A^T u_{i+1} - \beta_{i+1} v_i \end{bmatrix}$ \hspace{1cm} $i = 1, 2, \ldots$  \hspace{1cm} (3.1)

The scalars $\alpha_i \geq 0$ and $\beta_i \geq 0$ are chosen so that $\|u_i\| = \|v_i\| = 1$. With the definitions

$U_k \equiv [u_1, u_2, \ldots, u_k]$, $B_k \equiv \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_k \end{bmatrix}$,

$V_k \equiv [v_1, v_2, \ldots, v_k]$, $B_{k+1} \equiv \begin{bmatrix} \beta_1 & \alpha_2 \\ \beta_2 & \alpha_3 \\ \vdots & \vdots \\ \beta_k & \alpha_{k+1} \end{bmatrix}$,

(where $B_k$ is the rectangular matrix introduced in section 2), the recurrence
relations (3.1) may be rewritten as

\[ U_{k+1}(B_1 \epsilon_1) = b \quad , \]
\[ AV_k = U_{k+1} P_k \quad , \]
\[ A^T U_{k+1} = V_k P_k^T + \sigma_{k+1} V_{k+1} \epsilon_{k+1}^T \quad . \]

If exact arithmetic were used then we would also have \( U_{k+1}^T U_{k+1} = I \) and \( V_k^T V_k = I \), but in any event the above equations hold to within machine precision.

**Bidiag 2 (Starting vector \( A^T b \); reduction to upper bidiagonal form)**

(a) \( \theta_1 v_1 = A^T b \), \( \rho_1 p_1 = A v_1 \).

(b) \[ \begin{align*}
\theta_{i+1} v_{i+1} &= A^T p_i - \rho_i v_i \\
\rho_{i+1} p_{i+1} &= A v_{i+1} - \theta_{i+1} p_i
\end{align*} \] \( i = 1, 2, \ldots \) \( (3.5) \)

Again, \( \rho_i \geq 0 \) and \( \theta_i \geq 0 \) are chosen so that \( ||v_i|| = ||v_i|| = 1 \). In this case, if

\[ P_k \equiv [p_1, p_2, \ldots, p_k] , \quad R_k \equiv \begin{bmatrix}
\rho_1 & \theta_2 \\
\rho_2 & \theta_3 \\
\vdots & \ddots \\
\rho_{k-1} & \theta_k \\
\rho_k
\end{bmatrix} \]

we may rewrite (3.5) as

\[ V_k(\theta_i \epsilon_i) = A^T b \quad , \]
\[ AV_k = P_k R_k \quad . \]
\[ A^T P_k = V_k R_k^T + \theta_{k+1} v_{k+1} e_k^T, \]

(3.8)

and with exact arithmetic we would also have \[ P_k^T P_k = V_k^T V_k = I. \]

Bidiag 2 is the procedure originally given by Golub and Kahan (with the particular starting vector \( A^T b \)). Either procedure may be derived from the other by choosing the appropriate starting vector and interchanging \( A \) and \( A^T \).

### 3.1 Relationship between the bidiagonalizations

The principal connection between the two bidiagonalization procedures is that the matrices \( V_k \) are the same for each, and that the identity

\[ B_k^T B_k = R_k^T R_k \]

holds. This follows from the fact that \( v_1 \) is the same in both cases and \( V_k \) is mathematically the result of applying \( k \) steps of the Lanczos process (2.2) with \( M = A^T A \). The rather surprising conclusion is that \( R_k \) must be identical to the matrix that would be obtained from the conventional QR factorization of \( B_k \). Thus

\[ Q_k P_k = \begin{bmatrix} R_k \\ 0 \end{bmatrix} \]

(3.10)

where \( Q_k \) is orthogonal. In the presence of rounding errors these identities will of course cease to hold. However, they throw light on the advantages of algorithm LSQR over two earlier methods, LSCG and LSLQ, as discussed in section 7.4.

The relationship between the orthonormal matrices \( U_k \) and \( P_k \) can be shown to be

\[ U_{k+1} = [U_k \ U_{k+1}] = \begin{bmatrix} P_k & \frac{r_k}{\| r_k \|} \\ \end{bmatrix} Q_k \]

(3.11)
for some vector $r_k$. We also have the identities

$$
\alpha_1^2 + \beta_2^2 = \rho^2_1, \quad \alpha_1 \beta_1 = \theta_1,
$$

$$
\alpha_i^2 + \beta_{i+1}^2 = \rho_i^2 + \theta_i^2, \quad \alpha_i \beta_i = \rho_{i-1} \theta_i \text{ for } i > 1.
$$
4. ALGORITHM LSQR

The quantities generated from \( A \) and \( b \) by Bidiag 1 will now be used to solve the least-squares problem, \( \min \| b - Ax \| \).

Let the quantities

\[
\begin{align*}
x_k &= V_k y_k \\
r_k &= b - Ax_k \\
t_{k+1} &= B_k e_1 - B_k y_k
\end{align*}
\]

be defined in terms of some vector \( y_k \). It readily follows from (3.2) and (3.3) that the equation

\[
r_k = U_{k+1} t_{k+1}
\]

holds to working accuracy. If the columns of \( U_{k+1} \) were exactly orthogonal we would have \( \| r_k \| = \| t_{k+1} \| \), which immediately suggests choosing \( y_k \) to minimize \( \| t_{k+1} \| \). Hence we are led naturally to the least-squares problem

\[
\min_{y_k} \| B_k e_1 - B_k y_k \|
\]

which forms the basis for LSQR.

Computationally it is advantageous to solve (4.5) using the standard QR factorization of \( B_k \) (Golub [9]), i.e. the same factorization (3.10) that links the two bidiagonalizations. This takes the form

\[
Q_k \begin{bmatrix} B_k & B_k e_1 \end{bmatrix} = \begin{bmatrix} R_k & f_k \end{bmatrix} \begin{bmatrix} \rho_1 & \theta_2 \\ \rho_2 & \theta_3 \\ \vdots \\ \rho_{k-1} & \theta_k \\ \rho_k & \phi_k \\ \phi_{k+1} \\ \phi_k \\ \phi_{k+1} \end{bmatrix}
\]
where $Q_k \equiv Q_{k, k+1} \cdots Q_2, Q_1, 2$ is a product of plane rotations (e.g. Wilkinson [27]) designed to eliminate the subdiagonals $\beta_2, \beta_3, \ldots$ of $B_k$. The vectors $y_k$ and $t_{k+1}$ could then be found from

\[
P_k y_k = f_k, \quad (4.7)
\]
\[
t_{k+1} = Q_k \begin{bmatrix} 0 \\ \phi_{k+1} \end{bmatrix}. \quad (4.8)
\]

However, $y_k$ in (4.7) will normally have no elements in common with $y_{k-1}$. Instead we note that $[R_k \ f_k]$ is the same as $[R_{k-1} \ f_{k-1}]$ with a new row and column added. Hence, one way of combining (4.1) and (4.7) efficiently is according to

\[
x_k = V_k R_k^{-1} f_k \equiv D_k f_k \quad (4.9)
\]

where the columns of $D_k = [d_1 \ d_2 \ \ldots \ d_k]$ can be found successively from the system $R_k^T D_k^T = V_k^T$ by forward substitution. With $d_0 = x_0 = 0$ this gives

\[
d_k = \frac{1}{\rho_k} (v_k - \theta_k d_{k-1}) \quad (4.10)
\]
\[
x_k = x_{k-1} + \phi_k d_k \quad (4.11)
\]

and only the most recent iterates need be saved. The broad outline of algorithm LSQR is now complete.
4.1 Recurrence relations

The QR factorization (4.6) is determined by constructing the k-th plane rotation \( Q_{kk+1} \) to operate on rows \( k \) and \( k+1 \) of the transformed matrix \( [B_k \ 0_{1 \times 1}] \) to annihilate \( \beta_{k+1} \). This gives the following simple recurrence relation:

\[
\begin{bmatrix}
    c_k & s_k \\
    s_k & -c_k
\end{bmatrix}
\begin{bmatrix}
    \rho_k & 0 \\
    B_{k+1} & \alpha_{k+1}
\end{bmatrix}
= 
\begin{bmatrix}
    \rho_k & \phi_{k+1} \\
    0 & \phi_{k+1}
\end{bmatrix}
\]

(4.12)

where \( \rho_1 = \alpha_1, \phi_1 = \beta_1 \), and the scalars \( c_k \) and \( s_k \) are the nontrivial elements of \( Q_{kk+1} \). The quantities \( \rho_k, \phi_k \) are intermediate scalars that are subsequently replaced by \( \rho_k, \phi_k \).

The rotations \( Q_{kk+1} \) are discarded as soon as they have been used in (4.12), since \( Q_k \) itself is not required. We see that negligible work is involved in computing the QR factorization to obtain \( R_k, f_k \) and \( \phi_{k+1} \).

Some of the work in (4.10) can be eliminated by using vectors \( \omega_k = \rho_k d_k \) in place of \( d_k \). The main steps of LSQR can now be summarized as follows. (As usual the scalars \( \alpha_k \geq 0 \) and \( \beta_k \geq 0 \) are chosen to normalize the corresponding vectors; for example, \( \alpha_1 v_1 = A^T u_1 \) implies the computations \( \overline{v}_1 = A^T u_1, \alpha_1 = \| \overline{v}_1 \|, v_1 = (1/\alpha_1) \overline{v}_1 \).)
Algorithm LSQR

1. (Initialize.)
   \[ \beta_1 u_1 = b, \quad \alpha_1 v_1 = A^T u_1, \quad \omega_1 = v_1, \quad x_0 = 0, \]
   \[ \phi_1 = \beta_1, \quad \rho_1 = \alpha_1. \]

2. For \( i = 1, 2, 3, \ldots \) repeat steps 3 - 6.

3. (Continue the bidiagonalization.)
   \[
   \begin{align*}
   (a) & \quad \beta_{i+1} u_{i+1} = A v_i - \alpha_i u_i \\
   (b) & \quad \alpha_{i+1} v_{i+1} = A^T u_{i+1} - \rho_i v_i.
   \end{align*}
   \]

4. (Construct and apply next orthogonal transformation.)
   \[
   \begin{align*}
   (a) & \quad \rho_i = (\bar{\rho}_i^2 + \beta_i^{2})^{1/2} \\
   (b) & \quad \sigma_i = \bar{\rho}_i / \rho_i \\
   (c) & \quad s_i = \beta_i / \rho_i \\
   (d) & \quad \theta_{i+1} = s_i \alpha_i + 1 \\
   (e) & \quad \bar{\rho}_{i+1} = -\sigma_i \alpha_i + 1 \\
   (f) & \quad \phi_i = s_i \phi_i \\
   (g) & \quad \bar{\phi}_{i+1} = s_i \bar{\phi}_i.
   \end{align*}
   \]

5. (Update \( x, \omega \).)
   \[
   \begin{align*}
   (a) & \quad x_i = x_{i-1} + (\phi_i / \rho_i) \omega_i \\
   (b) & \quad \omega_{i+1} = v_{i+1} - (\theta_{i+1} / \rho_i) \omega_i.
   \end{align*}
   \]

6. (Test for convergence.)
   Exit if some stopping criteria (yet to be discussed) have been met.
5. ESTIMATION OF NORMS

Here we show that estimates of the quantities \( ||r_k||, \|A^T r_k\|, \|x_k\|, \|A\| \) and \( \text{cond}(A) \) can be obtained at minimal cost from items already required by LSQR. All five quantities will be used later to formulate stopping rules.

Knowledge of \( \|A\| \) and perhaps \( \text{cond}(A) \) can also provide useful debugging information. For example, a user must define his matrix \( A \) by providing two subroutines to compute products of the form \( A v \) and \( A^T u \). These subroutines will typically use data derived from earlier computations, and may employ rather complex data structures in order to take advantage of the sparsity of \( A \). If the estimates of \( \|A\| \) and/or \( \text{cond}(A) \) prove to be unexpectedly high or low then at least one of the subroutines is likely to be incorrect. As a rule of thumb we recommend that all columns of \( A \) be scaled to have unit length \((\|A e_j\| = 1, j = 1, \ldots, n)\), since this usually removes some unnecessary ill-conditioning from the problem. Under these circumstances, a programming error should be suspected if the estimate of \( \|A\| \) differs by a significant factor from \( n^{\frac{k}{2}} \) (since the particular norm estimated will be \( \|A\|_F \)).

For the purposes of estimating norms we shall often assume that the orthogonality relations \( U_k^T U_k = I \) and \( V_k^T V_k = I \) hold, and that \( \|U_k\|_2 = \|V_k\|_2 = 1 \). In actual computations these are rarely true, but the resulting estimates have proved to be remarkably reliable.

5.1 Estimates of \( \|r_k\| \) and \( \|A^T r_k\| \)

From (4.4) and (4.8) we have

\[
r_k = \bar{r}_{k+1}^T u_{k+1}^T q_k^T e_{k+1} \quad (5.1)
\]

(which explains the use of \( r_k \) in (3.11)) and hence by assuming \( U_{k+1}^T U_{k+1} = I \) we obtain the estimate
where the form of $\phi_{k+1}$ follows from (4.12). LSQR is unusual in not having the residual vector $r_k$ explicitly present, but we see that $\|r_k\|$ is available essentially free. Clearly the product of sines in (5.2) decreases monotonically. It should converge to zero if the system $Ax = b$ is compatible. Otherwise it will converge to a positive finite limit.

For least-squares problems a more important quantity is $A^T r_k$, which would be zero at the final iteration if exact arithmetic were performed. From (5.1), (3.4) and (4.6) we have

$$A^T r_k = \phi_{k+1} (V_k B_k^T + \alpha_{k+1} v_{k+1} e_{k+1}^T) Q_k^T e_{k+1}$$

$$= \phi_{k+1} V_k \left[ R_k^T \ 0 \right] e_{k+1} + \phi_{k+1} \alpha_{k+1} (e_{k+1}^T Q_k^T e_{k+1}) v_{k+1}.$$  

The first term vanishes and it is easily seen that the $(k+1)$th diagonal of $Q_k$ is $-\sigma_k$. Hence we have

$$A^T r_k = - (\phi_{k+1} \alpha_{k+1} \sigma_k) v_{k+1}$$  

and

$$\|A^T r_k\| = \phi_{k+1} \alpha_{k+1} |\sigma_k|$$

to working accuracy. No orthogonality assumptions are needed here.

5.2 An estimate of $\|x_k\|$

The upper bidiagonal matrix $R_k$ may be reduced to lower bidiagonal form by the orthogonal factorization

$$R_k \bar{Q}_k^T = \bar{L}_k$$

where $\bar{Q}_k$ is a suitable product of plane rotations. Defining $\bar{x}_k$ by the system

$$\bar{L}_k \bar{x}_k = f_k$$
it follows that $x_k = (V_k P_k^{-1} f_k = (V_k \tilde{Q}_k^T \tilde{v}_k \equiv \tilde{w}_k^T \tilde{z}_k$. Hence, under the assumption that $V_k^T V_k = I$ we can obtain the estimate

$$\|x_k\| = \|\tilde{z}_k\| \quad (5.7)$$

Note that the leading parts of $L_k$, $Q_k$, $\tilde{w}_k$ and $\tilde{z}_k$ do not change after iteration $k$. Hence we find that estimating $\|x_k\|$ via (5.5)-(5.7) costs only 13 multiplications per iteration, which is negligible for large $n$.

5.3 Estimation of $\|A\|_F$ and $\text{cond}(A)$

It is clear from (3.1) that all the $V_i$ lie in the range of $A^T$ and are therefore orthogonal to the null space of $A$ and $A^T A$. With appropriate orthogonality assumptions we have from (3.3) that

$$B_k^T B_k = V_k^T A^T A V_k$$

and so from the Courant-Fischer minimax theorem the eigenvalues of $B_k^T B_k$ are interlaced by those of $A^T A$ and are bounded above and below by the largest and smallest nonzero eigenvalues of $A^T A$. The same can therefore be said of the singular values of $B_k$ compared with those of $A$. It follows that for the 2- and $F$-norms,

$$\|B_k\| \leq \|A\| \quad (5.8)$$

where equality will be obtained in the 2-norm for some $k \leq \text{rank}(A)$ if $b$ is not orthogonal to the left-hand singular vector of $A$ corresponding to its largest singular value. Equality will only be obtained for the $F$-norm if $b$ contains components of all left-hand singular vectors of $A$ corresponding to nonzero singular values. Nevertheless we will use $\|B_k\|_F$ as a monotonically increasing estimate of the size of $A$.

The foregoing also implies that $B_k^T B_k = R_k^T R_k$ is nonsingular and for the 2- and $F$-norms
\( \|R_k^{-1}\| = \|B_k^+\| \leq \|A^+\| \).  

(5.9)

The remarks on equality are the same, except now "largest singular value" is replaced by "smallest nonzero singular value".

Combining these results with the definition \( D_k = V_k R_k^{-1} \) in (4.9) now gives

\[ 1 \leq \|B_k\| \|D_k\| \leq \|A\| \|A^+\| = \text{cond}(A) \]  

(5.10)

for the 2- and \( F \)-norms. Hence we take \( \|B_k\| \|D_k\| \) as a monotonically increasing estimate of \( \text{cond}(A) \), which starts at the optimistic estimate \( \|B_1\| \|D_1\| = 1 \).

Use of Frobenius norms allows the estimates to be accumulated cheaply, since \( \|B_k\|_F^2 = \|B_{k-1}\|_F^2 + \alpha_k^2 + \beta_{k+1}^2 \) and \( \|D_k\|_F^2 = \|D_{k-1}\|_F^2 + \|d_k\|_F^2 \).

The individual terms in the sum \( \|d_k\|_F^2 = \sum_{i=1}^{n} \delta_{ik}^2 \) can be used further for estimating standard errors, as we show next.

5.4 Standard errors

In regression problems with \( m > n = \text{rank}(A) \) the standard error in the \( i \)-th component of the true solution \( x \) is taken to be \( \sigma_i \) where

\[ \sigma_i^2 = \frac{\|b - Ax\|_2^2}{m - n} \sigma_{ii} \]  

(5.11)

and \( \sigma_{ii} = e_i^T (A^T A)^{-1} e_i \) is the \( i \)-th diagonal element of \( (A^T A)^{-1} \). Now from (3.3) and (3.10) we have \( V_k A^T A V_k = R_k^T R_k \), which with (4.9) gives

\[ D_k^T A^T A D_k = I \].

Assuming that premature termination does not occur, it follows that with exact arithmetic \( D_n^T D_n A^T = (A^T A)^{-1} \), and we can approximate the \( \sigma_{ii} \) by
\[ \sigma_{ii}^{(k)} = e_i^T D_k D_k^T e_i. \] Since \( D_k^T D_k = D_{k-1}^T D_{k-1} + d_k^T d_k \), we have

\[ \sigma_{ii}^{(k)} = \sigma_{ii}^{(k-1)} + \delta_{ik}^2, \quad \sigma_{ii}^{(0)} = 0, \]

and the \( \sigma_{ii}^{(k)} \) are monotonically increasing estimates of the \( \sigma_{ii} \).

In the implementation of LSQR we accumulate \( \sigma_{ii}^{(k)} \) for each \( i \), and upon termination at iteration \( k \) we set \( l = \max(m-n,1) \) and output the square roots of

\[ s_i^2 = \frac{\| h - Ax_k \|^2}{l} \sigma_{ii}^{(k)} \]

as estimates of the \( s_i \) in (5.11). The accuracy of these estimates cannot be guaranteed, especially if termination occurs early for some reason. However, we have obtained one reassuring comparison with the statistical package GLIM (Nelder [17]).

On a moderately ill-conditioned problem of dimensions 171 by 38, \( \text{cond}(A) \approx 10^3 \), relative machine precision \( \approx 10^{-11} \), an accurate solution \( x_k \) was obtained after 69 iterations, and at this stage all \( s_i^{(k)} \) agreed to at least one digit with the \( s_i \) output by GLIM, and many components agreed more closely.

A further comparison was obtained from the 1033 by 434 gravity-meter problem discussed in section 10.3. For this problem a sparse QR factorization was constructed, \( QA = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{bmatrix} \), and the quantities \( \sigma_{ii} \) were computed accurately using \( R^T v_i = e_i \), \( \sigma_{ii}^2 = \| v_i \|^2 \). Again the estimates of \( s_i^{(k)} \) from LSQR proved to be accurate to at least one significant figure, and the larger values were accurate to three or more digits.

Note that \( s_i^2 \) estimates the variance of the \( i \)-th component of \( x \), and that \( s_i^{(k)} \) approximates this variance estimate. In an analogous manner we could approximate certain covariance estimates by accumulating
\[
\sigma_{i,j}^{(k)} = \sigma_{i,j}^{(k-1)} + \delta_{i,k} \delta_{j,k}, \quad \sigma_{i,j}^{(0)} = 0,
\]
for any specific pairs \((i,j)\), and then computing
\[
\frac{||b - Ax_k||^2}{\sigma_{i,j}^{(k)}}
\]
on termination. This facility has not been implemented in LSQR and we have not investigated the accuracy of such approximations. Clearly only a limited number of pairs \((i,j)\) could be dealt with efficiently on large problems.
6. STOPPING CRITERIA

An iterative algorithm must include rules for deciding whether the current iterate \( x_k \) is an acceptable approximation to the true solution \( x \). Here we shall formulate stopping rules in terms of three dimensionless quantities ATOL, BTOL and CONLIM, which the user will be required to specify. The first two rules apply to compatible and incompatible systems respectively. The third rule applies to both. They are:

\begin{enumerate}
\item [S1:] Stop if \( \| r_k \| \leq \text{BTOL}\|b\| + \text{ATOL}\|A\| \| x_k \| \).
\item [S2:] Stop if \( \frac{\| A^T r_k \|}{\| A \| \| r_k \|} \leq \text{ATOL} \).
\item [S3:] Stop if \( \text{cond}(A) \geq \text{CONLIM} \).
\end{enumerate}

We can implement these rules efficiently using the estimates of \( \| r_k \|, \| A \|, \) etc., already described.

The criteria S1 and S2 are based on allowable perturbations in the data. The user may therefore set ATOL and BTOL according to the accuracy of the data. For example, if \((A, b)\) is the given data and \((\tilde{A}, \tilde{b})\) represents the (unknown) true values, then

\[ \text{ATOL} = \frac{\| A - \tilde{A} \|}{\| A \|} \]

should be used if an estimate of this is available. Similarly for BTOL.

Criteria S3 represents an attempt to regularize ill-conditioned systems.
6.1 Compatible systems

To justify S1, let \( r_k = b - Ax_k \) as usual, and define the quantities

\[
\delta_k = \text{BTOL}\|b\| + \text{ATOL}\|A\| \|x_k\|,
\]

\[
g_k = \text{BTOL}\|b\| \frac{r_k}{\delta_k},
\]

\[
h_k = \text{ATOL}\|A\| \|x_k\| \frac{r_k}{\delta_k}.
\]

Then \( r_k = g_k + h_k \), and

\[
\begin{pmatrix}
x_k \\
x_k^T x_k
\end{pmatrix}
\begin{bmatrix}
A + \frac{h_k x_k^T}{x_k^T x_k}

\end{bmatrix}
\begin{pmatrix}
x_k \\
x_k^T x_k
\end{pmatrix}
= b - g_k
\]

so that \( x_k \) is the exact solution for a system with both \( A \) and \( b \) perturbed. It can be seen that these perturbations are within their allowable bounds when the inequality in S1 holds. Hence, criterion S1 is consistent with the ideas of backward rounding error analysis and with knowledge of data accuracy. Since this argument does not depend on orthogonality, S1 can be used in any method for solving compatible linear systems.

6.2 Incompatible systems

Stewart [25] has observed that if

\[
r_k = b - Ax_k
\]

and \( \tilde{r}_k = b - (A + E_k)x_k \)

where

\[
E_k = - \frac{r_k x_k^T A}{\|r_k\|^2},
\]

then \( (A + E_k)^T \tilde{r}_k = 0 \), so that \( x_k \) and \( \tilde{r}_k \) are the exact solution and residual for a system with \( A \) perturbed. Since \( \|E_k\|_2 = \|A^T r_k\| / \|r_k\| \), the perturbation to \( A \) will be negligible if the test in S2 is satisfied.
In our particular method it happens that \( B_k x_k = 0 \), since (5.3) shows that \( x_k = V_k y_k \) is theoretically orthogonal to \( A^T r_k \). Hence \( r_k = r_k \), so both \( x_k \) and \( r_k \) are exact for the perturbed system. This strengthens the case for using rule S2.

In practice we find that \( \|A^T r_k\| / \|r_k\| \) can vary rather dramatically with \( k \), but it does tend to stabilize for large \( k \), and the stability is more apparent for LSQR than for the standard method of conjugate gradients (see \( \|A^T r_k\| \) in Figures 3 and 4, section 8). Criterion S2 is sufficient to ensure that \( x_k \) is an acceptable solution to the least-squares problem, but the existence of an easily computable test that is both sufficient and necessary remains an open question (Stewart [25]).

### 6.3 Ill-conditioned systems

Stopping rule S3 is a heuristic based on the following arguments. Suppose that \( A \) has singular values \( \sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n > 0 \). It has been observed in some problems that as \( k \) increases the estimate \( \|B_k\|_p \|D_k\|_p \approx \text{cond}(A) \) in (5.10) temporarily levels off near some of the values of the ordered sequence \( \sigma_1/\sigma_1, \sigma_1/\sigma_2, \ldots, \sigma_1/\sigma_n \), with varying numbers of iterations near each level. This tends to happen when the smaller \( \sigma_i \) are very close together, and therefore suggests criterion S3 as a means of regularizing such problems when they are very ill-conditioned, as in the discretization of ill-posed problems (e.g. Nashed [16]).

For example, if the singular values of \( A \) were known to be of order \( 1, 0.9, 10^{-3}, 10^{-6}, 10^{-7} \), the effect of the two smallest singular values could probably be suppressed by setting \( \text{CONLIM} = 10^4 \).

A more direct interpretation of rule S3 can be obtained from the fact that \( x_k = D_k f_k \). First, suppose that the singular value decomposition of \( A \)
is $A = UV^T$ where $U^TV = V^TU = I$, $\Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$, and let

$$A^{(r)} = U \Sigma^{(r)} V^T$$

be defined by setting $\sigma_{r+1} = \ldots = \sigma_n = 0$. A common method for regularizing the least-squares problem is to compute $x^{(r)} = V \Sigma^{(r)} + U^T b$ for some $r \leq n$, since it can readily be shown that the size of $x^{(r)}$ is bounded according to

$$\frac{\|A\|_2 \|x^{(r)}\|}{\|b\|} \leq \frac{\sigma_1}{\sigma_r} = \text{cond}(A^{(r)})$$

In the case of LSQR, we have $\|x_k\| \leq \|B_k\|_F \|b\|$ and so if rule S3 has not yet caused termination we know that $\|B_k\|_F \|x_k\| / \|b\| \leq \|B_k\|_F \|B_k\|_F < \text{CONLIM}$. Since $\|B_k\|_F$ usually increases to order $\|A\|_F$ quite early, we effectively have

$$\frac{\|A\|_F \|x_k\|}{\|b\|} < \text{CONLIM},$$

which is exactly analogous to the bound above.

### 6.4 Singular systems

It is sometimes the case that rank($A$) $< n$. Known dependencies can often be eliminated in advance, but others may remain if only through errors in the data or in formulation of the problem.

With conventional (direct) methods it is usually possible to detect rank deficiency and to advise the user that dependencies exist. In the present context it is more difficult to provide such useful information, but we recognise the need for a method that at least does not fail when applied (perhaps unknowingly) to a singular system. In such cases we again suggest the parameter CONLIM as a device for controlling the computation. Our experience with LSQR on singular problems is that convergence to an acceptable solution occurs normally, but if iterations are allowed to
continue the computed $x_k$ will begin to change again and then grow quite rapidly until
\[
\frac{\|A\| \|x_k\|}{\|b\|} \approx \frac{1}{\epsilon}
\] (6.1)
(while $\|r_k\|$ remains of reasonable size). The estimate of $\text{cond}(A)$ typically grows large before the growth in $x_k$. A moderate value of $\text{CONLIM}$ (say $1/\epsilon^k$) may therefore cause termination at a useful solution.

In some cases it can be useful to set $\text{CONLIM}$ as large as $1/\epsilon$ and allow $x_k$ to diverge. In this context we note that the algorithm SYMMLQ [21] can be applied to singular symmetric systems and that extreme growth in the resulting $\|x_k\|$ forms an essential part of a practical method for computing eigenvectors of large symmetric matrices (Lewis [15]). By analogy, in the presence of rounding errors LSQR will usually produce an approximate singular vector of the matrix $A$. In fact, using (6.1) and $\|r_k\| \leq \|b\|$ we see that the normalized vector $\overline{x}_k = x_k / \|x_k\|$ will usually satisfy
\[
\overline{Ax}_k = \frac{1}{\|x_k\|} (b - r_k) = \epsilon \frac{\|A\|}{\|b\|} (b - r_k)
\]
for large enough $k$, and hence will lie very nearly in the null space of $A$. The vector $\overline{x}_k$ may reveal to the user where certain unexpected dependencies exist. Suitable new rows could then be added to $A$, or certain linear constraints could be applied directly, as described in section 9.4.
7. OTHER METHODS

Several other conjugate-gradient methods are discussed here. All except the first (CGLS) are stated using notation consistent with sections 3-6, in order to illustrate certain analytic identities.

7.1 CGLS

If the conjugate-gradient method for symmetric positive definite systems is applied naively to the normal equations $A^T A x = A^T b$, the method does not perform well on ill-conditioned systems. To a large extent this is due to the explicit use of vectors of the form $A^T A p_i$. An algorithm with better numerical properties is easily derived by a slight algebraic rearrangement, making use of the intermediate vector $A p_i$ (e.g. Hestenes and Stiefel [11]). It is usually stated in notation similar to the following.

Algorithm CGLS

1. Set $r_0 = b$, $s_0 = A^T b$, $p_1 = s_0$, $y_0 = ||s_0||^2$, $x_0 = 0$.
2. For $i = 1, 2, 3 \ldots$ repeat the following:

(a) $q_i = A p_i$
(b) $\alpha_i = y_{i-1} / ||q_i||^2$
(c) $x_i = x_{i-1} + \alpha_i p_i$
(d) $r_i = r_{i-1} - \alpha_i q_i$
(e) $s_i = A^T r_i$
(f) $y_i = ||s_i||^2$
(g) $\beta_i = y_i / y_{i-1}$
(h) $p_{i+1} = s_i + \beta_i p_i$. 
A practical implementation of the method would also need to compute \( \|r_i\|, \|x_i\| \) and an estimate of \( \|A\| \) in order to use the stopping criteria developed in section 6. Otherwise the method is clearly simple and economical. Analytically it generates the same points \( x_i \) as LSQR. There is no simple connection with either of the bidiagonalizations, but the vectors \( v_{i+1} \) and \( d_i \) in LSQR are proportional to \( b_i \) and \( p_i \) respectively.

7.2 Craig's method for \( Ax = b \)

A very simple method is known for solving compatible systems \( Ax = b \). This is Craig's method, as described in Faddeev and Faddeeva [7]. It is derivable from Bidiag 1 as shown by Paige [18] and differs from all other methods discussed here by minimizing the error norm \( \|x_k - x\| \) at each step, rather than the residual norm \( \|b - Ax_k\| = \|A(x_k - x)\| \). We review the derivation briefly.

If \( L_k \) is the first \( k \) rows of \( B_k \),

\[
L_k = \begin{bmatrix}
\alpha_1 \\
\beta_2 & \alpha_2 \\
& \ddots \\
& & \ddots \\
& & & \beta_k & \alpha_k
\end{bmatrix}
\]

then equations (3.3) - (3.4) describing Bidiag 1 may be restated as

\[
AV_k = U_kL_k + \beta_{k+1}U_{k+1}c_k^T,
\]

\[
A^T U_k = V_kL_k^T.
\]

(7.1)
Craig's method is defined by the equations

\[ L_k \alpha_k = B_1 e_1, \quad x_k = V_k \alpha_k, \quad (7.2) \]

and we can show from (7.1) that the residual vector satisfies

\[ r_k = b - AV_k \alpha_k = -\zeta_k \beta_k u_{k+1}, \quad \text{and hence} \quad \alpha_k^T r_k = 0. \quad \text{We can therefore expect} \quad r_k \quad \text{to vanish (analytically) for some} \quad k \leq n. \]

The vectors \( z_k \) and \( x_k \) are readily computed from

\[ \zeta_k = -\frac{\beta_k}{\alpha_k^T}, \quad x_k = x_{k-1} + \zeta_k v_k, \]

where \( \zeta_0 = -1 \). Since the increments \( v_k \) form an orthogonal set there is no danger of cancellation, and the step-lengths \( \zeta_k \) are bounded by \( |\zeta_k| \leq \|x_k\| = \|x_k\| \leq \|x\| \). We can therefore expect the method to possess good numerical properties. This is confirmed by the comparison in section 8.

### 7.3 Extension of Craig's method

A scheme for extending Craig's method to least-squares problems was suggested by Paige in [18]. The vectors in (7.2) were retained and an additional vector of the form \( V_k \omega_k \) was computed in parallel. On termination, a suitable scalar \( \gamma_k \) was computed and the final solution taken to be

\[ x_k = (V_k \omega_k) - \gamma_k (V_k \omega_k) \equiv V_k \gamma_k, \quad (7.3) \]

In the present context this method may be interpreted as a means of solving the least-squares system (2.7), viz.

\[ \begin{bmatrix} I & B_k \end{bmatrix} \begin{bmatrix} t_{k+1} \\ y_k \end{bmatrix} = \begin{bmatrix} \beta_1 e_1 \\ 0 \end{bmatrix}, \quad (7.4) \]

using the fact that the underdetermined system \( B_k^T t_{k+1} = 0 \) has a unique
solution apart from a scalar multiple. Thus if we partition \( B_k^T \) and \( t_{k+1} \) according to

\[
B_k^T = \begin{bmatrix} \alpha_1 e_1 & \bar{L}_k \end{bmatrix}, \quad t_{k+1} = \gamma_k \begin{bmatrix} 1 \\ \bar{t}_k \end{bmatrix}
\]

for some \( \gamma_k \), the system \( \bar{L}_k \bar{t}_k = -\alpha_1 e_1 \) can be solved easily by forward substitution, and then from the top half of (7.4) we obtain the required equations

\[
L_k \omega_k = \begin{bmatrix} 1 \\ \bar{t}_{k-1} \end{bmatrix}, \quad \gamma_k = \beta_{k+1} / (\bar{L}_{k+1} \omega_k - \bar{t}_k)
\]

using straightforward notation. The first equation indicates that \( \omega_k \) can be computed by forward substitution, thus allowing \( V_k \omega_k \) to be accumulated as the algorithm progresses. The second shows how \( \gamma_k \) can be computed at each step (if desired) and at the final iteration for use in (7.3).

In the original presentation of this method the danger of cancellation in (7.3) was recognized, in the event that \( V_k \alpha_k \) and \( \gamma_k \) were large and nearly equal. When the system \( Ax = b \) is compatible the danger does not exist because each vector \( V_k \alpha_k \) is the same as in Craig's method and remains of reasonable size. We have observed in practice that \( \| \omega_k \| \) actually diverges with increasing \( k \), but the scalars \( \gamma_k \) converge to zero at a greater rate. The points \( x_k = V_k \alpha_k - \gamma_k V_k \omega_k \), if computed, are at all stages identical to those produced by LSQR, and for sufficiently large \( k \) are computationally just \( x_k = V_k \alpha_k \).

When \( b \) does not lie in the range of \( A \) we find that severe cancellation can occur, with the \( \gamma_k \) stabilizing at a moderate value but \( \| \alpha_k \| \) and \( \| \omega_k \| \) both diverging. For this reason the extension of Craig's method must be discarded.
Note that the divergence of $\|x_k\|$ in these circumstances shows that $L_k$ in (7.2) can be much more ill-conditioned than $A$. If Bidiag 1 is used as in [17] to estimate the singular values of $A$, it is the singular values of $R_k$ rather than $L_k$ that are relevant.

7.4 LSCG and LSLQ

A second algorithm for least-squares problems was given by Paige [18]. This is algorithm LSCG, based on Bidiag 2. In the notation of section 3 it is defined by the equations

$$B_k^T R_k y_k = \theta_1 e_1, \quad x_k = V_k y_k$$

and implemented in the form $R_k^T f_k = \theta_1 e_1, \quad x_k = (V_k R_k^{-1}) f_k$. Given the relations between the two bidiagonalizations we now recognize that this is analytically equivalent to LSQR, but numerically inferior, since it is effectively solving the least-squares problem $\min \|B_k y_k - \beta_i e_i\|$ by using the corresponding normal equations. (The latter are $B_k^T B_k y_k = B_k^T \beta_1 e_1$, $\alpha_i \beta_i e_1$ and by (3.9) and (3.12) this is equivalent to (7.5a).)

Algorithm LSIQ (Paige and Saunders [20]) is a refinement of LSCG, but again it is based on Bidiag 2 and the above normal equations and is therefore inferior to LSQR on ill-conditioned problems. The refinement has been described in section (5.2), giving $x_k = \overline{w}_k z_k$ where $\overline{w}_k$ is theoretically orthonormal, the intention being to avoid any possible cancellation that could occur in accumulating $x_k = D_k f_k = (V_k R_k^{-1}) f_k$. The same refinement can easily be made to LSQR, and it was implemented in an earlier version of the algorithm for the same reason. However, we have not been able to detect any numerical difference between $x_k = \overline{w}_k z_k$ and $x_k = D_k f_k$ in the two versions of LSQR, so the fear of cancellation appears to have been unfounded. We
have therefore retained the slightly more economical \( x_k = D_k f_k \), which also allows \( \text{cond}(A) \) to be estimated from \( ||D_k||_F \) as already described.

Algorithms LSCG and LSLQ need not be considered further.

7.5 Chen's algorithm RRLS

Another algorithm based on Bidiag 2 has been described by Chen [41]. This is algorithm RRLS, and it combines Bidiag 2 with the so-called residual-reducing method of Householder [12]. In the notation of section 3 it may be described as follows. The residual-reducing property is implicit in steps 2(b) and 2(c).

Algorithm RRLS

1. Set \( r_0 = b, \theta_1 v_1 = A^T b, \omega_1 = v_1, x_0 = 0 \).

2. For \( i = 1, 2, 3, \ldots \) repeat the following:

   (a) \( \rho_i p_i = \lambda_i \omega_i \)

   (b) \( \lambda_i = p_i^T r_i \)

   (c) \( r_i = r_{i-1} - \lambda_i p_i \)

   (d) \( \theta_{i+1} v_{i+1} = A^T p_i - \rho_i v_i \)

   (e) \( x_i = x_{i-1} + (\lambda_i / \rho_i) \omega_i \)

   (f) \( \omega_{i+1} = v_{i+1} - (\theta_{i+1} / \rho_i) \omega_i \)

where the scalars \( \rho_i \) and \( \theta_i \) are chosen so that \( ||p_i|| = ||v_i|| = 1 \).

As with CGLS, a practical implementation would also require \( ||r_i|| \) and \( ||x_i|| \). The square root of the sum \( \sum_{i=1}^{k} (\rho_i^2 + \theta_{i+1}^2) = ||R_k||_F^2 = ||\theta_k||_F^2 \) could be used to estimate \( ||A||_F \) and \( ||A^T r_i|| \) can also be estimated cheaply.

Note that the vectors \( v_i \) are generated as in Bidiag 2, but the vectors \( p_i \) come instead from step 2(a). Substituting the latter into step 2(d)
shows that RRLS requires explicit computation of the vectors $A^T A \omega_i$ (ignoring normalization by $r_i$). Unfortunately this must cast doubt on the numerical properties of the method, particularly when applied to compatible systems. Indeed we find that for some systems $A x = b$, the final norms $\|r_i\|$ and $\|x_i - x\|$ are larger, by a factor approaching $\text{cond}(A)$, than those obtained by CGLS and LSQR. This is illustrated in section 8.3.

A second algorithm called RRLSL has been described by Chen [4] in which the residual-reducing method is combined with Bidiag 1. However, the starting vector used is $A A^T b$ (rather than $b$), and products of the form $A^T A \omega_i$ are again required, so that improved performance seems unlikely. Chen reports that RRLS and RRLSL behaved similarly in all test cases tried.

In spite of the above comments we have also observed ill-conditioned least-squares problems for which RRLS obtains far greater accuracy than would normally be expected of any method (see section 8.4 for a possible explanation). Because of this unusual behavior we have investigated a residual-reducing version of LSQR as now described.

### 7.6 RRLSQR

If the residual vector $r_i$ is explicitly introduced, algorithm LSQR as summarized in section 4.1 can be modified slightly. First, the residual-reducing approach requires step 5(a) to be replaced by the two steps

$$r_i = r_{i-1} - \lambda_i p_i,$$
$$x_i = x_{i-1} + \lambda_i \omega_i,$$

where $p_i = A \omega_i$ and $\lambda_i = p_i^T r_{i-1} / \|p_i\|^2$. (In this case $p_i$ is unnormalized.) Second, the product $A \omega_i$ can be used to eliminate $A \omega_i$ from Bidiag 1, leading to an alternative method

$$B_{i+1} x_{i+1} = A \omega_i - \frac{\bar{p}_i}{\|r_{i-1}\|} r_{i-1}$$

(7.6)
for generating each $\beta_i$ and $u_i$. (This result is difficult to derive, but the key relation is $p_i / \|p_i\| = c_i r_{i-1} / \|r_{i-1}\| + s_i u_{i+1}$, which may be deduced from (3.11).)

The remainder of LSQR is retained, including the QR factorization of $B_k$. The coefficient of $r_{i-1}$ in (7.6) can be expressed in several ways; for example

$$\frac{\bar{v}_i}{\|r_{i-1}\|} = \frac{\bar{\rho}_i}{\phi_i} = \frac{1}{\tau_i} = (-1)^{i-1} \frac{\alpha_1 \alpha_2 \cdots \alpha_i}{\beta_1 \beta_2 \cdots \beta_i}$$

where $\tau_i$ comes from the system $L_k B_k = \beta_1 e_1$ of Craig's method. Different formulae lead to different iteration paths but no variation appears to be consistently better than the rest.

A summary of the resulting algorithm follows.

Algorithm RRLSQR

1. $r_0 = b$, $\beta_1 u_1 = b$, $\alpha_1 v_1 = A^T u_1$, $\omega_1 = v_1$, $x_0 = 0$,
   $\bar{\phi}_1 = \beta_1$, $\bar{\rho}_1 = \alpha_1$.

2. For $i = 1, 2, 3, \ldots$ repeat steps 3 - 6.

3. (a) $p_i = \lambda \omega_i$
   (b) $\lambda_i = p_i^T r_{i-1} / \|p_i\|^2$
   (c) $r_i = r_{i-1} - \lambda_i p_i$
   (d) $\beta_{i+1} u_{i+1} = p_i - (\bar{\rho}_i / \bar{\phi}_i) r_{i-1}$
   (e) $\alpha_{i+1} v_{i+1} = A^T u_{i+1} - \beta_{i+1} v_i$

4. Compute $\rho_i$, $s_i$, $\bar{\theta}_i$, $\bar{\rho}_{i+1}$, $\bar{\phi}_{i+1}$ as in section 4.1, step 4.

5. (a) $x_i = x_{i-1} + \lambda_i \omega_i$
   (b) $\omega_{i+1} = v_{i+1} - (\bar{\theta}_{i+1} / \rho_i) \omega_i$

6. Exit if appropriate.
This adaption of Bidiag 1 to obtain RRLSQR is analogous to (and was motivated by) Chen's adaption of Bidiag 2 to obtain RRLS. Note however that there are no products of the form $A^T Av$. In practice we find that RRLSQR typically performs at least as well as LSQR, as measured by the limiting $\|x_\ell - x\|$ attainable. Furthermore, it attains the same unusually high accuracy achieved by RRLS on certain ill-conditioned least-squares problems. On these grounds RRLSQR could sometimes be the preferred method. However, its work and storage requirements are significantly higher than for the other methods considered.

7.7 Storage and work

The storage and work requirements for the most promising algorithms are summarized below. Recall that $A$ is $m$ by $n$ and that for least-squares problems $m$ may be considerably larger than $n$. Craig's method is applicable only to compatible systems $Ax = b$, which usually means $m = n$.

<table>
<thead>
<tr>
<th>Method</th>
<th>Storage</th>
<th>Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Craig ($Ax = b$ only)</td>
<td>$u, Av$</td>
<td>$x, v$</td>
</tr>
<tr>
<td>CGLS</td>
<td>$r, q$</td>
<td>$x, p, s$</td>
</tr>
<tr>
<td>LSQR</td>
<td>$u, Av$</td>
<td>$x, v, w$</td>
</tr>
<tr>
<td>RRLS</td>
<td>$r, p$</td>
<td>$x, v, w, A^T p$</td>
</tr>
<tr>
<td>RRLSQR</td>
<td>$r, u, p$</td>
<td>$x, v, w$</td>
</tr>
</tbody>
</table>

All methods require the starting vector $b$. If necessary this may be overwritten by the first $m$-vector shown ($r$ or $u$). The $m$-vector $Av$ shown for Craig and LSQR represents working storage to hold products of the
form $Av$ and $A^T\mu$. (An $n$-vector would be needed if $m < n$.) In some applications this could be dispensed with if the bidiagonalization operations $Av - \alpha u$ and $A^T\mu - \beta v$ were implemented to overwrite $u$ and $v$ respectively. Similarly the $n$-vector $A^Tp$ for RRLS could in some cases be computed without extra storage.

The work shown for each method is the number of multiplications per iteration. For example, LSQR requires $3m + 5n$ multiplications. (A further $2n$ multiplications are needed to accumulate estimates of $\text{cond}(A)$ and standard errors for $x$.) Practical implementations of CGLS and RRLS would require a further $m + n$ multiplications to compute $\|r_\ell\|$ and $\|x_\ell\|$ for use in stopping rules, although this could be limited to every tenth iteration, say, without serious consequence.

All methods require one product $Av$ and one product $A^Tv$ each iteration. This could dominate the work requirements in some applications.
8. NUMERICAL COMPARISONS

Here we compare LSQR numerically with four of the methods discussed in section 7, denoted by CRAIG, CGLS, RRLS and RRLSQR. The machine used was a Burroughs B6700 with relative precision $\epsilon = 0.5 \times 8^{-12} = 0.7 \times 10^{-11}$.

The results given here are complementary to those given by Elfving [6], who compares CGLS with several other conjugate-gradient algorithms and also investigates their performance on problems where $A$ is singular.

8.1 Generation of test problems

The following steps may be used to generate a test problem $\min \|b - Ax\|$ with known solution $x$.

1. Choose vectors $x, y, z, c$ and diagonal matrix $D$ arbitrarily, with $\|y\| = \|z\| = 1$. (For any chosen $m \geq n$, the vectors should be of dimensions $n, m, n$ and $m-n$ respectively.)

2. Define $Y = I - 2yy^T$, $Z = I - 2zz^T$, $A = Y \begin{bmatrix} D \\ 0 \end{bmatrix} Z$.

3. Compute $r = Y \begin{bmatrix} 0 \\ c \end{bmatrix}$, $b = Ax + r$.

The minimal residual norm is then $\|r\| = \|c\|$. Since $A$ and $D$ have the same singular values, the condition of the problem is easily specified.

The particular problems used here will be called $P(m, n, d, p)$ to indicate dependence on four integer parameters, where $d$ represents duplication of singular values and $p$ is a power factor. The matrix $D$ is of the form $\text{diag}(\sigma_1^P)$ with each $\sigma_1$ duplicated $d$ times. Specific values for $x, y, z, c$ and $D$ were generated as follows:

1. $x = (n-1, n-2, n-3, \ldots, 2, 1, 0)^T$. 
2. $y_i = \sin(4\pi i / m)$, $z_i = \cos(4\pi i / n)$, followed by normalization so that 
$$\|y\| = \|z\| = 1.$$ 
3. $\sigma = (1/m, -2/m, 3/m, \ldots, + (m-n) / m)^T$.
4. $\sigma_i = \left[\frac{i-1+d}{d}\right] d/n$, where integer division is used for the term in parentheses. Choosing $n = qd$ to be a multiple of $d$ leads to $d$ copies of each singular value: 
$$(\sigma_i) = \left\{ \frac{1}{q}, \ldots, \frac{1}{q}, \frac{2}{q}, \ldots, \frac{2}{q}, \ldots, \frac{q}{q}, \ldots, \frac{1}{q} \right\}.$$ 

[For reference, this gives:

$$\|x\| = n(n/3)^{1/2}, \quad \|r\| = \|a\| = \frac{m-n}{m} \left( (m-n) / 3 \right)^{1/2},$$
$$\|A\| = \sigma_{\max} = (\sigma_n / \sigma_1)^{1/2} = q^{1/2}.\]"

The orthogonal matrices $Y$ and $Z$ are intended to reduce the possibility of anomalous numerical behavior. For example, when LSQR was applied to four cases of the problem $P(10, 10, 1, 8)$ the following error norms resulted:

<table>
<thead>
<tr>
<th>Case</th>
<th>$\log_{10}|x_k - x|$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m=n=10$, $\text{cond}(A) = 10^8$</td>
<td>$k=60$</td>
</tr>
<tr>
<td>1</td>
<td>$A = YDZ$ (as above)</td>
</tr>
<tr>
<td>2</td>
<td>$A = YD$</td>
</tr>
<tr>
<td>3</td>
<td>$A = DZ$</td>
</tr>
<tr>
<td>4</td>
<td>$A = D$</td>
</tr>
</tbody>
</table>

Since each case was a compatible system $Ax = b$, we normally would expect an error norm approaching $\|x\| \cdot \text{cond}(A) \cdot \epsilon = 10^{-2}$, so that case 1 is the most realistic. In case 2 the error was concentrated in the first and second components of $x_k$ (with the remaining components accurate almost to working precision), whereas in cases 3 and 4 the final $x_k$ was virtually exact in spite of the high condition number of $A$. 
Although cases 2-4 represent less expensive test problems, it is clear that results obtained from them could be very misleading. In the following sections we use only case 1. Even these test problems may be less than completely general. This is discussed in section 8.4.

8.2 $Ax = b$; failure of CGLS

Figure 1 illustrates the performance of five methods on the ill-conditioned system $P(10, 10, 1, 8)$, i.e. $m = n = 10$, one copy of each singular value, $\text{cond}(A) = 10^8$. The quantities $\log_{10}||r_k||$ and $\log_{10}||x_k - x||$ are plotted against iteration number $k$.

This example distinguishes the standard conjugate-gradient method CGLS from the remaining methods. All except CGLS reduced $||r_k||$ and $||x_k - x||$ to a satisfactory level before $k = 120$.

Also apparent is the erratic behavior of $||r_k||$ for method CRAIG, a potential penalty for minimizing $||x_k - x||$ at each step without regard to $||r_k||$. In theory all other methods minimize $||r_k||$ at each step and also reduce $||x_k - x||$ monotonically.

If any method is to be preferred in this example it would be LSQR, since it reached limiting accuracy at iteration 76 and stayed at essentially the same point thereafter. With values $\text{ATOL} = \text{BTOL} = \epsilon$ the stopping rule S1 as implemented in LSQR would have caused iteration at $k = 76$ as desired.

8.3 $Ax = b$; failure of RRLS

Figure 2 illustrates the same five methods applied to a larger problem $P(40, 40, 4, 7)$, in which each singular value is repeated four times and $\text{cond}(A) = 10^7$. In this case all methods except RRLS reduced $||r_k||$
satisfactorily to about $10^{-9}$ for $k \geq 105$. For method RRLS, $\|r_k\|$ remained of order $10^{-5}$ for $k \geq 30$ up to $k = 250$, and zero digits of accuracy were obtained in $x_k$.

A similar disparity between RRLS and the remaining methods was observed on the problems $P(40,40,4,p)$, $p = 5,6$, $\text{cond}(A) = 10^P$. In fairness, Chen [4] did not intend RRLS to be applied to compatible systems. However, the success of the other least-squares methods suggests that this is not an unreasonable demand.

8.4 $\min\|Ax - b\|$; high accuracy by RRLS and RRLSQR

Figure 3 shows the performance of four least-squares methods on the ill-conditioned problem $P(20,10,1,6)$. Since $\text{cond}(A)^2 = 10^{12} \approx 1/\varepsilon$, we would normally expect at most one digit of accuracy in the final $x_k$. This is achieved by LSQR and CGLS, with LSQR showing a smoother decrease of $\|A^T r_k\|$.

In contrast, the residual-reducing methods achieved at least six digits of accuracy in $x_k$. Similarly, three or four digits of accuracy were obtained on the problem $P(20,10,1,6)$, for which $\text{cond}(A) = 10^8$ is so high that no digits could be expected. At first sight it may appear that the residual-reducing methods possess some advantage on least-squares problems. However, this anomalous behavior cannot be guaranteed; for example, it did not occur on $P(80,40,4,6)$, as shown in Figure 4. Also, the final value of $\|A^T r_k\|$ is no smaller than for LSQR and this is really the more important quantity.

Part of the explanation for these occasional anomalies may lie in the following. Suppose the original data $(A,b)$ have solution and residual $(\hat{x},\hat{r})$, while perturbed data $(A + \delta A, b + \delta b)$ have $(\hat{x} + \delta x, \hat{r} + \delta r)$. If
A + δA has full column rank then it is straightforward to show that
\[
δx = (A + δA)^+(δb - δA\hat{x}) + ((A + δA)^T(A + δA))^{-1}δA^T\hat{r}
\]
In the present example \(ε = 0.7 \times 10^{-11}, \|A\|_2 = 1, \text{cond}(A) = 10^6, \|b\| = 2.4, \|\hat{r}\| = 17, \|\hat{r}\| = 7\). If the perturbations were caused by rounding errors in the initial data then \(\|\delta A\| = ε, \|\delta b\| = ε\), and the first term in the expression for \(δx\) could be about as large as \(10^{-4}\) in norm, and the second could be of order 7. Figure 3 suggests the second term is unusually small for the RRLS and RRLSQR computations. Looking at the particular form of the test problem, if we write
\[Y = [Y_1, Y_2], \quad A = Y_1DZ, \quad \hat{r} = Y_2c,\]
we have
\[(A + δA)^+ = A^+ = Z^TD^{-1}Y_1^T,
\]
and the second term in \(δx\) is effectively
\[Z^TD^{-2}δA^TY_2^c.
\]
Now suppose \(δA\) is simply an equivalent perturbation in \(A\) caused when \(A\) multiplies a vector \(v\) in our test case. Using the results of rounding error analyses given by Wilkinson [27],
\[(A + δA)v ≡ f_1(Y_1f_1(Df_1(Zv))) = (Y_1 + δY_1)(D + δD)(Z + δZ)v\]
where \(\|δY_1\| ≈ ε\|Y_1\|\) etc., and so
\[δA ≡ δY_1(D + δD)(Z + δZ) + Y_1(DδZ + δD(Z + δZ)).\]
Using this \(δA\) in the second term for \(δx\) effectively gives
\[Z^TD^{-1}(I + D^{-1}Z\delta Z^TD)(I + D^{-1}δD)δY_1^TY_2^c\]
which is bounded above by about \(7 \times 10^{-6}\) in norm, rather than 7 as expected.
This gives a hint of what might be happening above, since a more realistic problem would not admit such a relation between rounding errors and residual. This does not invalidate the other numerical comparisons, but it does emphasize the care needed when constructing artificial test problems.
8.5 \( \min \| Ax - b \| \); normal behavior

Figure 4 illustrates more typical performance of the four methods, using the least-squares problem \( P(80,40,4,6) \) for which \( \text{cond}(A) = 10^6 \). All methods reduced \( \| A^T r_k \| \) to a satisfactory level, and the final error norm is consistent with a conventional sensitivity analysis of the least-squares problem; in this case no more than one significant digit can be expected. Note that CGLS converged more slowly than the other methods. It also displayed rather undesirable fluctuations in \( \| A^T r_k \| \) considerably beyond the point at which the other methods reached limiting accuracy.

8.6 Some results using greater precision

Algorithm LSQR was also applied to the previous four test problems on an IBM 370 computer using double precision arithmetic, for which \( \epsilon = 2.2 \times 10^{-16} \). With increased precision, LSQR gave higher accuracy and also required fewer steps to attain this accuracy. This is best seen by referring to the figures. In Figure 1 the log of the residual reached \(-14.4\) at the 48th step and stayed there; the log of the error was then \(-8.8\) but decreased 20 steps later to \(-9.3\) and stayed there. In Figure 2 the logs of the residual and error were \(-13.8\) and \(-8.0\) at step 44 and differed negligibly from these values thereafter. In Figure 3, \( \log_{10} \| A^T r_k \| = -14.6 \) and \( \log_{10} \| x_k - x \| = -6.0 \) at \( k = 32 \) and thereafter, while in Figure 4, \( \log_{10} \| A^T r_k \| = -13.9 \) and \( \log_{10} \| x_k - x \| = -4.6 \) at \( k = 36 \) with little change for larger \( k \).
Figure 1. An ill-conditioned system $Ax = b$, $n = 10$, $\text{cond}(A) = 10^8$.

CGLS is unable to reduce $\|r_k\|$ or $\|x_k - x\|$ satisfactorily. CRAIG exhibits severe fluctuations in $\|r_k\|$.
Figure 2. An ill-conditioned system $Ax = b$, $n = 40$, $\text{cond}(A) = 10^7$. RRLS is unable to reduce $\|r_k\|$ or $\|x_k - x\|$ satisfactorily.
Figure 3. An ill-conditioned problem, \( \min \| Ax - b \| \), \( m = 20 \), \( n = 10 \), \( \text{cond}(A) = 10^6 \).

RRLS and RRLSQR achieve anomalously high accuracy in \( x_k \).
Figure 4. An ill-conditioned problem, $\min \|Ax - b\|$, $m = 80$, $n = 40$, $\text{cond}(A) = 10^6$.

All methods obtain a satisfactory solution, although CGLS exhibits slower convergence and undesirable fluctuations in $\|A^T r_k\|$. 
9. TRANSFORMATIONS

Here we discuss various ways of transforming the problem \( \min \|Ax - b\| \)
with a view to reducing the total work required for its solution.

As in the case of symmetric systems \( Ax = b \) (e.g. Chandra [3]), the
main aim behind scaling, partitioning, splitting, etc. is to improve the
distribution of the singular values of \( A \) so that at least one of the
following occurs:

(a) \( \text{cond}(A) = \frac{\sigma_{\text{max}}}{\sigma_{\text{min}}} \) is reduced, where \( \sigma_{\text{min}} \) is the smallest nonzero
singualar value;

(b) the number of distinct singular values is reduced.

Application of a conjugate-gradient algorithm to the transformed problem
should then result in more rapid convergence.

9.1 Partitioning by columns

It sometimes happens that the matrix \( A \) occurs naturally in the form
\( A = [B \ C] \) where the first partition \( B \) has special structure. For example,
\( B \) may be block diagonal, or the matrix \( B^TB \) may be diagonal or tridiagonal.
In other cases \( \text{cond}(A) \) may be large primarily because \( \text{cond}(B) \) is large.
We now show that as long as \( B \) has full column rank, we can always transform
the problem \( \min \|Bz + Cz - b\| \) into one of better condition and reduced
size. The strategy is to compute \( z \) separately using a conjugate-gradient
method on a problem of the form
\[
\min_{z} \| Tz + Tz - b \| \quad (9.1)
\]
for some matrix \( T \) and then obtain \( y \) using a direct method on the problem
\[
\min_{y} \| By - (b - Cz) \|. \quad (9.2)
\]
The form of \( T \) will depend on how much is known about \( B \), but in all cases
we shall have \( \text{cond}(T) \leq \text{cond}(A) \).
In general, let the QR factorization of \( B \) be
\[
B = [Y \quad Z] \begin{bmatrix} N \\ 0 \end{bmatrix} = YR
\]
where \( Q^T = [Y \quad Z] \) is orthogonal and \( R \) is upper triangular. If \( B \) has only a few columns then it is possible to construct \( Q \) and \( R \) cheaply using a product of Householder transformations or a product of plane rotations. In other circumstances only \( Y \) and \( R \) will be known, while if \( B \) is very large perhaps only \( R \) can be stored economically. In the latter case, it may be necessary to obtain \( R \) from the Cholesky factorization \( B^TB = R^TR \).

It can now be verified that any of the following matrices may serve as \( T \) in (9.1):
\[
T_1 = Z^T, \quad T_2 = I - YY^T, \\
T_2 = ZZ^T, \quad T_4 = I - B(B^TB)^{-1}B^T = I - BR^{-1}R^T B^T.
\]

In the first case, given that \( Q \) is orthogonal, we have
\[
\min \| Ax - b \| = \min \| QAx - QB \| \\
= \min \| \begin{bmatrix} R & Y^TC \\ 0 & ZZ^T \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} - \begin{bmatrix} y^Tb \\ z^Tb \end{bmatrix} \| \\
= \min \| ZZ^T z - z^T b \|
\]
since we can obtain \( y \) from \( Ry + y^TCz = y^Tb \) for any \( z \). (This is equivalent to (9.2).) Clearly we have \( \text{cond}(A) = \text{cond}(QA) \geq \text{cond}(Z^TC) \).

The result for \( T_2 \) is proved similarly using the fact that the matrix \( \bar{Q} \) below is orthogonal. Thus:
\[
\bar{Q} = \begin{bmatrix} 0 & y^T \\ y & ZZ^T \end{bmatrix}
\]
\[ \text{min} \| Ax - b \| = \text{min} \| \bar{Q} \begin{bmatrix} 0 & x - \bar{Q} \begin{bmatrix} a \\ b \end{bmatrix} \end{bmatrix} \| = \text{min} \| \begin{bmatrix} R & y^T C \\ 0 & ZZ^T C \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} - \begin{bmatrix} y^T b \\ ZZ^T b \end{bmatrix} \| . \]

Again it is clear that \( \text{cond}(A) \geq \text{cond}(ZZ^T C) \). In fact the operator \( T_2 = ZZ^T \) is of no use in practice, but it serves to prove the result for the remaining cases, using the relations \( T_2 = T_3 = T_4 \) and \( y = BR^{-1} \).

On numerical grounds \( T_1 = Z^T \) should be used whenever possible, particularly if \( \text{cond}(B) \) is large. Also, this is the only operator for which the transformed problem (9.1) has a reduced row dimension.

If \( T = T_3 \) or \( T = T_4 \) must be used, it is important to note that \( T = T^2 = T^T \) is a projection operator. When \( A \) and \( b \) are replaced by \( TC \) and \( Tb \) in Bidiag 1, the vectors in (3.1) satisfy \( Tu_i = u_i \). Hence the matrix-vector products required in the conjugate-gradient algorithms are \( TC v_i \) and just \( C^T u_i \) (not \( C^T Tu_i \)).

Note also that the product \( TC v_i \) may be regarded as the original matrix \( A \) operating on a vector, thus

\[ TC v_i = C v_i - B z_i = A \begin{bmatrix} -z_i \\ v_i \end{bmatrix} \]

where \( z_i \) solves the least-squares problem \( \text{min} \| B z_i - C v_i \| \). Each iteration is therefore essentially the same as for the original problem \( \text{min} \| Ax - b \| \) with the addition of a single subproblem involving the partition \( B \). From this point of view, partitioning is analogous to preconditioning symmetric systems \( Ax = b \) via the splitting \( A = M - N \), in which the only extra work required is the solution of a subproblem \( M z_i = r_i \) (e.g. Concus, Golub and O'Leary [5], Chandra [3]).
A trivial example of partitioning arises whenever the parameters being estimated include a constant term (the mean). In this case, $A = \begin{bmatrix} e & C \end{bmatrix}$ where $e$ is a column of 1's. Writing $x = \begin{bmatrix} y \\ z \end{bmatrix}$, we would then use $T = (I - \frac{1}{m} ee^T)$ in (9.1), solve for $z$ and then estimate the mean using $\hat{u} = e^T(b - Cz)/m$. This is nothing more than the well known practice of "subtracting off the mean" (note that $Tb = b - \overline{u}e$, where $\overline{u} = e^Tb/m$) and it remains good practice in the present context.

A very general development of partitioning in a recursive sense has been given by G.N. Wilkinson [26]. From one point of view this covers the situation where the first partition $B$ can itself be partitioned.

9.2 Partitioning by rows

A simpler situation occurs when the matrix $A$ has the form

$$A = \begin{bmatrix} B \\ C \end{bmatrix}$$

where $B$ is nonsingular and is such that systems of equations involving $B$ and $B^T$ can be solved efficiently. An algorithm called PARTCG was developed by Chen [4] for this situation, and he reports excellent results on a large practical problem for which $B$ was triangular.

It is worth noting that algorithm PARTCG can be derived by transforming the normal equations $(B^T B + C^T C)x = [B^T C^T]b$ to obtain the equivalent system

$$(I + B^{-T} C^T) b = [I \quad B^{-T} C^T]b$$

(where $Bx = y$) and then applying the symmetric conjugate-gradient algorithm. From this point of view it would appear preferable to retain the least-squares formulation and solve the problem

$$\min \| \begin{bmatrix} I \\ Cb^{-1} \end{bmatrix} y - b \|$$

by an algorithm such as CGS or LCRG.
This approach is a particular case of preconditioning. It is discussed by Björck in his comprehensive survey of direct and iterative methods for sparse least squares [1]. The first work along these lines appears to be that of Läuchli [14]. In general we may expect the method to be successful only if $B$ is well-conditioned, or if $\|CB^{-1}\|$ is of moderate size.

9.3 Scaling and preconditioning

The simplest form of preconditioning is column scaling, in which the original problem is replaced by

$$\min \|ADy - b\|, \quad x = Dy$$

(9.3)

for some diagonal matrix $D$. Normally it must be left to the user to compute $D$ from his knowledge of $A$ and to use it appropriately in his matrix-product subroutines. In the absence of better information the results of van der Sluis [24] indicate that each column of $AD$ should have the same Euclidean length, say 1. Thus the $j$-th diagonal of $D$ is usually

$$\delta_j = 1 / (a_j^T a_j)^{1/2}$$

although this should not be at the expense of exaggerating the influence of inaccurate data. A convenient method for excluding any unwanted columns of $A$ is to set the corresponding $\delta_j = 0$.

Note that the original matrix $A$ often contains a high percentage of unit coefficients. In such cases a user should treat $A$ and $D$ as separate operators when implementing the matrix-vector products, since over a series of 500 or 1000 iterations the computation of $w = Dv, p = Aw$ can be substantially cheaper than scaling the data explicitly and using $p = (AD)v$.

By analogy with (9.3), a more general approach to preconditioning is to employ some stable factorization of $A$, say
52

\[ A = LU = \begin{bmatrix} \end{bmatrix} \]

in which the left-hand factor \( L \) has better condition than \( A \), and the right-hand factor \( U \) is nonsingular. We assume that \( U \) will be a sparse matrix and that systems of equations involving \( U \) and \( U^T \) can be solved efficiently.

There are now two equivalent but distinct possibilities:

\[
\begin{align*}
\min_{L} & \| Ly - b \|, \quad \text{solve } UX = Y; \quad (9.4a) \\
\min_{A} & \| AU^{-1} y - b \|, \quad \text{solve } UX = Y. \quad (9.4b)
\end{align*}
\]

The advantages of each depend on the origin of \( L \) and \( U \). In some cases \( L \) may have the same sparsity as \( A \), thus favoring (9.4a). A nontrivial example of this is given in section 10.2. Otherwise (9.4b) may be preferred since it does not require \( L \) to be stored.

A promising approach towards automating this form of preconditioning is to use Gaussian elimination with row and column interchanges to produce a conventional sparse \( LU \) factorization with \( L \) trapezoidal and \( U \) triangular:

\[ P_1 A P_2 = LU = \begin{bmatrix} \end{bmatrix} \]

For convenience we shall ignore the permutation matrices \( P_1 \) and \( P_2 \), but the vital point is that they can be chosen so that \( L \) and \( U \) remain quite sparse and that either \( L \) or \( U \) can be kept reasonably well-conditioned. The use of an \( LU \) factorization for (small, dense) least-squares problems was first suggested by Peters and Wilkinson [22] with \( \text{cond}(L) \) being kept small so that the normal equations \( L^T Ly = L^T b \) corresponding to (9.4a) could be solved without serious numerical difficulty. In the present context it is again \( \text{cond}(L) \) that must be controlled, but this time the principal motive
is to ensure reasonably rapid convergence when a conjugate-gradient
algorithm is applied to (9.4).

The use of a sparse LU factorization for accelerating the convergence
of CGLS was first suggested by Björck in [1]. We give some promising
experimental results in section 10. Another form of automatic precondition-
ing has been given by Björck and Elfving [2] in their algorithm CGPCNE.
This uses the SSOR operator obtained from $A^TA$ and has significant advantages
in terms of simplicity and efficiency. (For example, the sparse factorization
of $A^TA$ is performed implicitly and is therefore not subject to the
usual problems associated with fill-in.) A future comparison of LU- and
SSOR-preconditioning will be of great interest. Note that both methods
require explicit access to the rows or columns of $A$ (thus $A$ cannot be
represented in product form), but this will not often be a restriction.

9.4 Linear constraints

Equality-constrained problems of the form

$$\min \|Ax - b\| \quad \text{subject to} \quad Cx = d \tag{9.5}$$

arise in many applications. Without loss of generality we may assume that
$C$ has full row rank. The combined matrix $\begin{bmatrix} A \\ C \end{bmatrix}$ will usually have full column
rank.

If a statistical model happens to be "overspecified", i.e. the matrix
$A$ is known to contain column dependencies, the user will normally wish to
impose some simple restrictions which do not increase the residual norm
but do remove ambiguity from the solution. (For example, a particular sub-
set of the parameters $x$ may be required to sum to zero.) Thus with appro-
priate $C$ and $d$ it will be known in advance that

$$\min \|Ax - b\| = \min \left\| \begin{bmatrix} A \\ C \end{bmatrix} x - \begin{bmatrix} b \\ d \end{bmatrix} \right\|,$$
and the constraints will be satisfied at the solution if they are simply included as additional rows of $A$ and $b$. This procedure is easy to implement, but it does increase the size of the least-squares problem and the approximations $x_k$ will not necessarily satisfy $Cx_k = d$ throughout the iterations.

More generally (whether $A$ is singular or not), a factorization of the constraint matrix may be used to solve (9.5). For example, let

$$QC^T = \begin{bmatrix} R \\ 0 \end{bmatrix}, \quad Q^T = [Y \quad Z],$$

$$R^T y = d, \quad c = b - Ay,$$

where $Q$ is orthogonal and $R$ is upper triangular. The unconstrained problem

$$\min \|AZa - c\|$$

(9.6)

can be solved by a conjugate-gradient method, and the solution to the original problem is then $x = Q^T \begin{bmatrix} y \\ z \end{bmatrix}$. The operator $Az$ in (9.6) should not be formed explicitly, since $Q$ will normally be computed as a product of elementary transformations. Thus $Z$ should be used in the form $Z = Q^T \begin{bmatrix} 0 \\ J \end{bmatrix}$.

Similarly, $c = b - AQ^T \begin{bmatrix} y \\ 0 \end{bmatrix}$.

This procedure reduces the dimension of the least-squares problem to be solved, and any iterate $z_k$ will give a point $x_k = Q^T \begin{bmatrix} y \\ z_k \end{bmatrix}$ that satisfies $Cx_k = d$. If the constraints are nearly dependent, the solution of $R^T y = d$ is ill-conditioned and this will normally be revealed by growth in the vectors $y$ and $c$. However, the presence of constraints should not impair the convergence of a conjugate-gradient method on (9.6), since with $Q$ orthogonal it is easily shown that $\text{cond}(AZ) \leq \text{cond}(A)$ regardless of $\text{cond}(C)$.
If $C$ is large and sparse it would be more efficient to compute a stable triangular factorization of the form

$$LC^T = \begin{bmatrix} U \\ 0 \end{bmatrix}$$

using Gaussian elimination with row and column interchanges. The above procedure may then be applied with $L$ and $U$ in place of $Q$ and $R$. In this case we have $Z = L^T[O\ T]$ and the bound on the condition of the reduced problem is weakened to $\text{cond}(AZ) \leq \text{cond}(A) \text{cond}(Z) \leq \text{cond}(A) \text{cond}(L)$. This may not be of serious consequence since in practice the interchange strategy can usually keep $\text{cond}(L)$ moderate while maintaining sparsity in $L$ and $U$. 
10. AN APPLICATION IN GEOPHYSICS

The Fortran implementation of LSQR has been applied recently to some least-squares problems arising from the analysis of gravity-meter observations (Woodward [28]). The formulation of the underlying model is similar to that described by Reilly [23]. (The purpose is to estimate the mean value of gravity at each of a series of stations and to determine a set of instrumental parameters for each gravity meter, including drift rate and a correction to the calibration.)

We have taken one such set of observations to obtain a representative test problem

\[ \min \| Ax - b \|, \quad A = A_0 D \] (10.1)

in which \( A_0 \) is 1033 by 434 with each row containing 5 nonzero coefficients, the first two being unity. The diagonal matrix \( D \) was chosen to normalize the columns of \( A_0 \) and to exclude some containing insufficient data. Ignoring excluded columns, \( A \) is effectively 1033 by 320 and of full rank. Since \( \| b \| = 6000 \) and \( \| r \| = \| b - Ax \| = 0.75 \) for the solution \( x \), the system being solved is almost compatible.

The times quoted below are seconds of processor time using unoptimized Fortran code on a Burroughs B6700. All condition number estimates are those obtained by LSQR. The stopping rule S2 was used with ATOL = 10^{-8} (see section 6). Thus for a given operator \( A \) (which in some cases was \( L \) or \( AU^{-1} \)) a solution was accepted when the estimate of \( \| A^T r_k \| / (\| A \| \| r_k \|) \) decreased below 10^{-8}.

10.1 Original formulation

Even with column scaling as indicated in (10.1), the rate of convergence of LSQR proved to be very slow. Figure 5 shows a plot of \( \log_e \| r_k \| \) for the
first 600 iterations, and the steady growth of the estimate of \(\text{cond}(A)\).
After 1600 iterations the run was terminated with \(\|r_k\|\) and \(\|A^Tr_k\|/(\|A\|\|r_k\|)\)
still as large as 0.92 and \(10^{-4}\) respectively. (Thus \(\|r_k\|\) had not yet entered
the usual long flat "tail".) It appeared that \(\text{cond}(A) \approx 10^5\) and that well
over 2000 iterations would have been required to reach a satisfactory
solution.

As an aside, this was the longest run performed and therefore best
illustrates the accuracy of the norm estimates in section 5. After 1600
iterations the estimates of \(\|r_k\|\), \(\|A^Tr_k\|\) and \(\|x_k\|\) were correct to eight,
five, and eight digits respectively, a very satisfactory result. The value
\(\|B_k\|_F = 56\) over-estimated \(\|A\|_F = \sqrt{320} = 18\) by a noticeable amount in spite
of the bound (5.8). This is typical and in fact desirable, given the
manner in which \(\|A\|\) is used in the three stopping criteria of section 6.

10.2 Preconditioning with \(LU\) factors

In order to test the preconditioning in (9.4), a sparse \(LU\) factorization
of \(A\) was computed in the form

\[
\begin{bmatrix}
0 \\
A
\end{bmatrix}
= \begin{bmatrix}
U \\
0
\end{bmatrix}
\] (10.2)

using Gaussian elimination with row interchanges. The nonzeros in \(U\) were
stored compactly by rows, and the rows of \(A\) were eliminated one by one in
their natural order. Thus each \(L_j\) represents a stabilized elementary
transformation of the form

\[
\begin{bmatrix}
1 \\
\lambda_j & 1
\end{bmatrix}
\] or \[
\begin{bmatrix}
1 \\
1 & \lambda_j
\end{bmatrix}
\] (10.3)

suitably imbedded in the identity matrix of order \(m + n\), and several \(L_j\)
were required to eliminate each row in turn. This gives \( A = LU \) where \( L \) is stored in product form as the operator
\[
L = [0 \ I]L_1^{-1}L_2^{-1}L_3^{-1} \cdots \begin{bmatrix} I \\ 0 \end{bmatrix}.
\]

The multipliers were constrained to satisfy \( |\lambda_j| < 1/\tau \) for some pivot tolerance \( \tau \in (0, 1) \), thus allowing the usual compromise between sparsity and stability. Since \( L \) and \( U \) proved to be very sparse, \( \tau = 0.99 \) was chosen to minimize \( \text{cond}(L) \). The number of nonzeros in \( A \), \( L \) and \( U \) were about 5600, 6500 and 1500 respectively, and the factorization time of 10 seconds was negligible.

Some statistics for the three equivalent forms of problem (10.1) are summarized in Table 1. It is clear that \( LU \) preconditioning leads to a significant reduction in computation time for this particular problem, with only a moderate increase in storage requirements.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Condition (estimate)</th>
<th>Total iterations</th>
<th>Time per iteration</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A )</td>
<td>( \geq 10^5 )</td>
<td>( \geq 2000 )</td>
<td>1.0</td>
<td>( \geq 2000 )</td>
</tr>
<tr>
<td>( L )</td>
<td>3500</td>
<td>390</td>
<td>2.4</td>
<td>920</td>
</tr>
<tr>
<td>( AU^{-1} )</td>
<td>3500</td>
<td>400</td>
<td>1.3</td>
<td>520</td>
</tr>
</tbody>
</table>

Table 1. Application of LSQR to three equivalent problems:
\[
\min ||Ax - b||, \min ||Ly - b||, \min ||AU^{-1}y - b||.
\]

Note that in an \( LU \) factorization with row interchanges, \( \text{cond}(L) \) is not directly related to \( \text{cond}(A) \). (For example, \( L \) itself is independent of the column scale of \( A \).) In this case, \( \text{cond}(L) \approx 3500 \) is rather higher than we might have wished. This is partly due to the columns of unit coefficients in \( A_o \) which generate many multipliers that are "worst possible", i.e. \( |\lambda_j| = 1 \).
Better numerical properties could be expected if the elimination were performed column-wise rather than row-wise (cf. Wilkinson [27, pp. 162-166]). However, a more complex data structure would then be required for efficiency, and since many of the \( \lambda_j \) would still be of order 1 the reduction in \( \text{cond}(L) \) could prove to be only slight.

In this context it is normally \( \text{cond}(U) \) that reflects the condition of \( A \). There was no growth in our particular \( U \) (largest off-diagonal element \( = 0.7 \)) but the largest and smallest diagonals of \( U \) were 1.0 and 0.00021 respectively, giving the conservative estimate: \( \text{cond}(U) \approx 5000 \).

Perhaps the most surprising result is that the numerical performance of the operator \( AU^{-1} \) was not significantly different from that of \( L \). This means that the bound \( \text{cond}(AU^{-1}) < \text{cond}(A) \text{cond}(U) \approx 10^8 \) is fortunately not operative, even though \( A \) and \( U^{-1} \) are applied as separate operators. It is an open question how far this would remain true as \( \text{cond}(A) \) and \( \text{cond}(U) \) increase, but as long as it does remain true \( AU^{-1} \) is much less expensive to use.

The speed advantage of \( AU^{-1} \) over \( L \) is largely due to the unit coefficients in \( A \) and to the intrinsic simplicity of the relevant subroutines. This could be expected in other practical applications.

Figure 5 compares the decrease of \( \| p_k \| \) for the operators \( A \) and \( AU^{-1} \), the latter being indistinguishable from \( L \).

10.3 A problem-dependent transformation

Prior to implementation of the above preconditioning the slow convergence of LSQR on (10.1) necessitated a reconsideration of the problem formulation. Briefly, it was noted from the structure of \( A_0 \) that the
nonunit coefficients could be translated to give a more favorable
distribution around zero (Woodward [28]). Algebraically, the columns of $A_o$
ocite{woodward1982}
occurred in pairs of the form $[e \ a]$ in which the vectors $e$ and $a$
possessed the same sparsity pattern (with $e$ containing unit coefficients).

The shift $\tilde{a} = a - \mu e$ was applied to each such pair, where
$\mu = c^T a / e^T e$
was the mean of the nonzeros in $a$. This amounted to a column transformation

$$
[e \ \tilde{a}] = [e \ a] \begin{bmatrix} 1 & -\mu \\ & 1 \end{bmatrix}
$$

such that $e$ and $\tilde{a}$ were orthogonal. The usual column scaling then made each
pair of columns orthonormal.

The effect here is another preconditioning via the factorization
$A_o = \tilde{A}_o T$
in which $T$ is unit upper triangular and $\tilde{A}_o$ has the same sparsity
as $A_o$. Convergence of LSQR on the problem

$$
\min ||\tilde{x} - b||, \quad \tilde{A} = \tilde{A}_o \tilde{b}
$$

(10.4)

proved to be greatly accelerated, as shown in Figure 6. Compared to the
original problem (10.1), the reduction in $\text{cond}(A)$ was clearly important.
From the sharp drops in $||p_k||$ it seems that another contributing factor was
an increased repetition of singular values. Unfortunately a further
"automatic" preconditioning via $\tilde{A} = \bar{L} \bar{U}$ appeared to negate the latter
effect, as shown in Figure 6. The smooth decrease in $||p_k||$ and the long
"tail" before stopping criterion S2 was satisfied indicates that $\bar{L} = \bar{A} \bar{U}^{-1}$ has
its singular values spread rather uniformly throughout their range.

Some statistics for the transformed problem (10.4) are summarized in
Table 2. Again the operators $\bar{L}$ and $\bar{A} \bar{U}^{-1}$ have virtually the same numerical
properties.
Figure 5. Comparison of $\min \|Ax - b\|$ and $\min \|AU^{-1}y - b\|$ for problem (10.1).

1 = $\log_e \|r_k\|$ for operator $A$.
2 = $\log_e \|r_k\|$ for operator $AU^{-1}$.
3 = estimate of $\text{cond}(A)$.
4 = estimate of $\text{cond}(AU^{-1})$.

The numbers marked on 2 are those tested against ATOL in stopping criterion S2, i.e. estimates of $\|L^T r_k\| / (\|L\| \|r_k\|)$ where $L = AU^{-1}$.

Results for the operator $L$ are essentially the same as for $AU^{-1}$. 
Figure 6. Similar comparison for the transformed problem (10.2).

1 = \log \| r_k \| \quad \text{for operator } \bar{A}.
2 = \log \| r_k \| \quad \text{for operators } \bar{A}U^{-1}, \bar{L}.
3 = \text{estimate of } \text{cond}(\bar{A}).
4 = \text{estimate of } \text{cond}(\bar{A}U^{-1}) = \text{cond}(\bar{L}).

<table>
<thead>
<tr>
<th>Operator</th>
<th>Condition</th>
<th>Total iterations</th>
<th>Time per iteration</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>\bar{A}</td>
<td>3100</td>
<td>250</td>
<td>1.0</td>
<td>250</td>
</tr>
<tr>
<td>\bar{L}</td>
<td>3200</td>
<td>376</td>
<td>2.4</td>
<td>880</td>
</tr>
<tr>
<td>\bar{A}U^{-1}</td>
<td>3100</td>
<td>371</td>
<td>1.3</td>
<td>460</td>
</tr>
</tbody>
</table>

Table 2. Application of LSQR to three equivalent forms of problem (10.4).
10.4 Comparison with QR factorization

The transformed problem (10.4) was also solved directly using a sparse orthogonal factorization

\[
\begin{bmatrix}
Q & 0 \\
\overline{A}
\end{bmatrix} =
\begin{bmatrix}
R & 0 \\
0 & 0
\end{bmatrix},
\begin{bmatrix}
Q \\
b
\end{bmatrix} =
\begin{bmatrix}
\xi \\
d
\end{bmatrix},
\overline{A}\xi = c,
\]

where \( Q \) represents a product of plane rotations analogous to the row-wise elimination (10.2). The factorization was reasonably sparse: \( Q = 33000 \) plane rotations, \( R = 9000 \) nonzeros. It was implemented as efficiently as possible subject to processing the rows of \( \overline{A} \) in their natural order. Some economies could be achieved using different row orderings, e.g. Gentleman [8], but the factorization time of 380 seconds is representative of the great expense of QR- compared to LU-factorization. Computation of the standard error estimates (from \( R \) - see section 5.4) required a further 160 seconds.

Comparison with the operator \( \overline{A} \) in Table 2 shows that LSQR furnished an equally accurate \( \overline{x} \) and rather less accurate standard errors in about half the time required by the direct method. The storage requirements (and incidentally the paging activity in the B6700 virtual memory environment) were also substantially less. These advantages of the conjugate-gradient approach to least squares must become more apparent with increasing problem size, as long as the relevant matrix operator remains well conditioned.

10.5 A larger problem

Finally, to indicate the effect of problem size, we note some results for a similar example which contained approximately twice as many observations (and hence twice as much data defining \( A \)). Column means were subtracted as before, giving problem (10.4) with \( A \) 1850 by 875.
In solving this case, LSQR obtained the estimate \( \text{cond}(\tilde{A}) = 3200 \), essentially the same as for the smaller problem. About 500 iterations and 1000 seconds of processor time were required, a four-fold increase in total work.
11. SUMMARY

The aim has been to present the derivation of an algorithm and details of its implementation, along with sufficient experimental evidence to suggest that the algorithm compares favorably with other similar methods and that it can be relied upon to give satisfactory numerical solutions to problems of practical importance.

Reliable stopping criteria were regarded here as being essential to any iterative method for solving the problems $Ax = b$ and $\min \|Ax - b\|$. The criteria developed for LSQR may be useful for other solution methods. Estimates of $\|A\|$, $\text{cond}(A)$ and standard errors for $x$ have also been developed to provide useful information to the user at minimal cost.

Finally, some results obtained using a sparse $LU$ factorization of $A$ illustrate an effective method for accelerating convergence on ill-conditioned problems.

ACKNOWLEDGEMENTS

We wish to thank Dr Robert Davies of the DSIR Applied Mathematics Division for his assistance and advice on many aspects of this work. We are also grateful to Dr Derek Woodward of DSIR Geophysics Division for providing details of his analyses of gravity-meter observations, and to the Computer Services Centre at the Victoria University of Wellington (and to Burroughs Corporation) for providing the necessary computing facilities.
REFERENCES


APPENDIX A: FORTRAN PROGRAMS

Listings are given here of the following Fortran routines:

1. **LSQR** - the principal subroutine implementing the sparse least-squares algorithm.

2. Auxiliary routines required by LSQR.
   - **NORM1Z** - normalize a vector to unit length.
   - **SAXPY** - compute \( ax + y \) for scalar \( a \) and vectors \( x, y \).
   - **SNRM2** - compute the euclidean norm of a vector.
   - **ATIMES** - a sample routine for computing the product \( p = Av \).
   - **ATRANS** - a sample routine for computing the product \( p = A^T u \).

3. Routines for generating a test problem.
   - **HPROD** - apply a Householder transformation to a vector according to \( p = (I - 2x^Tx)v \), where \( \|x\| = 1 \).
   - **LSTP** - generate a least-squares test problem \( P(m,n,d,p) \) of the form described in section 8.1 of this paper.
   - **TEST** - request generation of a specific test problem and then solve it using LSQR.

4. A sample main program calling TEST.

These routines are written in the PFORT subset of ANSI Fortran (Ryder and Hall [2]). There are no machine-dependent constants. Conversion between single and double precision is accomplished by interchanging the following characters throughout:

- \( \text{ABS} \) \( \downarrow \downarrow \rightarrow \text{DABS} \)
- \( \text{REAL} \) \( \downarrow \downarrow \rightarrow \text{REAL*8 or DOUBLE PRECISION} \)
- \( \text{SQRT} \) \( \downarrow \downarrow \rightarrow \text{DSQRT} \)

The routines **SAXPY** and **SNRM2** correspond to two members of the BLAS collection (Lawson, Hanson, et al [1]). For improved efficiency the BLAS versions should be used when available. The appropriate double precision routines would then be **DAXPY** and **DNRM2**.


SUBROUTINE LSOQ( M,N,MAX,P,P,U,V,W,X,SE,
    1 ATOL,BTOL,CONLM,ITNLIM,NOUT,
    2 ISTOP,RNORM,ANORM,ACOND)

INTEGER M,N,MAX,ITNLIM,NOUT,ISTOP
REAL B(M),P(MAX),U(M),V(N),W(N),X(N),SE(N),
    1 ATOL,BTOL,CONLM,RNORM,ANORM,ACOND

LSOR FINDS A SOLUTION X TO THE FOLLOWING PROBLEMS...
(UNSYMMETRIC EQUATIONS) SOLVE A.X = B,
(LINEAR LEAST SQUARES) MINIMIZE NORM(R) WHERE R = B - A.X,
IN WHICH A IS A REAL MATRIX OF DIMENSIONS M BY N AND B IS A
GIVEN M-VECTOR. NORMALLY RANK(A) SHOULD BE N, WITH M >= N,
BUT THIS IS NOT ESSENTIAL (E.G. SOME OF THE ROWS AND COLUMNS OF
A MAY BE ZERO).

THE MATRIX A IS INTENDED TO BE LARGE AND SPARSE. IT IS ACCESSED
BY MEANS OF TWO SUBROUTINE CALLS OF THE FORM

    CALL ATIMES( V,P,M,N )
    CALL ATTRANS( U,P,M,N )

WHICH MUST RETURN THE PRODUCTS P = A.V AND P = A(TRANSPOSE).U
FOR GIVEN INPUT VECTORS V AND U.

SUBROUTINES ATIMES AND ATTRANS ARE TO BE SUPPLIED BY THE USER.
THEY MUST NOT ALTER V OR U RESPECTIVELY. THE LENGTH OF THE
OUTPUT VECTOR P IS RESPECTIVELY M AND N, BUT IN EITHER CASE
THE ARRAY PARAMETER P MAY BE DECLARED TO HAVE LENGTH MAX(M,N).
(THE MAY BE USEFUL IF P IS USED FOR WORKSPACE BEFORE EXIT.)

NOTE. THE NUMBER OF ITERATIONS REQUIRED BY LSOR DEPENDS
CRITICALLY ON THE CONDITION NUMBER OF THE MATRIX A. POOR
SCALING OF THE COLUMNS OF A SHOULD BE AVOIDED. THIS CAN
USUALLY BE TAKEN CARE OF WHEN PROGRAMMING SUBROUTINES ATIMES
AND ATRANS. IN THE ABSENCE OF BETTER INFORMATION THE NONZERO
COLUMNS OF A SHOULD BE SCALING SO THAT THEY ALL HAVE THE SAME
EUCLIDEAN NORM (USUALLY 1.0).

PARAMETERS
-------
M INPUT THE NUMBER OF ROWS IN A.
N INPUT THE NUMBER OF COLUMNS IN A.
MAX INPUT MAX(M,N). USED ONLY TO SPECIFY THE LENGTH
OF P IN THE DIMENSION STATEMENT ABOVE.
B(M) INPUT THE RHS VECTOR OF LENGTH M.

P(MAX) OUTPUT RETURNS THE VECTOR A(TRANSPOSE) . R, WHERE R = B - A . X IS THE RESIDUAL VECTOR CORRESPONDING TO THE COMPUTED SOLUTION X.

U(M) OUTPUT RETURNS THE RESIDUAL VECTOR R = B - A . X.

V(N) WORKSPACE

W(N) WORKSPACE

X(N) OUTPUT RETURNS THE COMPUTED SOLUTION X.

SE(N) OUTPUT RETURNS STANDARD ERROR ESTIMATES FOR THE COMPONENTS OF X.

NOTE. THE FOLLOWING DISCUSSION OF TOLERANCES IS IN TERMS OF A QUANTITY EPS WHICH STANDS FOR THE RELATIVE PRECISION OF FLOATING-POINT ARITHMETIC ON THE MACHINE BEING USED. TYPICAL VALUES ARE

BURROUGHS B6700 SINGLE PRECISION, EPS = 1 . 0E-11
CDC 6600, 7600 SINGLE PRECISION, EPS = 1 . 0E-14
IBM 360, 370 SINGLE PRECISION, EPS = 1 . 0E-6
DOUBLE PRECISION, EPS = 1 . 0D-16
UNIVAC 1100 SERIES SINGLE PRECISION, EPS = 1 . 0E-8
DOUBLE PRECISION, EPS = 1 . 0D-18

(ALL VALUES APPROXIMATE).

ATOL INPUT AN ESTIMATE OF THE RELATIVE ACCURACY OF THE DATA DEFINING THE MATRIX A. ATOL WILL NORMALLY BE IN THE RANGE EPS TO SQRT(EPS).
SUGGESTED VALUE -- ATOL = 1000*EPS

BTOL INPUT AN ESTIMATE OF THE RELATIVE ACCURACY OF THE DATA DEFINING THE RHS VECTOR B. BTOL WILL NORMALLY BE IN THE RANGE EPS TO SQRT(EPS).
SUGGESTED VALUE -- BTOL = 1000*EPS

CONLIM INPUT AN UPPER LIMIT ON COND(A), THE APPARENT CONDITION NUMBER OF THE MATRIX A (IGNORING KNOWN SINGULARITIES). ITERATIONS WILL TERMINATE IF A COMPUTED ESTIMATE OF COND(A) EXCEEDS CONLIM. THIS IS INTENDED TO PREVENT CERTAIN SMALL OR ZERO SINGULAR VALUES OF A FROM COMING INTO EFFECT AND CAUSING UNWANTED GROWTH IN THE COMPUTED X. HFNCF IT MAY ASSIST IN REGULARIZING ILL-CONDITIONED SYSTEMS.
CONLIM will normally be in the range 1000 to 1/EPS.
Suggested value --
CONLIM = 1/(1000*EPS) for compatible systems,
CONLIM = 1/(10*SQRT(EPS)) for least squares.

NOTE. IF THE USER IS NOT CONCERNED ABOUT THE PARAMETERS
ATOL, BTOL AND CONLIM, ANY OR ALL OF THEM MAY BE SET TO ZERO.

ITNLIM INPUT
AN UPPER LIMIT ON THE NUMBER OF ITERATIONS.
Suggested value --
ITNLIM = N/2 for well-conditioned systems,
ITNLIM = 4*N otherwise.

NOUT INPUT
LINEPRINTER UNIT NUMBER. IF POSITIVE,
A SUMMARY WILL BE PRINTED ON UNIT NOUT.

ISTOP OUTPUT
AN INTEGER GIVING THE REASON FOR TERMINATION...

0
X = 0 IS THE EXACT SOLUTION.
NO ITERATIONS WERE PERFORMED.

1
THE EQUATIONS A.X = B ARE PROBABLY COMPATIBLE. NORM(R) IS SUFFICIENTLY SMALL
GIVEN THE VALUES OF ATOL AND BTOL.

2
THE SYSTEM A.X = B IS PROBABLY NOT COMPATIBLE. A LEAST-SQUARES SOLUTION HAS BEEN OBTAINED, FOR WHICH NORM(A,R) IS SUFFICIENTLY SMALL GIVEN THE VALUE OF ATOL.

3
THE SYSTEM IS MORE ILL-CONDITIONED THAN EXPECTED. AN ESTIMATE OF COND(A) HAS EXCEEDED CONLIM.

4
THE ITERATION LIMIT ITNLIM WAS REACHED.

5
THE EQUATIONS A.X = B ARE PROBABLY COMPATIBLE. NORM(R) IS AS SMALL AS SEEMS REASONABLE ON THIS MACHINE.

6
THE SYSTEM A.X = B IS PROBABLY NOT COMPATIBLE. A LEAST-SQUARES SOLUTION HAS BEEN OBTAINED, FOR WHICH NORM(A,R) IS AS SMALL AS SEEMS REASONABLE ON THIS MACHINE.

7
COND(A) SEEMS TO BE SO LARGE THAT THERE IS NOT MUCH POINT IN DOING FURTHER ITERATIONS, GIVEN THE PRECISION OF THIS MACHINE.

RNORM OUTPUT
NORM(R) = SORT(R.R), THE NORM OF THE FINAL RESIDUAL VECTOR R = B - A.X.
ANORM OUTPUT
AN ESTIMATE OF THE FROBENIUS NORM OF A.
THIS IS THE SQUARE-ROOT OF THE SUM OF SQUARES
OF THE ELEMENTS OF A. IF THE COLUMNS OF A
HAVE ALL BEEN SCALED TO HAVE LENGTH 1.0,
ANORM SHOULD INCREASE TO ROUGHLY SORT(N).
A RADICALLY DIFFERENT VALUE FOR ANORM MAY
INDICATE AN ERROR IN ONE OF THE SUBROUTINES
ATIMES OR ATRANS.

ACOND OUTPUT
AN ESTIMATE OF \text{COND}(A), THE CONDITION
NUMBER OF A. A VERY HIGH VALUE OF ACOND
MAY AGAIN INDICATE AN ERROR IN SUBROUTINES
ATIMES OR ATRANS.

TO CHANGE PRECISION
-------------------
IF SUBSTITUTE BLAS Routines ARE IN USE, ALTER THE WORDS
ABS, REAL, SORT
THROUGHOUT ROUTINES LSQR, NORMLZ, SAXPY, SNRM2.

IF AUTHENTIC BLAS Routines ARE IN USE, ALTER THE WORDS
ABS, REAL, SAXPY, SNRM2, SORT
THROUGHOUT ROUTINES LSQR, NORMLZ.

REFERENCE
A BIDIAGONALIZATION ALGORITHM FOR SPARSE
LINEAR EQUATIONS AND LEAST-SQUARES PROBLEMS,
TECHNICAL REPORT SOL 78-19, DEPARTMENT OF
OPERATIONS RESEARCH, STANFORD UNIVERSITY, 1978

AUTHORS
C.C. PAIGE
MCGILL UNIVERSITY, CANADA
M.A. SAUNDERS
DSIR, NEW ZEALAND

LSQR. THIS VERSION DATED 2 OCTOBER 1978.

SUBROUTINES AND FUNCTIONS
USER
ATIMES, ATRANS
LSQR
NORMLZ
BLAS
SAXPY
FORTRAN
ABS, MOD, SQRT
FUNCTIONS AND LOCAL VARIABLES

INTEGER I, ITN, MOD, NCONV, NSTOP
REAL ABS, ALFA, ARNORM, BBNORM, BETA, BNORM,
1 CS, CS2, CTOL, DDNORM, DELTA, GAMMA, GBAR,
2 ONE, PHI, RBAR, RHO, RHS, RTOL,
3 SINES, SN, SN2, SORT, T, TEST1, TEST2, TEST3,
4 THETA, T1, T2, T3, XNORM, XXNORM, Z, ZBAR, ZERO

INITIALIZE

ZERO = 0
ONE = 1
IF (NOUT .GT. 0) WRITE(NOUT, 1000) M, N, ATOL, BTOL, CONLIM, ITN
CALL NORMLZ(B, U, M, BETA)
CALL ATTRANS(U, P, M, N)
CALL NORMLZ(P, V, N, ALFA)

DO 20 I = 1, N
   W(I) = V(I)
   X(I) = ZERO
   SE(I) = ZERO
CONTINUE

SINES = ONE
BBNORM = ZERO
DDNORM = ZERO
XXNORM = ZERO
Z = ZERO
CS2 = -ONE
SN2 = ZERO
CTOL = ZERO

IF (CONLIM .GT. ZERO) CTOL = ONE/CONLIM
ITN = 0
ISTOP = 0
NSTOP = 0

ANORM = ZERO
ACOND = ZERO
RBAR = ALFA
BNORM = BETA
RNORM = BETA
XNORM = ZERO
ARNORM = ALFA*BETA

IF (NOUT .GT. 0) WRITE(NOUT, 1200)
IF (NOUT .GT. 0) WRITE(NOUT, 1400) ITN, X(1), RNORM, ARNORM
IF (NOUT .GT. 0) WRITE(NOUT, 1500)
IF (ARNORM .LE. ZERO) GO TO 700
MAIN ITERATION LOOP

PERFORM NEXT STEP OF THE BIDIAgonALIZATION
TO OBTAIN NEW BETA, U, ALFA, V

100 ITN = ITN+1
CALL ATimes( V,P,M,N )
CALL SAXPY( M,(-ALFA),U,1,P,1 )
CALL NORMLZ( P,U,M,BETA )

BBNORM = BBNORM + ALFA**2 + BETA**2

CALL ATRANS( U,P,M,N )
CALL SAXPY( N,(-BETA),V,1,P,1 )
CALL NORMLZ( P,V,N,ALFA )

COMPUTE NEXT PLANE ROTATION FOR THE OR-FACTORIZATION
OF THE LOWER BIDIAgonAL MATRIX B (I.E. Q.B = R)

RHO = SORT(RBAR**2 + BETA**2)
CS = RBAR/RHO
SN = BETA/RHO
THETA = SN*ALFA
RBAR = -CS*ALFA
PHI = CS*RNORM
RNORM = SN*RNORM
ARNORM = ABS(RBAR)*RNORM
SINES = SN*SINES

UPDATE X AND THE STANDARD ERROR ESTIMATES

T1 = PHI/RHO
T2 = -THETA/RHO
T3 = ONE/RHO
CALL SAXPY( N,T1,W,1,X,1 )

DO 350 I = 1, N
  T = W(I)
  W(I) = T*T2 + V(I)
  T = (T*T3)**2
  SE(I) = T + SE(I)
  DDNORM = T + DDNORM
350 CONTINUE
ESTIMATE NORM(X) USING AN LDL-FACTORIZATION
OF THE UPPER BIDIAGONAL MATRIX R (I.E. R.O = L)

DELTA = SN2*RHO
GBAR = -CS2*RHO
RHS = PHI - DELTA*Z
ZBAR = RHS/GBAR
XNORM = SORT(XXNORM + ZBAR**2)
GAMMA = SORT(GBAR**2 + THETA**2)
CS2 = GBAR/GAMMA
SN2 = THETA/GAMMA
Z = RHS/GAMMA
XXNORM = XXNORM + Z**2

TEST FOR CONVERGENCE

ANORM = SORT(BRNORM)
ACOND = ANORM*SORT(DDNORM)
RTOL = BTOL + ATOL*ANORM*XNORM/RNORM
TEST1 = SINES
TEST2 = ARNORM/(ANORM*RNORM)
TEST3 = ONE/ACOND

THE FOLLOWING THREE TESTS ARE INDEPENDENT OF THE TOLERANCES
ATOL, BTOL AND CONLIM. THEY ARE INTENDED TO GUARD AGAINST
ACCIDENTAL SETTING OF THOSE PARAMETERS TO EXTREME VALUES.
THEY ARE EQUIVALENT TO THE NORMAL TESTS USING THE VALUES
ATOL = EPS, BTOL = EPS, CONLIM = 1/EPS.

T1 = ONE + TEST1
T2 = ONE + TEST2
T3 = ONE + TEST3
IF (T3 .LE. ONE) ISTOP = 7
IF (T2 .LE. ONE) ISTOP = 6
IF (T1 .LE. ONE) ISTOP = 5

ALLOW FOR TOLERANCES SET BY USER

IF (ITN .GE. ITNLIM) ISTOP = 4
IF (TEST3 .LE. CTOL) ISTOP = 3
IF (TEST2 .LE. ATOL) ISTOP = 2
IF (TEST1 .LE. RTOL) ISTOP = 1
SEE IF IT IS TIME TO PRINT SOMETHING

IF (NOUT .LE. 0) GO TO 600
IF (M.LE.40 .OR. N.LE.40) GO TO 400
IF (ITN .LE. 10) GO TO 400
IF (ITN .GE. ITNLIM-10) GO TO 400
IF (MOD(ITN,10) .EQ. 0) GO TO 400
IF (TEST3 .LE. 2.0*CTOL) GO TO 400
IF (TEST2 .LE. 10.0*ATOL) GO TO 400
IF (TEST1 .LE. 10.0*RTOL) GO TO 400
GO TO 600

PRINT A LINE FOR THIS ITERATION

400 CONTINUE
WRITE(NOUT, 1400) ITN,X(1),RNORM,ARNORM,TEST1,TEST2,ANORM,ACOND
IF (MOD(ITN,10) .EQ. 0) WRITE(NOUT, 1500)

STOP IF POSSIBLE.
THE CONVERGENCE CRITERIA ARE REQUIRED TO BE MET ON NCONV
CONSECUTIVE ITERATIONS, WHERE NCONV IS SET BELOW.
SUGGESTED VALUE -- NCONV = 1, 2 OR 3

600 IF (ISTOP .EQ. 0) NSTOP = 0
IF (ISTOP .EQ. 0) GO TO 100
NCONV = 1
NSTOP = NSTOP+1
IF (NSTOP .LT. NCONV .AND. ITN .LT. ITNLIM) ISTOP = 0
IF (ISTOP .EQ. 0) GO TO 100

END OF ITERATION LOOP
C PRINT SUMMARY OF FINAL SOLUTION STATUS

700 CONTINUE
IF (NOUT .LE. 0) GO TO 800
WRITE(NOUT, 1900) ITN, ISTOP
IF (ISTOP .EQ. 0) WRITE(NOUT, 2000)
IF (ISTOP .EQ. 1) WRITE(NOUT, 2100)
IF (ISTOP .EQ. 2) WRITE(NOUT, 2200)
IF (ISTOP .EQ. 3) WRITE(NOUT, 2300)
IF (ISTOP .EQ. 4) WRITE(NOUT, 2400)
IF (ISTOP .EQ. 5) WRITE(NOUT, 2500)
IF (ISTOP .EQ. 6) WRITE(NOUT, 2600)
IF (ISTOP .EQ. 7) WRITE(NOUT, 2700)

C COMPUTE FINAL RESIDUAL R = B - A.X, AND A(TRANPOSE).R

800 T1 = RNORM
T2 = ARNORM
T3 = XNORM
CALL ATIMES( X, P, M, N )

DO 900 I = 1, M
   U(I) = B(I) - P(I)
900 CONTINUE

CALL NORMLZ( U, P, M, RNORM )
CALL ATTRANS( U, P, M, N )
CALL NORMLZ( P, W, N, ARNORM )
CALL NORMLZ( X, W, N, XNORM )
IF (NOUT .GT. 0) WRITE(NOUT, 3000) BNORM, ARNORM, ACOND, T1, T2, T3,
1   RNORM, ARNORM, XNORM

C FINISH OFF THE STANDARD ERROR ESTIMATES

T = ONE
IF (M .GT. N) T = M-N
T = RNORM/SQRT(T)

DO 950 I = 1, N
   SE(I) = T*SORT(SE(I))
950 CONTINUE

RETURN
C

1000 FORMAT(
    1  // 25X, 46HLSQR -- LEAST-SQUARES SOLUTION OF A.X = B
    2  // 25X, 46H(I.E. MINIMIZE NORM(R), WHERE R = B - A.X )
    3  // 25X, 29HTHE MATRIX A HAS DIMENSIONS, 16, 5H BY, 16
    4  // 25X, 8HATOL =, 1PE10.2, 10X, 8HBTOL =, 1PE10.2
    5  // 25X, 8HCONLIM =, 1PE10.2, 10X, 8HITNLIM =, 110)
1200 FORMAT(/ 3X, 3HITN, 9X, 4HX(1), 14X, 7HNORM(R), 8X, 9HNORM(A.R),
       1 3X, 23HCOMPATIBLE INCOMPATIBLE, 4X, 7HNORM(A), 4X, 7HCOND(A) /)
1400 FORMAT(I6, 1PE20.10, 1PE19.10, 1PE12.3, 1P2E13.3, 1P2E11.2)
1500 FORMAT(1X)
1900 FORMAT(
    1  // 19H NO. OF ITERATIONS , I10
    2  // 19H STOPPING CONDITION, I10)
2000 FORMAT(1H+, 34X, 44H(THE EXACT SOLUTION IS X = 0))
2100 FORMAT(1H+, 34X, 44H(NORM(R) IS SMALL ENOUGH, GIVEN ATOL, BTOL))
2200 FORMAT(1H+, 34X, 44H(NORM(A.R) IS SMALL ENOUGH, GIVEN ATOL))
2300 FORMAT(1H+, 34X, 44H(COND(A) HAS EXCEEDED CONLIM))
2400 FORMAT(1H+, 34X, 44H(ITERATION LIMIT REACHED))
2500 FORMAT(1H+, 34X, 44H(NORM(R) IS SMALL ENOUGH FOR THIS MACHINE))
2600 FORMAT(1H+, 34X, 44H(NORM(A.R) IS SMALL ENOUGH FOR THIS MACHINE))
2700 FORMAT(1H+, 34X, 44H(COND(A) IS TOO LARGE FOR THIS MACHINE))
3000 FORMAT(
    1  // 19H NORM(B) TRUE, 1PE20.10,
    2  5X, 19H NORM(A) ESTIMATE, 1PE15.5,
    3  5X, 19H COND(A) ESTIMATE, 1PE15.5
    4  // 19H NORM(R) ESTIMATE, 1PE20.10,
    5  5X, 19H NORM(A.R) ESTIMATE, 1PE15.5,
    6  5X, 19H NORM(X) ESTIMATE, 1PE15.7
    7  // 19H TRUE, 1PE20.10,
    8  5X, 19H TRUE, 1PE15.5,
    9  5X, 19H TRUE, 1PE15.7)

C END OF LSQR
END
SUBROUTINE NORMLZ( P,V,N,BETA )
INTEGER N
REAL P(N),V(N),BETA

NORMLZ IS REQUIRED BY SUBROUTINE LSQR.
IT NORMALIZES THE VECTOR P AND RETURNS THE RESULT IN V.
ON RETURN, BETA = NORM(P) AND BETA*V = P.

FUNCTIONS

BLAS SNRM2

INTEGER I
REAL ONE,SNRM2,T,ZERO

ZERO = 0
ONE = 1
BETA = SNRM2( N,P,1 )
IF (BETA .LE. ZERO) RETURN
T = ONE/BETA

DO 20 I = 1, N
  V(I) = P(I)*T
20 CONTINUE
RETURN

END OF NORMLZ
END

SUBROUTINE SAXPY( N,A,X,INCX,Y,INCY )
REAL A,X(N),Y(N)

THIS MAY BE REPLACED BY THE CORRESPONDING BLAS ROUTINE.
The FOLLOWING IS A SIMPLE VERSION FOR USE WITH LSQR.

DO 10 I = 1, N
  Y(I) = A*X(I) + Y(I)
10 CONTINUE
RETURN

END

REAL FUNCTION SNRM2( N,X,INCX )
REAL X(N)

THIS MAY BE REPLACED BY THE CORRESPONDING BLAS ROUTINE.
The FOLLOWING IS A SIMPLE VERSION FOR USE WITH LSQR.

SNRM2 = 0
DO 10 I = 1, N
  SNRM2 = X(I)**2 + SNRM2
10 CONTINUE
SNRM2 = SQRT(SNRM2)
RETURN

END
SUBROUTINE ATIMES( V,P,M,N )
INTEGER M,N
REAL V(N),P(M)

C COMPUTE P = A.V FOR TEST MATRIX A

INTEGER I,N1
REAL ZERO
REAL D,R,XTRUE,Y,Z
COMMON /LSCOMM/ D(100),R(100),XTRUE(100),Y(100),Z(100)

CALL HPROD( V,P,Z,N )
DO 100 I=1,N
   P(I) = D(I)*P(I)
100 CONTINUE

IF (M .LE. N) GO TO 500
ZERO = 0
N1 = N+1
DO 200 I=N1,M
   P(I) = ZERO
200 CONTINUE

500 CALL HPROD( P,P,Y,M )
RETURN

C END OF ATIMES
END

SUBROUTINE ATRANS( U,P,M,N )
INTEGER M,N
REAL U(M),P(N)

C COMPUTE P = A(TRANSPOSE).U FOR TEST MATRIX A

INTEGER I
REAL D,R,XTRUE,Y,Z
COMMON /LSCOMM/ D(100),R(100),XTRUE(100),Y(100),Z(100)

CALL HPROD( U,P,Y,M )
DO 100 I=1,N
   P(I) = D(I)*P(I)
100 CONTINUE

CALL HPROD( P,P,Z,N )
RETURN

C END OF ATRANS
END
SUBROUTINE HPDO( V, P, Z, N )
INTEGER N
REAL P(N), V(N), Z(N)

C APPLY HOUSEHOLDER TRANSFORMATION TO GET P = ( I - 2 Z Z' ) . V

INTEGER I
REAL S

S = 0
DO 100 I = 1, N
    S = Z(I) * V(I) + S
100 CONTINUE

S = S + S
DO 200 I = 1, N
    P(I) = V(I) - S * Z(I)
200 CONTINUE

RETURN

C END OF HPDO
END

SUBROUTINE LSTP( M, N, NDUPLC, NPOWER, B, ACOND, RNORM )
INTEGER M, N, NDUPLC, NPOWER
REAL B(M), ACOND, RNORM

C GENERATE A SPARSE LEAST-SQUARES TEST PROBLEM, A . X = B,
C WHERE A = Y . D . Z IS M BY N, D IS DIAGONAL, AND
C Y AND Z ARE HOUSEHOLDER TRANSFORMATIONS.

FUNCTIONS AND SUBROUTINES

TESTPROB ATIMES, HPDO
LSQR NORMLZ
FORTRAN COS, FLOAT, SIN

INTEGER I, J, MN
REAL ALFA, BETA, COS, FLOAT, FOURPT, SIN, T
REAL D, R, XTRUE, Y, Z
COMMON /LSCOMM/ D(100), R(100), XTRUE(100), Y(100), Z(100)

---------------
MAKE TWO VECTORS OF NORM 1.0, FOR HOUSEHOLDER TRANSFORMATIONS
---------------

FOURPT = 4.0 * 3.141592
ALFA = FOURPT / FLOAT(M)
BETA = FOURPT / FLOAT(N)

DO 100 I = 1, M
    Y(I) = SIN(FLOAT(I) * ALFA)
100 CONTINUE

DO 200 I = 1, N
    Z(I) = COS(FLOAT(I) * BETA)
200 CONTINUE
CALL NORMLZ( Y,Y,M,ALFA )
CALL NORMLZ( Z,Z,N,BETA )

-----------------------------
SET SINGULAR VALUES FOR DIAGONAL MATRIX D
-----------------------------
DO 300 I=1,N
  J = (I-1+NDUPLC)/NDUPLC
  T = J*NDUPLC
  T = T/FLOAT(N)
  D(I) = T**NPOWER
300 CONTINUE

ACOND = D(N)/D(1)

-----------------------------
SET SOLUTION XTRUE
-----------------------------
DO 400 I=1,N
  XTRUE(I) = N-I
400 CONTINUE

-----------------------------
COMPUTE RHS B
-----------------------------
CALL ATIMES( XTRUE,B,M,N )
IF (M .LE. N) RETURN

-----------------------------
FOR LEAST SQUARES, ADD RESIDUAL R
-----------------------------
DO 500 I=1,N
  R(I) = 0.0
500 CONTINUE

T = 1
MN = M-N
DO 600 I=1,MN
  J = N+I
  R(J) = T*FLOAT(I)/FLOAT(M)
  T = -T
600 CONTINUE

CALL HPROD( R,R,Y,M )

DO 700 I=1,M
  B(I) = B(I) + R(I)
700 CONTINUE

CALL NORMLZ( R,R,M,RNORM )
RETURN

END OF LSTP
END
SUBROUTINE TEST( M,N,NDUPLC,NPOWER )
INTEGER M,N,NDUPLC,NPOWER

C EXAMPLE DRIVER ROUTINE FOR TESTING LSQR
C
INTEGER ISTOP,ITNLIM,J,NOUT
REAL B(100),P(100),U(100),V(100),
     W(100),X(100),SE(100),
     ATOL,BTOL,CONLIM,PNORM,ANORM,ACOND

C GENERATE SPECIFIED TEST PROBLEM
C
CALL LSTP( M,N,NDUPLC,NPOWER,B,ACOND,RNORM )

NOUT = 6
WRITE(NOUT, 1000) NDUPLC,NPOWER,ACOND,RNORM

C SET TOLERANCES FOR LSQR
C
ATOL = 1.0E-10
BTOL = ATOL
CONLIM = 1.0E+10
IF (M .GT. N) CONLIM = 1.0E+5
ITNLIM = 100

CALL LSQR( M,N,M,B,P,U,V,W,X,SE,
     ATOL,BTOL,CONLIM,ITNLIM,NOUT,ISTOP,RNORM,ANORM,ACOND )

C OUTPUT RESULTS
C
WRITE(NOUT, 2000)
WRITE(NOUT, 3000) (J,X(J), J=1,N)
WRITE(NOUT, 4000) (J,SE(J),J=1,N)
RETURN

1000 FORMAT(1H1
1 / 28H LEAST-SQUARES TEST PROBLEM.
2 / 25H SINGULAR VALUES REPEATED, I3, 8H TIMES., 8X,
3 / 15H POWER FACTOR =, I3
4 / 11H COND(A) =, 1PE12.4, 8X,
5 / 11H NORM(R) =, 1PE20.19)
2000 FORMAT(/ 9H SOLUTION)
3000 FORMAT(/ 16H STANDARD ERRORS)
4000 FORMAT(5(I7, 1PE17.9))

C END OF TEST
END

C EXAMPLE MAIN PROGRAM
C
CALL TEST( 10,10,1,6 )
CALL TEST( 80,40,4,2 )
STOP
END
APPENDIX B: OUTPUT FROM LSQR

The following listings illustrate the solution of two test problems using the Fortran routines in Appendix A. The problems are:

- \(P(10,10,1,6)\) - an ill-conditioned compatible system \(Ax = b\),
- \(P(60,40,4,2)\) - a reasonably well-conditioned least-squares problem, \(\min \|Ax - b\|_2\),

where problem \(P(m,n,d,p)\) is defined in section 8.1. The machine used was a Burroughs B6700. The particular tolerances input to LSQR (namely ATOL = BTOL = 1.0E-10) requested slightly less than maximum attainable precision for this machine.

The quantities output by subroutine LSQR each iteration are as follows. They are expressed in terms of the current approximate solution vector \(x_k\) and the corresponding residual vector \(r_k = b - Ax_k\).

- **ITN**: The iteration number. For larger problems a line is printed every tenth iteration.
- **X(1)**: The value of the first component of \(x_k\).
- **NORM(R)**: The value of \(\|r_k\|\). This converges to zero if \(Ax = b\) is compatible; otherwise to a positive limit.
- **NORM(A.R)**: The value of \(\|A^T r_k\|\). This converges to zero in all cases.
- **COMPATIBLE**: A dimensionless quantity which should converge to zero if and only if \(Ax = b\) is compatible. It is a product of sines \(\theta_1 \theta_2 \ldots \theta_k\) which estimates \(\|r_k\| / \|b\|\).
- **INCOMPATIBLE**: A dimensionless quantity which should converge to zero if and only if the optimum residual \(r = b - Ax\) is nonzero. It is an estimate of \(\|A^T r_k\| / (\|A\|_F \|r_k\|)\).
- **NORM(A)**: A monotonically increasing estimate of \(\|A\|_F\).
- **COND(A)**: A monotonically increasing estimate of \(\text{cond}(A) = \|A\|_F \|A^+\|_F\).
### LEAST-SQUARES TEST PROBLEM.

<table>
<thead>
<tr>
<th>COND(A) = 1.0000E+06</th>
<th>NORM(R) = 0.</th>
</tr>
</thead>
</table>

**LSQR --- LEAST-SQUARES SOLUTION OF A.X = B**  
(I.E. MINIMIZE NORM(R), WHERE B = B - A.X )

**THE MATRIX A HAS DIMENSIONS 10 BY 10**

<table>
<thead>
<tr>
<th>ATOL</th>
<th>1.0E-10</th>
<th>BTOL</th>
<th>1.0E-10</th>
<th>CONLIM</th>
<th>1.0E+10</th>
<th>ITNLIM</th>
<th>100</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>ITN</th>
<th>X(1)</th>
<th>NORM(R)</th>
<th>NORM(A.R)</th>
<th>COMPATIBLE INCOMPATIBLE</th>
<th>NORM(A)</th>
<th>NORM(R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2.052E+00</td>
<td>2.052E+00</td>
<td>2.052E+00</td>
<td>2.052E+00</td>
<td>2.052E+00</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-2.9218120797E-01</td>
<td>2.9396364399E-01</td>
<td>2.777E-02</td>
<td>1.137E-02</td>
<td>8.351E-02</td>
<td>1.13E+00</td>
</tr>
<tr>
<td>3</td>
<td>-2.8897591886E-01</td>
<td>2.5614275400E-01</td>
<td>2.176E-02</td>
<td>1.365E-02</td>
<td>7.396E-02</td>
<td>1.75E+00</td>
</tr>
<tr>
<td>4</td>
<td>-1.6482708922E-01</td>
<td>2.8079115638E-01</td>
<td>9.416E-03</td>
<td>9.132E-02</td>
<td>4.829E-02</td>
<td>1.17E+00</td>
</tr>
<tr>
<td>5</td>
<td>-4.3458584222E-01</td>
<td>9.8516462677E-02</td>
<td>1.512E-03</td>
<td>4.481E-02</td>
<td>1.312E-02</td>
<td>1.17E+00</td>
</tr>
<tr>
<td>6</td>
<td>-1.1935068368E+00</td>
<td>2.6712942622E-02</td>
<td>6.629E-03</td>
<td>1.190E-02</td>
<td>2.166E-02</td>
<td>1.17E+00</td>
</tr>
<tr>
<td>7</td>
<td>-1.9640094294E+00</td>
<td>2.5314279652E-02</td>
<td>1.832E-04</td>
<td>1.151E-02</td>
<td>4.634E-03</td>
<td>1.17E+00</td>
</tr>
<tr>
<td>8</td>
<td>-1.977951358E+00</td>
<td>2.5223397617E-02</td>
<td>1.130E-03</td>
<td>1.147E-02</td>
<td>2.918E-02</td>
<td>1.15E+00</td>
</tr>
<tr>
<td>9</td>
<td>-1.4987142112E+00</td>
<td>3.8117048718E-03</td>
<td>4.249E-06</td>
<td>1.733E-03</td>
<td>6.849E-04</td>
<td>1.63E+00</td>
</tr>
<tr>
<td>10</td>
<td>-1.4886698928E+00</td>
<td>3.8116443566E-03</td>
<td>3.639E-06</td>
<td>1.733E-03</td>
<td>5.822E-04</td>
<td>1.64E+00</td>
</tr>
<tr>
<td>11</td>
<td>-1.4863681392E+00</td>
<td>3.8034054546E-03</td>
<td>2.493E-04</td>
<td>1.738E-03</td>
<td>3.981E-02</td>
<td>1.65E+00</td>
</tr>
<tr>
<td>12</td>
<td>-9.8166258668E-01</td>
<td>1.6976019812E-01</td>
<td>1.746E-04</td>
<td>7.146E-04</td>
<td>5.335E-02</td>
<td>1.93E+00</td>
</tr>
<tr>
<td>13</td>
<td>-8.8371043513E-01</td>
<td>4.8540073308E-04</td>
<td>4.278E-06</td>
<td>1.845E-06</td>
<td>5.462E-03</td>
<td>1.93E+00</td>
</tr>
<tr>
<td>14</td>
<td>-8.3787692581E-01</td>
<td>4.8532696151E-04</td>
<td>1.686E-07</td>
<td>1.845E-04</td>
<td>2.876E-04</td>
<td>2.00E+00</td>
</tr>
<tr>
<td>15</td>
<td>-8.3703803473E-01</td>
<td>4.8530423731E-04</td>
<td>2.875E-08</td>
<td>1.845E-04</td>
<td>3.535E-05</td>
<td>2.00E+00</td>
</tr>
<tr>
<td>16</td>
<td>-8.3782995856E-01</td>
<td>4.8530421761E-04</td>
<td>6.436E-09</td>
<td>1.845E-04</td>
<td>7.763E-05</td>
<td>2.00E+00</td>
</tr>
<tr>
<td>18</td>
<td>4.4830311734E-01</td>
<td>2.2665093442E-04</td>
<td>2.943E-06</td>
<td>1.831E-04</td>
<td>5.755E-03</td>
<td>2.26E+00</td>
</tr>
<tr>
<td>20</td>
<td>4.4836416966E-01</td>
<td>1.3726150576E-05</td>
<td>3.956E-06</td>
<td>6.242E-06</td>
<td>1.245E-01</td>
<td>2.32E+00</td>
</tr>
<tr>
<td>21</td>
<td>-2.4869779378E-01</td>
<td>1.3097399322E-05</td>
<td>1.084E-06</td>
<td>5.956E-06</td>
<td>3.299E-02</td>
<td>2.31E+04</td>
</tr>
<tr>
<td>22</td>
<td>-2.4838138572E-01</td>
<td>9.6163025747E-06</td>
<td>3.765E-10</td>
<td>4.373E-06</td>
<td>1.555E-04</td>
<td>2.53E+04</td>
</tr>
<tr>
<td>23</td>
<td>-2.4830813272E-01</td>
<td>9.6159157016E-06</td>
<td>1.186E-08</td>
<td>4.373E-06</td>
<td>4.892E-04</td>
<td>2.53E+04</td>
</tr>
<tr>
<td>24</td>
<td>-2.4830853635E-01</td>
<td>9.6156667487E-06</td>
<td>3.149E-09</td>
<td>4.373E-06</td>
<td>1.289E-04</td>
<td>2.54E+04</td>
</tr>
<tr>
<td>25</td>
<td>-2.4830853393E-01</td>
<td>9.6156668146E-06</td>
<td>1.280E-09</td>
<td>4.373E-06</td>
<td>4.944E-05</td>
<td>2.69E+04</td>
</tr>
<tr>
<td>26</td>
<td>-2.4830853372E-01</td>
<td>9.6156669389E-06</td>
<td>3.565E-10</td>
<td>4.373E-06</td>
<td>1.336E-05</td>
<td>2.78E+04</td>
</tr>
<tr>
<td>27</td>
<td>-2.4830853374E-01</td>
<td>9.6156663839E-06</td>
<td>9.624E-12</td>
<td>4.373E-06</td>
<td>3.599E-07</td>
<td>2.78E+04</td>
</tr>
<tr>
<td>29</td>
<td>-2.482695733E-01</td>
<td>9.6156623289E-06</td>
<td>2.784E-08</td>
<td>4.373E-06</td>
<td>1.423E-03</td>
<td>2.79E+04</td>
</tr>
<tr>
<td>30</td>
<td>-1.9569041273E-01</td>
<td>9.5881720382E-06</td>
<td>4.985E-09</td>
<td>4.361E-06</td>
<td>1.738E-04</td>
<td>2.98E+05</td>
</tr>
<tr>
<td>No.</td>
<td>Value</td>
<td>Value</td>
<td>Value</td>
<td>Value</td>
<td>Value</td>
<td>Value</td>
</tr>
<tr>
<td>-----</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>31</td>
<td>-1.956808622E-01</td>
<td>9.581167047E-06</td>
<td>1.532E-09</td>
<td>4.361E-06</td>
<td>5.326E-05</td>
<td>3.08E+00</td>
</tr>
<tr>
<td>32</td>
<td>-1.956758341E-01</td>
<td>9.581164799E-06</td>
<td>9.431E-10</td>
<td>4.761E-06</td>
<td>3.264E-05</td>
<td>3.01E+00</td>
</tr>
<tr>
<td>33</td>
<td>-1.952128599E-01</td>
<td>9.587922496E-06</td>
<td>7.155E-10</td>
<td>4.16E-06</td>
<td>2.475E-05</td>
<td>3.02E+00</td>
</tr>
<tr>
<td>34</td>
<td>-1.952069909E-01</td>
<td>9.587920817E-06</td>
<td>5.58E-09</td>
<td>4.36E-06</td>
<td>1.92E-04</td>
<td>3.02E+00</td>
</tr>
<tr>
<td>35</td>
<td>9.752465478E+00</td>
<td>3.408579219E-06</td>
<td>1.351E-07</td>
<td>1.741E-06</td>
<td>1.39E-02</td>
<td>3.18E+00</td>
</tr>
<tr>
<td>36</td>
<td>8.499518943E+00</td>
<td>2.319037654E-06</td>
<td>1.416E-07</td>
<td>1.917E-06</td>
<td>1.127E-01</td>
<td>3.18E+00</td>
</tr>
<tr>
<td>37</td>
<td>8.999713240E+00</td>
<td>9.537733972E-06</td>
<td>1.39E-08</td>
<td>3.887E-06</td>
<td>3.62E-02</td>
<td>3.22E+00</td>
</tr>
<tr>
<td>38</td>
<td>8.999973652E+00</td>
<td>3.837350176E-06</td>
<td>2.525E-08</td>
<td>1.18E-08</td>
<td>2.58E-01</td>
<td>3.23E+00</td>
</tr>
<tr>
<td>39</td>
<td>8.99997683E+00</td>
<td>1.495247214E-08</td>
<td>1.516E-09</td>
<td>6.391E-09</td>
<td>7.417E-02</td>
<td>3.17E+00</td>
</tr>
<tr>
<td>40</td>
<td>8.99997986E+00</td>
<td>3.815689880E-09</td>
<td>1.523E-11</td>
<td>1.735E-09</td>
<td>1.17E-03</td>
<td>3.39E+00</td>
</tr>
</tbody>
</table>

### Stopping Conditions

<table>
<thead>
<tr>
<th>Condition</th>
<th>Count</th>
<th>Norm(R)</th>
<th>True/False</th>
<th>Norm(A)</th>
<th>Estimate</th>
<th>3.18E+06</th>
<th>Cond(A)</th>
<th>Estimate</th>
<th>1.86E+06</th>
<th>Norm(X)</th>
<th>Estimate</th>
<th>1.68E+06</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stopping</td>
<td></td>
<td>Norm(R)</td>
<td>True</td>
<td>2.19E+00</td>
<td>5.93E+00</td>
<td>1</td>
<td>3.19E+00</td>
<td>1</td>
<td>1.66E+00</td>
<td>1</td>
<td>1.68E+00</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td></td>
<td>Norm(A)</td>
<td>True</td>
<td>3.01E+00</td>
<td>9.47E-04</td>
<td>1</td>
<td>3.01E+00</td>
<td>1</td>
<td>1.68E+00</td>
<td>1</td>
<td>1.68E+00</td>
<td></td>
</tr>
<tr>
<td>Solution</td>
<td></td>
<td>True</td>
<td>True</td>
<td>3.14E+00</td>
<td>1.52E+00</td>
<td>1</td>
<td>5.00E+00</td>
<td>1</td>
<td>4.90E+00</td>
<td>1</td>
<td>4.90E+00</td>
<td></td>
</tr>
<tr>
<td>Errors</td>
<td></td>
<td>True</td>
<td>True</td>
<td>3.16E+00</td>
<td>1.45E+00</td>
<td>1</td>
<td>5.00E+00</td>
<td>1</td>
<td>4.71E+00</td>
<td>1</td>
<td>4.71E+00</td>
<td></td>
</tr>
</tbody>
</table>
LEAST-SQUARES TEST PROBLEM.

SINGULAR VALUES REPEATED 4 TIMES.

\[
\begin{align*}
\text{COND}(A) &= 1.3089 \times 10^2 \\
\text{NOR}(R) &= 1.1499 \times 10^5 \\
\end{align*}
\]

\[
\text{LSQR} \quad \text{LEAST-SQUARES SOLUTION OF } A \times X = B \\
(\text{I.E. MINIMIZE } \text{NOR}(R), \text{ WHERE } R = B - A \times X)
\]

THE MATRIX A HAS DIMENSIONS \( \mathbb{R}^m \) BY \( \mathbb{R}^n \)

\[
\begin{align*}
\text{ATOL} &= 1.338 \times 10^{-10} \\
\text{BTOL} &= 1.00 \times 10^{-10} \\
\text{CONLIM} &= 1.00 \times 10^4 \\
\text{ITLIM} &= 1.00
\end{align*}
\]

<table>
<thead>
<tr>
<th>ITN</th>
<th>X(I)</th>
<th>\text{NOR}(R)</th>
<th>\text{NOR}(A,R)</th>
<th>\text{COMPATIBLE INCOMPATIBLE}</th>
<th>\text{NOR}(A)</th>
<th>\text{COND}(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2.8885842421E+01</td>
<td>2.8885842421E+01</td>
<td>1.42E+01</td>
<td>1.42E+01</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-2.3314124243E-01</td>
<td>1.9828790898E+01</td>
<td>3.78E+00</td>
<td>7.86E+00</td>
<td>6.24E+00</td>
<td>7.18E+00</td>
</tr>
<tr>
<td>2</td>
<td>9.2514146574E-01</td>
<td>1.4498146876E+01</td>
<td>5.51E+00</td>
<td>5.16E+01</td>
<td>1.69E+01</td>
<td>1.00E+02</td>
</tr>
<tr>
<td>3</td>
<td>-2.3149775348E+00</td>
<td>1.1916971306E+00</td>
<td>3.45E+00</td>
<td>3.95E+01</td>
<td>2.42E+01</td>
<td>1.28E+02</td>
</tr>
<tr>
<td>4</td>
<td>-5.4412306278E+00</td>
<td>8.6829713292E+00</td>
<td>2.19E+00</td>
<td>3.09E+01</td>
<td>1.76E+01</td>
<td>1.43E+02</td>
</tr>
<tr>
<td>5</td>
<td>-5.0275711966E+00</td>
<td>6.8719707239E+00</td>
<td>1.31E+00</td>
<td>2.47E+01</td>
<td>1.25E+01</td>
<td>1.52E+02</td>
</tr>
<tr>
<td>6</td>
<td>-4.1976154768E+00</td>
<td>5.2922887341E+00</td>
<td>6.04E+00</td>
<td>1.84E+00</td>
<td>7.95E+00</td>
<td>1.35E+02</td>
</tr>
<tr>
<td>7</td>
<td>-6.6145728850E+02</td>
<td>3.8613906426E+00</td>
<td>2.72E+00</td>
<td>1.37E+00</td>
<td>4.48E+00</td>
<td>1.59E+02</td>
</tr>
<tr>
<td>8</td>
<td>5.8127699460E+00</td>
<td>2.7225894432E+00</td>
<td>4.35E-02</td>
<td>9.69E-02</td>
<td>9.18E-02</td>
<td>9.35E+00</td>
</tr>
<tr>
<td>9</td>
<td>1.8688163660E+00</td>
<td>1.9925787110E+00</td>
<td>2.96E-02</td>
<td>7.05E-02</td>
<td>9.35E-02</td>
<td>1.59E+00</td>
</tr>
<tr>
<td>10</td>
<td>1.8689611668E+00</td>
<td>1.9924456407E+00</td>
<td>7.40E-02</td>
<td>7.94E-02</td>
<td>1.93E-03</td>
<td>1.20E+00</td>
</tr>
<tr>
<td>11</td>
<td>1.4920895940E+00</td>
<td>1.9727297215E+00</td>
<td>2.07E-01</td>
<td>7.82E-02</td>
<td>5.39E-02</td>
<td>1.88E+00</td>
</tr>
<tr>
<td>12</td>
<td>3.8760040890E+01</td>
<td>1.8610128863E+00</td>
<td>4.33E-02</td>
<td>6.62E-02</td>
<td>1.05E-02</td>
<td>2.05E+01</td>
</tr>
<tr>
<td>13</td>
<td>3.8983824899E+00</td>
<td>1.8599397490E+00</td>
<td>4.56E-04</td>
<td>6.62E-02</td>
<td>1.14E-04</td>
<td>2.14E+01</td>
</tr>
<tr>
<td>14</td>
<td>3.9000002111E+00</td>
<td>1.8599919514E+00</td>
<td>4.24E-04</td>
<td>6.62E-02</td>
<td>1.03E-04</td>
<td>2.28E+01</td>
</tr>
<tr>
<td>15</td>
<td>3.9000000055E+00</td>
<td>1.8599995153E+00</td>
<td>1.26E-07</td>
<td>6.62E-02</td>
<td>1.05E-07</td>
<td>2.27E+01</td>
</tr>
<tr>
<td>16</td>
<td>3.9000000018E+00</td>
<td>1.8599995153E+00</td>
<td>2.33E-07</td>
<td>6.62E-02</td>
<td>5.54E-08</td>
<td>2.27E+01</td>
</tr>
<tr>
<td>17</td>
<td>3.8999999250E+00</td>
<td>1.8599995153E+00</td>
<td>3.26E-08</td>
<td>6.62E-02</td>
<td>7.15E-09</td>
<td>2.45E+01</td>
</tr>
<tr>
<td>18</td>
<td>3.8999999999E+00</td>
<td>1.8599995153E+00</td>
<td>3.78E-09</td>
<td>6.62E-02</td>
<td>6.72E-10</td>
<td>2.46E+01</td>
</tr>
<tr>
<td>19</td>
<td>3.9000000001E+00</td>
<td>1.8599995153E+00</td>
<td>4.05E-10</td>
<td>6.62E-02</td>
<td>8.41E-11</td>
<td>2.59E+01</td>
</tr>
</tbody>
</table>

NO. OF ITERATIONS: 19

STOPTING CONDITION: 2 (NOR(A,R) IS SMALL ENOUGH, GIVEN ATOL)

\[
\begin{align*}
\text{NOR}(B) &= \text{TRUE} \\
\text{NOR}(R) &= \text{ESTIMATE} \\
\end{align*}
\]

\[
\begin{align*}
\text{COND}(A) \quad \text{ESTIMATE} \quad 2.792078 \times 10^{02} \\
\text{NOR}(X) \quad \text{ESTIMATE} \quad 1.433178 \times 10^{02}
\end{align*}
\]
<table>
<thead>
<tr>
<th>SOLUTION</th>
<th>STANDARD ERRORS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 3.900000000E+01 2 3.800000000E+01</td>
<td>1 1.17405796E+01 2 1.14084847E+01</td>
</tr>
<tr>
<td>6 3.399999999E+01 7 3.299999999E+01</td>
<td>6 1.775805396E+00 7 3.99761946E+00</td>
</tr>
<tr>
<td>11 2.000000012E+01 12 2.800000003E+01</td>
<td>11 3.95847612E+00 12 3.42319222E+00</td>
</tr>
<tr>
<td>16 2.400000000E+01 17 2.300000000E+01</td>
<td>16 1.636315294E+00 17 2.44062067E+00</td>
</tr>
<tr>
<td>21 1.400000001E+02 22 1.900000001E+01</td>
<td>21 3.887139999E+00 22 3.348173999E+00</td>
</tr>
<tr>
<td>26 1.400000008E+01 27 1.300000000E+01</td>
<td>26 1.351041103E+00 27 2.411460018E+00</td>
</tr>
<tr>
<td>31 8.99999993E+00 32 7.99999994E+00</td>
<td>31 3.879527642E+00 32 3.29284479E+00</td>
</tr>
<tr>
<td>36 4.000000002E+00 37 3.800000005E+00</td>
<td>36 1.272382778E+00 37 2.414297039E+00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SOLUTION</th>
<th>STANDARD ERRORS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 3.700000000E+01 4 3.600000000E+01 5 3.500000000E+01 6 3.400000000E+01</td>
<td>3 1.300000000E+01 4 2.600000000E+01 5 2.500000000E+01 6 2.400000000E+01</td>
</tr>
<tr>
<td>8 3.199999999E+01 9 3.100000000E+01 10 3.000000000E+01</td>
<td>13 2.700000000E+01 14 2.600000000E+01 15 2.500000000E+01 16 2.400000000E+01</td>
</tr>
<tr>
<td>13 2.200000000E+01 14 2.100000000E+01 15 2.000000000E+01</td>
<td>18 2.200000000E+01 19 2.100000000E+01 20 2.000000000E+01 21 1.900000000E+01</td>
</tr>
<tr>
<td>18 2.100000000E+01 19 2.000000000E+01 20 1.900000000E+01</td>
<td>23 1.700000000E+01 24 1.600000000E+01 25 1.500000000E+01 26 1.400000000E+01</td>
</tr>
<tr>
<td>23 1.600000000E+01 24 1.500000000E+01 25 1.400000000E+01</td>
<td>28 1.199999999E+01 29 1.100000000E+01 30 1.000000000E+01 31 9.99999999E+00</td>
</tr>
<tr>
<td>28 1.199999999E+01 29 1.100000000E+01 30 1.000000000E+01</td>
<td>33 6.99999999E+00 34 6.99999999E+00 35 5.00000000E+00 36 5.00000000E+00</td>
</tr>
<tr>
<td>33 6.99999999E+00 34 6.99999999E+00 35 5.00000000E+00</td>
<td>38 2.800000000E+00 39 1.000000000E+00 40 7.67767014E-09</td>
</tr>
</tbody>
</table>
**Title:** A Bidiagonalization Algorithm for Sparse Linear Equations and Least-Squares Problems

**Authors:** Christopher C. Paige and Michael A. Saunders

**Abstract:**
This document has been approved for public release and sale; its distribution is unlimited.

**Keywords:**
- Analysis of Variance
- Conjugate-Gradient Method
- Least Squares
- Linear Equations
- Preconditioning
- Regression
- Sparse Matrix

**References:**
See attached.
A BIDIAgonALIZATION ALGORITHM FOR SPARSE LINEAR EQUATIONS AND LEAST-SQUARES PROBLEMS
SOL 78-19
Christopher C. Paige and Michael A. Saunders

A method is given for solving \( Ax = b \) and \( \min \| Ax - b \|_2 \) where the matrix \( A \) is large and sparse. The method is based on the bidiagonalization procedure of Golub and Kahan. It is analytically equivalent to the method of conjugate gradients (CG) but possesses more favorable numerical properties. The Fortran implementation of the method (subroutine LSQR) incorporates reliable stopping criteria and provides estimates of various quantities including standard errors for \( x \) and the condition number of \( A \). Numerical tests are described comparing LSQR with several other CG algorithms. Further results for a large practical problem illustrate the effect of pre-conditioning least-squares problems using a sparse LU factorization of \( A \).