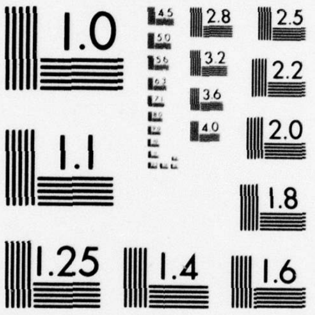
AD-A063 334   STANFORD UNIV CALIF SYSTEMS OPTIMIZATION LAB   F/G 12/1
A BIDIAGONALIZATION ALGORITHM FOR SPARSE LINEAR EQUATIONS AND L--ETC(U)
OCT 78 C C PAIGE, M A SAUNDERS   N00014-75-C-0267
UNCLASSIFIED   SOL-78-19   NL

1 OF 1
AD
A063334


1.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0
10
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

Systems
Optimization
Laboratory

1

LEVEL II

Department of Operations Research
Stanford University
Stanford, CA 94305

1

SYSTEMS OPTIMIZATION LABORATORY
DEPARTMENT OF OPERATIONS RESEARCH
Stanford University
Stanford, California
94305

6

A BIDIAGONALIZATION ALGORITHM FOR SPARSE LINEAR EQUATIONS AND LEAST-SQUARES PROBLEMS.

by

10 Christopher C. Paige[†] [redacted] Michael A. Saunders[*]

14 SOL-78-19

9 TECHNICAL REP[redacted]T, SOL-78-19

11 Oct[redacted]78

12 95p.

15 NØØ14-75-C-Ø267,
NSF-MCS76-2ØØ19

[†] 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.

408 765

CONTENTS

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:

$$\text{Unsymmetric equations:} \qquad \text{solve } Ax = b$$

$$\text{Linear least squares:} \qquad \text{minimize } \|Ax - b\|_2$$

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 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-likc 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, $r = b - Ax$, $A^T r$, $\|r\|_2$, $\|A\|_F$, 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 $\|v\|_2 = (v^T v)^{\frac{1}{2}}$. For a matrix A, $\|A\|$ will usually mean the Frobenius norm, $\|A\|_F = (\Sigma\alpha_{ij}^2)^{\frac{1}{2}}$, and the condition number for an unsymmetric matrix A is defined by $\mathrm{cond}(A) = \|A\| \, \|A^+\|$ where A^+ denotes the pseudo-inverse of A. The relative precision of floating-point arithmetic will be ε, 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)

(a) $\beta_1 v_1 = b$.

(b)
$$\left.\begin{aligned} w_i &= Mv_i - \beta_i v_{i-1} \\ \alpha_i &= v_i^T w_i \\ \beta_{i+1} v_{i+1} &= w_i - \alpha_i v_i \end{aligned}\right\} i = 1, 2, \ldots \qquad (2.1)$$

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

$$MV_k = V_k T_k + \beta_{k+1} v_{k+1} e_k^T \qquad (2.2)$$

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 β_{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 η_k gives $MV_k y_k = V_k T_k y_k + \beta_{k+1} v_{k+1} \eta_k$. Since $V_k(\beta_1 e_1) = b$ by definition, it follows that if y_k and x_k are defined by the equations

$$T_k y_k = \beta_1 e_1 \tag{2.3}$$

$$x_k = V_k y_k \tag{2.4}$$

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 $T_k = L_k D_k L_k^{\mathrm{T}}$ and is reliable when M (and hence T_k) is positive definite, while algorithm SYMMLQ employs the orthogonal factorization $T_k = \bar{L}_k Q_k$ 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

$$\begin{bmatrix} I & A \\ A^{\mathrm{T}} & \end{bmatrix} \begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ o \end{bmatrix} \tag{2.5}$$

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

$$T_{2k+1}y_{2k+1} \equiv \begin{bmatrix} 1 & \alpha_1 & & & & & & \\ \alpha_1 & 0 & \beta_2 & & & & & \\ & \beta_2 & 1 & \alpha_2 & & & & \\ & & \alpha_2 & 0 & \beta_3 & & & \\ & & & \cdot & \cdot & \cdot & & \\ & & & & \cdot & \cdot & \cdot & \\ & & & & & \alpha_k & 0 & \beta_{k+1} \\ & & & & & & \beta_{k+1} & 1 \end{bmatrix} \begin{bmatrix} \tau_1 \\ \eta_1 \\ \tau_2 \\ \eta_2 \\ \cdot \\ \cdot \\ \eta_k \\ \tau_{k+1} \end{bmatrix} = \beta_1 e_1 \qquad (2.6)$$

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 $T_{2k+1} = \bar{L}_{2k+1}Q_{2k+1}$. 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

$$\begin{bmatrix} I & B_k \\ B_k^T & \end{bmatrix} \begin{bmatrix} t_{k+1} \\ y_k \end{bmatrix} = \begin{bmatrix} \beta_1 e_1 \\ 0 \end{bmatrix} \qquad (2.7)$$

where $B_k \equiv \text{bidiag}(\beta_i, \alpha_i)$ is a lower bidiagonal matrix of order $k+1$ by k. Comparing (2.7) with (2.5) immediately reveals another least-squares problem, $\min \|B_k y_k - \beta_1 e_1\|$. 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 $\min \|Ax - b\|$ leads to a series of subproblems $\min \|B_k y_k - \beta_1 e_1\|$ 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)

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

$$\left.\begin{aligned}\text{(b)}\quad \beta_{i+1} u_{i+1} &= A v_i - \alpha_i u_i \\ \alpha_{i+1} v_{i+1} &= A^T u_{i+1} - \beta_{i+1} v_i\end{aligned}\right\}\quad i = 1,\ 2,\ \ldots \tag{3.1}$$

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

$$U_k \equiv [u_1, u_2, \ldots, u_k]\,, \quad B_k \equiv \begin{bmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \beta_3 & \cdot & \\ & & \cdot\ \cdot & \\ & & \cdot & \alpha_k \\ & & & \beta k{+}1 \end{bmatrix},$$
$$V_k \equiv [v_1, v_2, \ldots, v_k]\,,$$

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

$$U_{k+1}(\beta_1 e_1) = b \quad , \tag{3.2}$$

$$AV_k = U_{k+1}B_k \quad , \tag{3.3}$$

$$A^T U_{k+1} = V_k B_k^{\,T} + \alpha_{k+1} v_{k+1} e_{k+1}^{\,T} \quad . \tag{3.4}$$

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.

<u>Bidiag 2</u> (Starting vector $A^T b$; reduction to upper bidiagonal form)

(a) $\quad \theta_1 v_1 = A^T b \,, \qquad \rho_1 p_1 = A v_1 \;.$

$$\text{(b)}\quad \left.\begin{aligned} \theta_{i+1} v_{i+1} &= A^T p_i - \rho_i v_i \\ \rho_{i+1} p_{i+1} &= A v_{i+1} - \theta_{i+1} p_i \end{aligned}\right\} \quad i = 1,\, 2,\, \ldots \tag{3.5}$$

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

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

$$V_k \equiv [v_1, v_2, \ldots, v_k] \,,$$

we may rewrite (3.5) as

$$V_k(\theta_1 e_1) = A^T b \quad , \tag{3.6}$$

$$AV_k = P_k R_k \quad , \tag{3.7}$$

$$A^T P_k = V_k R_k^T + \theta_{k+1} v_{k+1} e_k^T \quad , \qquad (3.8)$$

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

$$B_k^T B_k = R_k^T R_k \qquad (3.9)$$

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

$$Q_k B_k = \begin{bmatrix} R_k \\ 0 \end{bmatrix} \qquad (3.10)$$

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

$$U_{k+1} = [U_k \; u_{k+1}] = \begin{bmatrix} P_k & \frac{r_k}{\|r_k\|} \end{bmatrix} Q_k \qquad (3.11)$$

for some vector r_k. We also have the identities

$$\alpha_1^2 + \beta_2^2 = \rho_1^2 \quad , \qquad \alpha_1\beta_1 = \theta_1 \quad ,$$
$$\alpha_i^2 + \beta_{i+1}^2 = \rho_i^2 + \theta_i^2 \, , \qquad \alpha_i\beta_i = \rho_{i-1}\theta_i \text{ for } i > 1 \, . \tag{3.12}$$

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

$$x_k = V_k y_k \tag{4.1}$$

$$r_k = b - Ax_k \tag{4.2}$$

$$t_{k+1} = \beta_1 e_1 - B_k y_k \tag{4.3}$$

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

$$r_k = U_{k+1} t_{k+1} \tag{4.4}$$

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

$$\min_{y_k} \|\beta_1 e_1 - B_k y_k\| \tag{4.5}$$

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

$$Q_k \begin{bmatrix} B_k & \beta_1 e_1 \end{bmatrix} = \begin{bmatrix} R_k & f_k \\ & \bar{\phi}_{k+1} \end{bmatrix} \equiv \left[\begin{array}{ccccc:c} \rho_1 & \theta_2 & & & & \phi_1 \\ & \rho_2 & \theta_3 & & & \phi_2 \\ & & \cdot & \cdot & & \cdot \\ & & & \rho_{k-1} & \theta_k & \phi_{k-1} \\ & & & & \rho_k & \phi_k \\ \hdashline & & & & & \bar{\phi}_{k+1} \end{array}\right] \tag{4.6}$$

where $Q_k \equiv Q_{k,k+1} \ldots Q_{2,3}Q_{1,2}$ is a product of plane rotations (e.g. Wilkinson [27]) designed to eliminate the subdiagonals β_2, β_3, ... of B_k. The vectors y_k and t_{k+1} could then be found from

$$R_k y_k = f_k \quad , \tag{4.7}$$

$$t_{k+1} = Q_k^{\mathrm{T}} \left[\begin{array}{c} 0 \\ \hline \bar{\phi}_{k+1} \end{array}\right] \quad . \tag{4.8}$$

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

$$x_k = V_k R_k^{-1} f_k \equiv D_k f_k \tag{4.9}$$

where the columns of $D_k \equiv [d_1 \; d_2 \; \ldots \; d_k]$ can be found successively from the system $R_k^{\mathrm{T}} D_k^{\mathrm{T}} = V_k^{\mathrm{T}}$ by forward substitution. With $d_o = x_o = 0$ this gives

$$d_k = \frac{1}{\rho k} (v_k - \theta_k d_{k-1}) \quad , \tag{4.10}$$

$$x_k = x_{k-1} + \phi_k d_k \quad , \tag{4.11}$$

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 $Q_{k,k+1}$ to operate on rows k and $k+1$ of the transformed $[B_k \;\; \beta_1 e_1]$ to annihilate β_{k+1}. This gives the following simple recurrence relation:

$$\begin{bmatrix} c_k & s_k \\ s_k & -c_k \end{bmatrix} \begin{bmatrix} \bar\rho_k & 0 & \bar\phi_k \\ \beta_{k+1} & \alpha_{k+1} & 0 \end{bmatrix} = \begin{bmatrix} \rho_k & \theta_{k+1} & \phi_k \\ 0 & \bar\rho_{k+1} & \bar\phi_{k+1} \end{bmatrix} \qquad (4.12)$$

where $\bar\rho_1 \equiv \alpha_1$, $\bar\phi_1 \equiv \beta_1$, and the scalars c_k and s_k are the nontrivial elements of $Q_{k,k+1}$. The quantities $\bar\rho_k, \bar\phi_k$ are intermediate scalars that are subsequently replaced by ρ_k, ϕ_k.

The rotations $Q_{k,k+1}$ 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 $\bar\phi_{k+1}$.

Some of the work in (4.10) can be eliminated by using vectors $w_k \equiv \rho_k d_k$ in place of d_k. The main steps of LSQR can now be summarized as follows. (As usual the scalars $\alpha_i \geq 0$ and $\beta_i \geq 0$ are chosen to normalize the corresponding vectors; for example, $\alpha_1 v_1 = A^T u_1$ implies the computations $\bar v_1 = A^T u_1$, $\alpha_1 = \|\bar v_1\|$, $v_1 = (1/\alpha_1)\bar v_1$.)

Algorithm LSQR

```
1. (Initialize.)
      $\beta_1 u_1 = b$ ,   $\alpha_1 v_1 = A^T u_1$ ,   $w_1 = v_1$ ,   $x_o = 0$ ,
      $\bar{\phi}_1 = \beta_1$ ,   $\bar{\rho}_1 = \alpha_1$ .

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

3. (Continue the bidiagonalization.)
   (a) $\beta_{i+1} u_{i+1} = A v_i - \alpha_i u_i$
   (b) $\alpha_{i+1} v_{i+1} = A^T u_{i+1} - \rho_{i+1} v_i$ .

4. (Construct and apply next orthogonal transformation.)
   (a) $\rho_i = (\bar{\rho}_i^2 + \beta_{i+1}^2)^{\frac{1}{2}}$
   (b) $c_i = \bar{\rho}_i / \rho_i$
   (c) $s_i = \beta_{i+1} / \rho_i$
   (d) $\theta_{i+1} = s_i \alpha_{i+1}$
   (e) $\bar{\rho}_{i+1} = -c_i \alpha_{i+1}$
   (f) $\phi_i = c_i \bar{\phi}_i$
   (g) $\bar{\phi}_{i+1} = s_i \bar{\phi}_i$ .

5. (Update $x,w$.)
   (a) $x_i = x_{i-1} + (\phi_i / \rho_i) w_i$
   (b) $w_{i+1} = v_{i+1} - (\theta_{i+1} / \rho_i) w_i$ .

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 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 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 Av 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 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 ($\|Ae_j\| = 1$, $j = 1, \ldots, n$), 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 $n^{\frac{1}{2}}$ (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

$$r_k = \bar{\phi}_{k+1} U_{k+1} Q_k^T e_{k+1} \qquad (5.1)$$

(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

$$\|r_k\| = \bar{\phi}_{k+1} = \beta_1 s_k s_{k-1} \ldots s_1 \quad , \tag{5.2}$$

where the form of $\bar{\phi}_{k+1}$ 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

$$\begin{aligned} A^T r_k &= \bar{\phi}_{k+1}(V_k B_k^{\ T} + \alpha_{k+1} v_{k+1} e_{k+1}^{\ T}) Q_k^{\ T} e_{k+1} \\ &= \bar{\phi}_{k+1} V_k \begin{bmatrix} R_k^{\ T} & 0 \end{bmatrix} e_{k+1} \ + \ \bar{\phi}_{k+1}\alpha_{k+1}(e_{k+1}^{\ T} Q_k^{\ T} e_{k+1}) v_{k+1} \ . \end{aligned}$$

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

$$A^T r_k = -\,(\bar{\phi}_{k+1}\alpha_{k+1} c_k) V_{k+1} \tag{5.3}$$

and

$$\|A^T r_k\| = \bar{\phi}_{k+1}\alpha_{k+1}|c_k| \tag{5.4}$$

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

$$R_k \bar{Q}_k^{\ T} = \bar{L}_k \tag{5.5}$$

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

$$\bar{L}_k \bar{z}_k = f_k \quad , \tag{5.6}$$

it follows that $x_k = (V_k R_k^{-1}) f_k = (V_k \bar{Q}_k^T) \bar{z}_k \equiv \bar{W}_k \bar{z}_k$. Hence, under the assumption that $V_k^T V_k = I$ we can obtain the estimate

$$\|x_k\| = \|\bar{z}_k\| \quad . \tag{5.7}$$

Note that the leading parts of $\bar{L}_k$, $\bar{Q}_k$, $\bar{W}_k$ and $\bar{z}_k$ 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 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

$$B_k^T B_k = V_k^T A^T A V_k \quad ,$$

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,

$$\|B_k\| \leq \|A\| \quad , \tag{5.8}$$

where equality will be obtained in the 2-norm for some $k \leq$ 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

$$\|R_k^{-1}\| = \|B_k^{+}\| \le \|A^{+}\| \ . \tag{5.9}$$

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

$$1 \le \|B_k\| \ \|D_k\| \le \|A\| \ \|A^{+}\| = \operatorname{cond}(A) \tag{5.10}$$

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

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 $\|D_k\|_F^2 = \|D_{K-1}\|_F^2 + \|d_k\|^2$. The individual terms in the sum $\|d_k\|^2 \equiv \sum_{i=1}^{n} \delta_{ik}^{\ 2}$ can be used further for estimating standard errors, as we show next.

5.4 Standard errors

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

$$s_i^{\ 2} \equiv \frac{\|b - Ax\|^2}{m - n} \sigma_{ii} \tag{5.11}$$

and $\sigma_{ii} \equiv e_i^{\ T} (A^T A)^{-1} e_i$ is the i-th diagonal element of $(A^T A)^{-1}$. Now from (3.3) and (3.10) we have $V_k^{\ T} A^T A V_k = R_k^{\ T} R_k$, 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 $D_n D_n^{\ T} = (A^T A)^{-1}$, and we can approximate the σ_{ii} by

$\sigma_{ii}^{(k)} \equiv e_i^T D_k D_k^T e_i$. Since $D_k D_k^T = D_{k-1} D_{k-1}^T + d_k d_k^T$, we have

$$\sigma_{ii}^{(k)} = \sigma_{ii}^{(k-1)} + \delta_{ik}^2 \ , \quad \sigma_{ii}^{(o)} \equiv 0 \quad ,$$

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

$$s_i^{(k)2} \equiv \frac{\|b - Ax_k\|^2}{l} \sigma_{ii}^{(k)}$$

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, (cond$(A) \simeq 10^3$, relative machine precision $\simeq 10^{-11}$), 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, $QA = \begin{bmatrix} R \\ 0 \end{bmatrix}$, and the quantities σ_{ii} were computed accurately using $R^T v_i = e_i$, $\sigma_{ii} = \|v_i\|^2$. 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 $s_i^{(k)2}$ approximates this variance estimate. In an analogous manner we could approximate certain covariance estimates by accumulating

$$\sigma_{ij}^{(k)} = \sigma_{ij}^{(k-1)} + \delta_{ik}\delta_{jk} , \qquad \sigma_{ij}^{(o)} \equiv 0 ,$$

for any specific pairs (i,j), and then computing

$$\frac{\|b - Ax_k\|^2}{l} \sigma_{ij}^{(k)}$$

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:

S1: Stop if $\|r_k\| \le \text{BTOL}\|b\| + \text{ATOL}\|A\|\,\|x_k\|$.

S2: Stop if $\dfrac{\|A^T r_k\|}{\|A\|\,\|r_k\|} \le \text{ATOL}$.

S3: Stop if $\text{cond}(A) \ge \text{CONLIM}$.

We can implement these rules efficiently using the estimates of $\|r_k\|$, $\|A\|_F$, 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

$$\text{ATOL} = \|A - \tilde{A}\| \,/\, \|A\|$$

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

$$\delta_k \equiv \mathrm{BTOL}\|b\| + \mathrm{ATOL}\|A\| \, \|x_k\| \quad ,$$

$$g_k \equiv \mathrm{BTOL}\|b\| \, \frac{r_k}{\delta_k} \quad ,$$

$$h_k \equiv \mathrm{ATOL}\|A\| \, \|x_k\| \, \frac{r_k}{\delta_k} \quad .$$

Then $r_k = g_k + h_k$, and

$$\left(A + \frac{h_k x_k^{\mathrm{T}}}{x_k^{\mathrm{T}} x_k} \right) x_k = b - g_k$$

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 $$E_k = - \frac{r_k r_k^{\mathrm{T}} A}{\|r_k\|^2} \quad ,$$

then $(A + E_k)^{\mathrm{T}} \tilde{r}_k = 0$, so that x_k and $\tilde{r}_k$ are the exact solution and residual for a system with A perturbed. Since $\|E_k\|_2 = \|A^{\mathrm{T}} r_k\| \, / \, \|r_k\|$, the perturbation to A will be negligible if the test in S2 is satisfied.

In our particular method it happens that $E_k x_k = 0$, since (5.3) shows that $x_k = V_k y_k$ is theoretically orthogonal to $A^T r_k$. Hence $\tilde{r}_k = r_k$, 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 $\|B_k\|_F \|D_k\|_F \simeq$ cond(A) in (5.10) temporarily levels off near some of the values of the ordered sequence σ_1/σ_1, σ_1/σ_2, ..., σ_1/σ_n, with varying numbers of iterations near each level. This tends to happen when the smaller σ_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 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 $A = U\Sigma V^{\mathrm{T}}$ where $U^{\mathrm{T}}U = V^{\mathrm{T}}V = VV^{\mathrm{T}} = I$, $\Sigma = \mathrm{diag}(\sigma_1,\sigma_2,\ldots,\sigma_n)$, and let

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

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

$$\frac{\|A\|_2 \; \|x^{(r)}\|}{\|b\|} \le \frac{\sigma_1}{\sigma_r} \equiv \mathrm{cond}(A^{(r)}).$$

In the case of LSQR, we have $\|x_k\| \le \|D_k\|_F\|b\|$ and so if rule S3 has not yet caused termination we know that $\|B_k\|_F\|x_k\| \,/\, \|b\| \le \|B_k\|_F\|D_k\|_F <$ 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 $\mathrm{rank}(A) < n$. 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

$$\frac{\|A\|\ \|x_k\|}{\|b\|} \simeq \frac{1}{\varepsilon} \tag{6.1}$$

(while $\|r_k\|$ remains of reasonable size). The estimate of cond(A) typically grows large before the growth in x_k. A moderate value of CONLIM (say $1/\varepsilon^{\frac{1}{2}}$) may therefore cause termination at a useful solution.

In some cases it can be useful to set CONLIM as large as $1/\varepsilon$ 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 $\|r_k\| \le \|b\|$ we see that the normalized vector $\bar{x}_k \equiv x_k \,/\, \|x_k\|$ will usually satisfy

$$\begin{aligned} A\bar{x}_k &= \frac{1}{\|x_k\|}(b - r_k) \\ &\simeq \varepsilon \frac{\|A\|}{\|b\|}(b - r_k) \\ &= O(\varepsilon)\|A\| \end{aligned}$$

for large enough k, and hence will lie very nearly in the null space of A. The vector $\bar{x}_k$ 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 $A^TAx = A^Tb$, 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^TAp_i. An algorithm with better numerical properties is easily derived by a slight algebraic rearrangement, making use of the intermediate vector Ap_i (e.g. Hestenes and Stiefel [11]). It is usually stated in notation similar to the following.

Algorithm CGLS

```
1.  Set $r_o = b,\ s_o = A^Tb,\ p_1 = s_o,\ \gamma_o = \|s_o\|^2,\ x_o = 0$ .
2.  For $i = 1,2,3 \ldots$ repeat the following:
        (a)   $q_i = Ap_i$
        (b)   $\alpha_i = \gamma_{i-1} / \|q_i\|^2$
        (c)   $x_i = x_{i-1} + \alpha_i p_i$
        (d)   $r_i = r_{i-1} - \alpha_i q_i$
        (e)   $s_i = A^Tr_i$
        (f)   $\gamma_i = \|s_i\|^2$
        (g)   $\beta_i = \gamma_i / \gamma_{i-1}$
        (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 s_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 ,

$$L_k = \begin{bmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \cdot & \cdot & \\ & & \cdot & \cdot \\ & & \beta_k & \alpha_k \end{bmatrix} ,$$

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

$$\begin{aligned} AV_k &= U_kL_k + \beta_{k+1}u_{k+1}e_k^{\mathrm{T}} , \\ A^{\mathrm{T}}U_k &= V_kL_k^{\mathrm{T}} . \end{aligned} \qquad (7.1)$$

Craig's method is defined by the equations

$$L_k z_k = \beta_1 e_1 , \qquad x_k = V_k z_k \quad , \tag{7.2}$$

and we can show from (7.1) that the residual vector satisfies $r_k \equiv b - AV_k z_k = -\zeta_k \beta_{k+1} u_{k+1}$ and hence $U_k^T r_k = 0$. We can therefore expect r_k to vanish (analytically) for some $k \le n$.

The vectors z_k and x_k are readily computed from

$$\zeta_k = -(\beta_k/\alpha_k)\zeta_{k-1} , \quad x_k = x_{k-1} + \zeta_k v_k \quad ,$$

where $\zeta_o \equiv -1$. Since the increments v_k form an orthogonal set there is no danger of cancellation, and the step-lengths ζ_k are bounded by $|\zeta_k| \le \|z_k\| = \|x_k\| \le \|x\|$. 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 $V_k w_k$ was computed in parallel. On termination, a suitable scalar γ_k was computed and the final solution taken to be

$$x_k = (V_k z_k) - \gamma_k (V_k w_k) \equiv V_k y_k \quad . \tag{7.3}$$

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

$$\begin{bmatrix} I & B_k \\ B_k^T & \end{bmatrix} \begin{bmatrix} t_{k+1} \\ y_k \end{bmatrix} = \begin{bmatrix} \beta_1 e_1 \\ 0 \end{bmatrix} \tag{7.4}$$

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

$$B_k^T \equiv \begin{bmatrix} \alpha_1 e_1 & \bar{L}_k \end{bmatrix} \quad , \qquad t_{k+1} \equiv \gamma_k \begin{bmatrix} 1 \\ \bar{t}_k \end{bmatrix}$$

for some γ_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

$$L_k w_k = \begin{bmatrix} 1 \\ \bar{t}_{k-1} \end{bmatrix} \quad , \qquad \gamma_k = \beta_{k+1} \zeta_k / (\beta_{k+1} \omega_k - \bar{\tau}_k) \quad ,$$

using straightforward notation. The first equation indicates that w_k can be computed by forward substitution, thus allowing $V_k w_k$ to be accumulated as the algorithm progresses. The second shows how γ_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 $V_k z_k$ and $\gamma_k V_k w_k$ were large and nearly equal. When the system $Ax = b$ is compatible the danger does not exist because each vector $V_k z_k$ is the same as in Craig's method and remains of reasonable size. We have observed in practice that $\|w_k\|$ actually diverges with increasing k, but the scalars γ_k converge to zero at a greater rate. The points $x_k = V_k z_k - \gamma_k V_k w_k$, if computed, are at all stages identical to those produced by LSQR, and for sufficiently large k are computationally just $x_k = V_k z_k$.

When b does not lie in the range of A we find that severe cancellation can occur, with the γ_k stabilizing at a moderate value but $\|z_k\|$ and $\|w_k\|$ both diverging. For this reason the extension of Craig's method must be discarded.

Note that the divergence of $\|z_k\|$ 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 B_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

$$R_k^T R_k y_k = \theta_1 e_1 \quad , \qquad x_k = V_k y_k \tag{7.5}$$

and implemented in the form $R_k^T f_k = \theta_1 e_1$, $x_k = (V_k R_k^{-1}) f_k$. 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 $\min\|B_k y_k - \beta_1 e_1\|$ by using the corresponding normal equations. (The latter are $B_k^T B_k y_k = B_k^T \beta_1 e_1 = \alpha_1 \beta_1 e_1$ and by (3.9) and (3.12) this is equivalent to (7.5a).)

Algorithm LSLQ (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 $x_k = \bar{W}_k \bar{z}_k$ where $\bar{W}_k$ is theoretically orthonormal, the intention being to avoid any possible cancellation that could occur in accumulating $x_k = D_k f_k \equiv (V_k R_k^{-1}) f_k$. 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 $x_k = \bar{W}_k \bar{z}_k$ 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 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 [4]. 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  $r_o = b,\ \ \theta_1 v_1 = A^T b,\ \ w_1 = v_1,\ \ x_o = 0$ .
2.  For $i = 1,2,3,\ldots$ repeat the following:
    (a)    $\rho_i p_i = A w_i$
    (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)    $x_i = x_{i-1} + (\lambda_i / \rho_i) w_i$
    (f)    $w_{i+1} = v_{i+1} - (\theta_{i+1} / \rho_i) w_i$ ,
    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 $\sum_{i=1}^{k} (\rho_i^2 + \theta_{i+1}^2) = \|R_k\|_F^2 = \|B_k\|_F^2$ 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^TAw_i (ignoring normalization by ρ_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 $Ax = b$, the final norms $\|r_i\|$ and $\|x_i - x\|$ are larger, by a factor approaching 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 AA^Tb (rather than b), and products of the form A^TAw_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

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

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

$$\beta_{i+1}u_{i+1} = Aw_i - \frac{\bar{\rho}_i}{\|r_{i-1}\|} r_{i-1} \tag{7.6}$$

for generating each β_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

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

where ζ_i comes from the system $L_k z_k = \beta_1 e_1$ 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.

<u>Algorithm RRLSQR</u>

```
1.  $r_o = b$,   $\beta_1 u_1 = b$,   $\alpha_1 v_1 = A^T u_1$,   $w_1 = v_1$,   $x_o = 0$,
    $\bar{\phi}_1 = \beta_1$,   $\bar{\rho}_1 = \alpha_1$ .

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

3.  (a)   $p_i = A w_i$
    (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 $\rho_i$, $c_i$, $s_i$, $\theta_{i+1}$, $\bar{\rho}_{i+1}$, $\bar{\phi}_{i+1}$ as in section 4.1, step 4.

5.  (a)   $x_i = x_{i-1} + \lambda_i w_i$
    (b)   $w_{i+1} = v_{i+1} - (\theta_{i+1} / \rho_i) w_i$

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 $A^{\mathrm{T}}Aw_i$. In practice we find that RRLSQR typically performs at least as well as LSQR, as measured by the limiting $\|x_i - x\|$ 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 $m = n$.

	Storage		Work	
	m	n	m	n
Craig ($Ax = b$ only)	u, Av	x, v	3	4
CGLS	r, q	x, p, s	2	3
LSQR	u, Av	x, v, w	3	5
RRLS	r, p	x, v, w, $A^{\mathrm{T}}p$	4	5
RRLSQR	r, u, p	x, v, w	6	5

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^Tu. (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 $A^Tu - \beta v$ 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 cond(A) and standard errors for x.) Practical implementations of CGLS and RRLS would require a further $m + n$ multiplications to compute $\|r_i\|$ and $\|x_i\|$ 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 $\varepsilon = 0.5 \times 8^{-12} \simeq 0.7 \times 10^{-11}$.

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

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

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 $\mathrm{diag}(\sigma_i^p)$ with each σ_i 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. $c = (1/m, -2/m, 3/m, \ldots, \underline{+}(m-n)/m)^{\mathrm{T}}$.

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:

$$(\sigma_i) = \left(\frac{1}{q}, \ldots, \frac{1}{q}, \quad \frac{2}{q}, \ldots, \frac{2}{q}, \quad \ldots, \quad \frac{q}{q}, \ldots, \frac{q}{q}\right).$$

[For reference, this gives: -

$$\|x\| \simeq n(n/3)^{\frac{1}{2}}, \qquad \|r\| = \|c\| \simeq \frac{m-n}{m}\,((m-n)/3)^{\frac{1}{2}},$$

$$\|A\|_F = \|D\|_F \simeq (n/3)^{\frac{1}{2}}, \qquad \mathrm{cond}(A) = \mathrm{cond}(D) \simeq (\sigma_n/\sigma_1)^p = q^p .]$$

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:

Case ($m=n=10$, cond$(A) = 10^8$)		$\log_{10}\|x_k - x\|$ $k=60$	80	100	120
1	$A = YDZ$ (as above)	-0.3	-3.3	-3.3	-3.3
2	$A = YD$	-0.5	-3.9	-3.9	-4.1
3	$A = DZ$	-2.1	-5.9	-5.9	-9.2
4	$A = D$	-9.4	-9.4	-9.4	-9.4

Since each case was a compatible system $Ax = b$, we normally would expect an error norm approaching $\|x\| \cdot \mathrm{cond}(A) \cdot \varepsilon \simeq 10^{-2}$, 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, 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 $\|x_k - x_k\|$ 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 ATOL = BTOL = ε 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 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 $\text{cond}(A)^2 = 10^{12} \simeq 1/\varepsilon$, 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 $P(20,10,1,8)$, 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

$A + \delta A$ has full column rank then it is straightforward to show that

$$\delta x = (A + \delta A)^{+}(\delta b - \delta A\hat{x}) + ((A + \delta A)^{\mathrm{T}}(A + \delta A))^{-1}\delta A^{\mathrm{T}}\hat{r} .$$

In the present example $\varepsilon \simeq 0.7 \times 10^{-11}$, $\|A\|_2 = 1$, cond$(A) = 10^6$, $\|b\| \simeq 2.4$, $\|\hat{x}\| \simeq 17$, $\|\hat{r}\| \simeq 1$. If the perturbations were caused by rounding errors in the initial data then $\|\delta A\| \simeq \varepsilon$, $\|\delta b\| \simeq \varepsilon$, 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

$$Y \equiv [Y_1, Y_2], \quad A = Y_1 DZ, \quad \hat{r} = Y_2 c,$$

we have

$$(A + \delta A)^{+} \simeq A^{+} = Z^{\mathrm{T}}D^{-1}Y_1^{\mathrm{T}},$$

and the second term in δx is effectively

$$Z^{\mathrm{T}}D^{-2}Z\,\delta A^{\mathrm{T}}Y_2 c .$$

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],

$$(A + \delta A)v \equiv fl(Y_1 fl(D fl(Zv))) = (Y_1 + \delta Y_1)(D + \delta D)(Z + \delta Z)v$$

where $\|\delta Y_1\| \simeq \varepsilon\|Y_1\|$ etc., and so

$$\delta A \equiv \delta Y_1(D + \delta D)(Z + \delta Z) + Y_1(D\delta Z + \delta D(Z + \delta Z)) .$$

Using this δA in the second term for δx effectively gives

$$Z^{\mathrm{T}}D^{-1}(I + D^{-1}Z\,\delta Z^{\mathrm{T}}D)(I + D^{-1}\delta D)\delta Y_1^{\mathrm{T}}Y_2 c$$

which is bounded above by about 7×10^{-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 $\mathrm{cond}(A) = 10^6$. 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 $\varepsilon \simeq 2.2 \times 10^{-16}$. 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 -8.6 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.

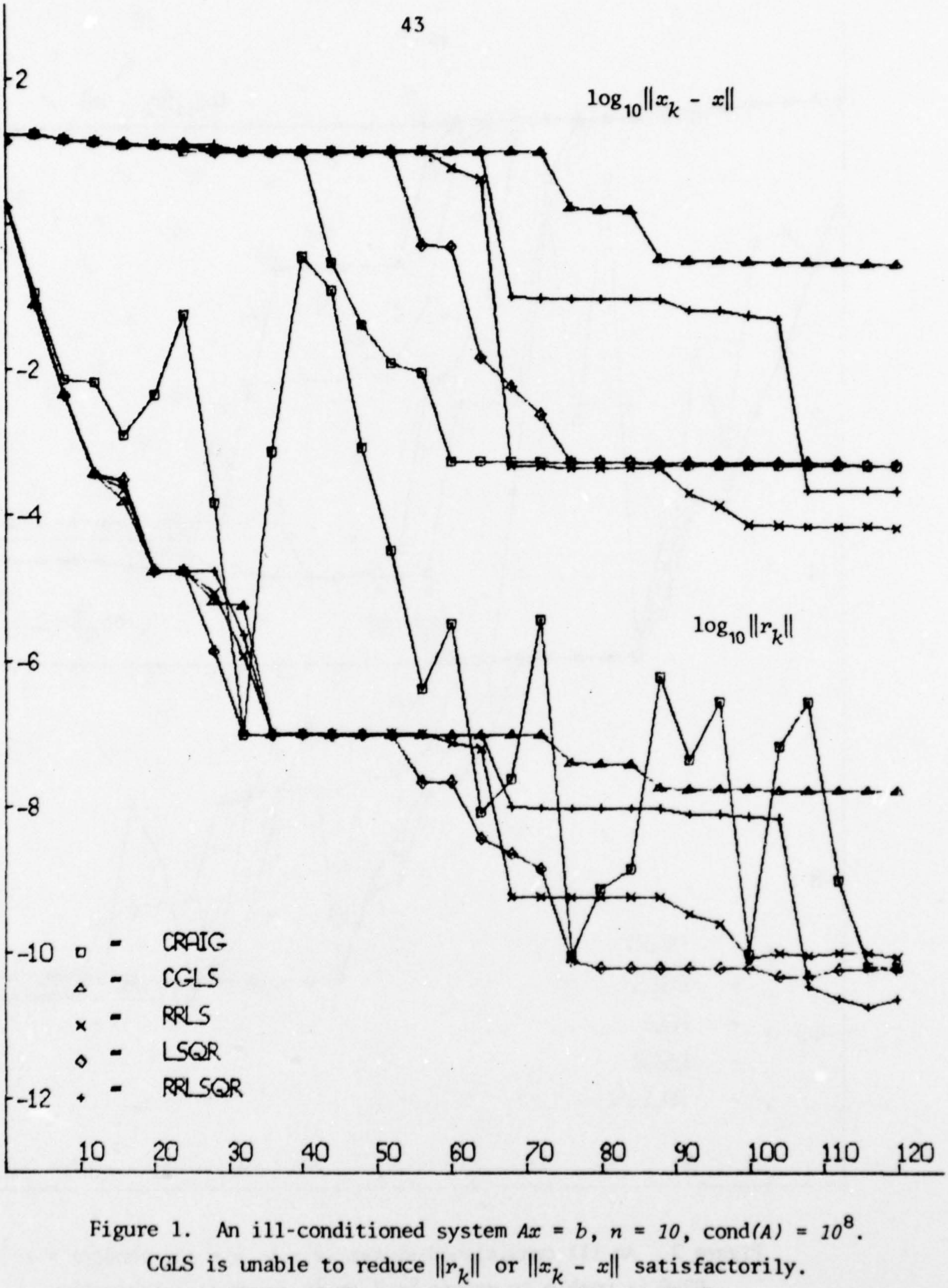


Figure 1. An ill-conditioned system $Ax = b$, $n = 10$, cond$(A) = 10^8$. CGLS is unable to reduce $\|r_k\|$ or $\|x_k - x\|$ satisfactorily. CRAIG exhibits severe fluctuations in $\|r_k\|$.

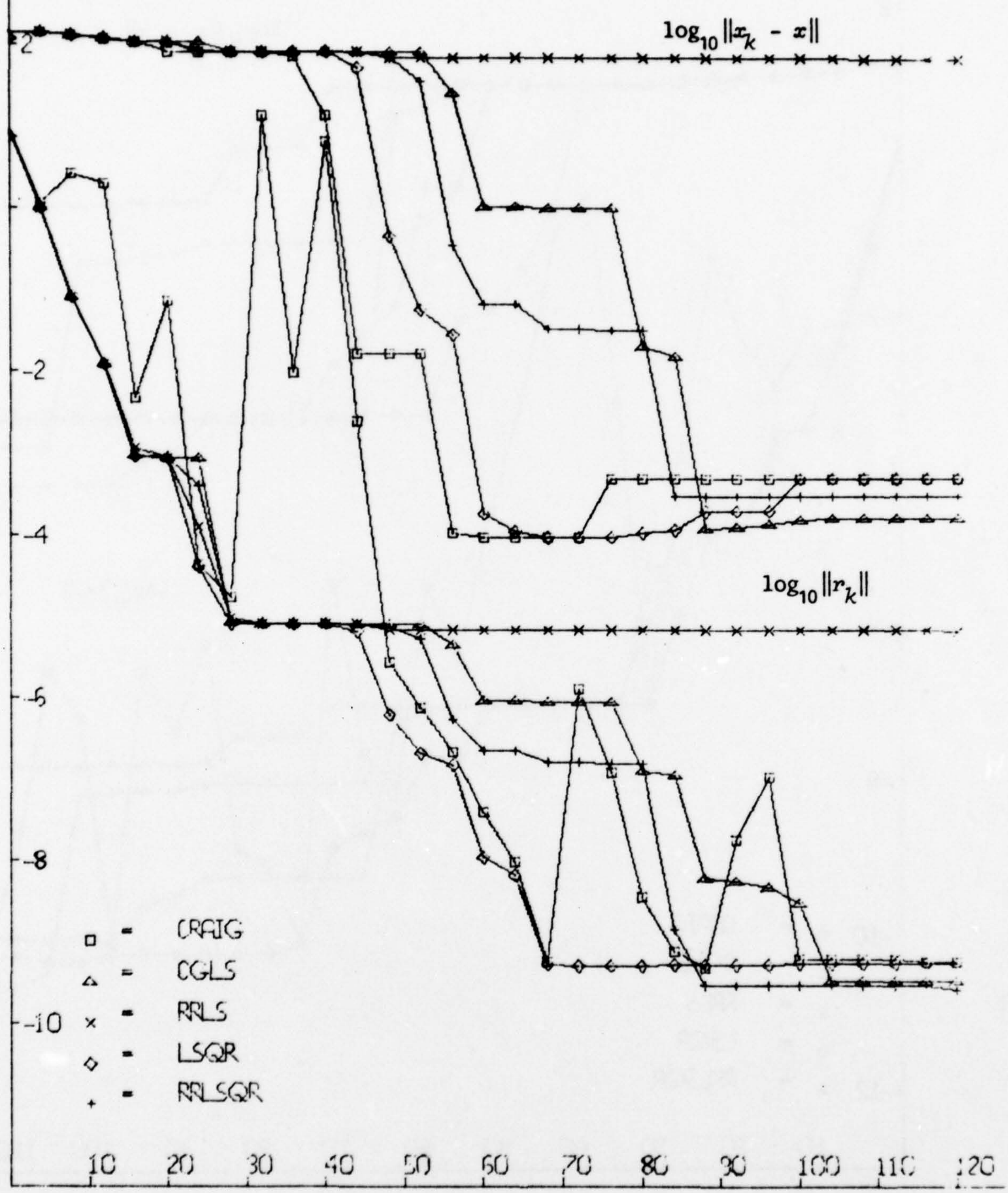


Figure 2. An ill-conditioned system $Ax = b$, $n = 40$, $\mathrm{cond}(A) = 10^7$. RRLS is unable to reduce $\|r_k\|$ or $\|x_k - x\|$ satisfactorily.

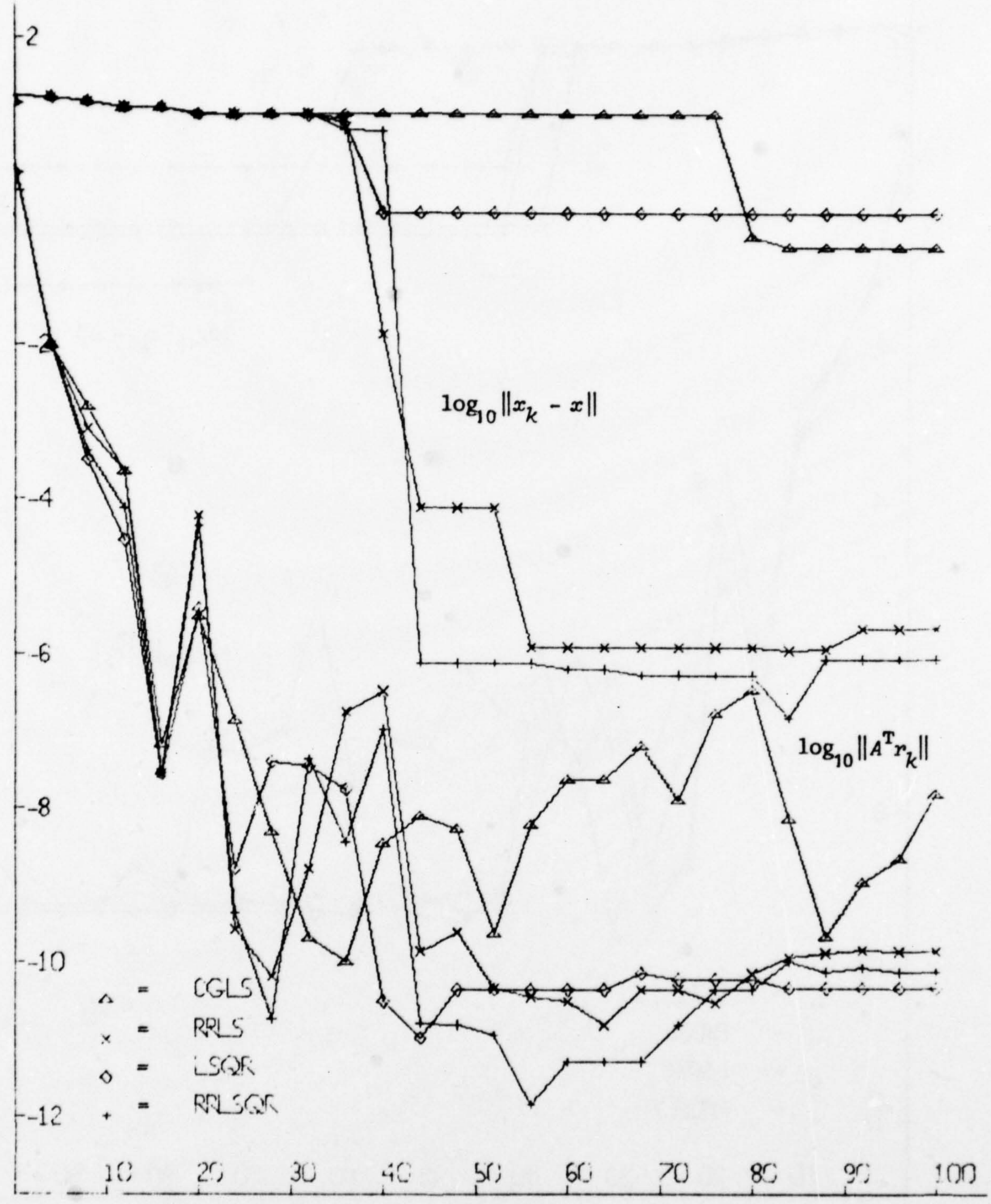


Figure 3. An ill-conditioned problem, $\min\|Ax - b\|$, $m = 20$, $n = 10$, $\text{cond}(A) = 10^6$. RRLS and RRLSQR achieve anomalously high accuracy in x_k.

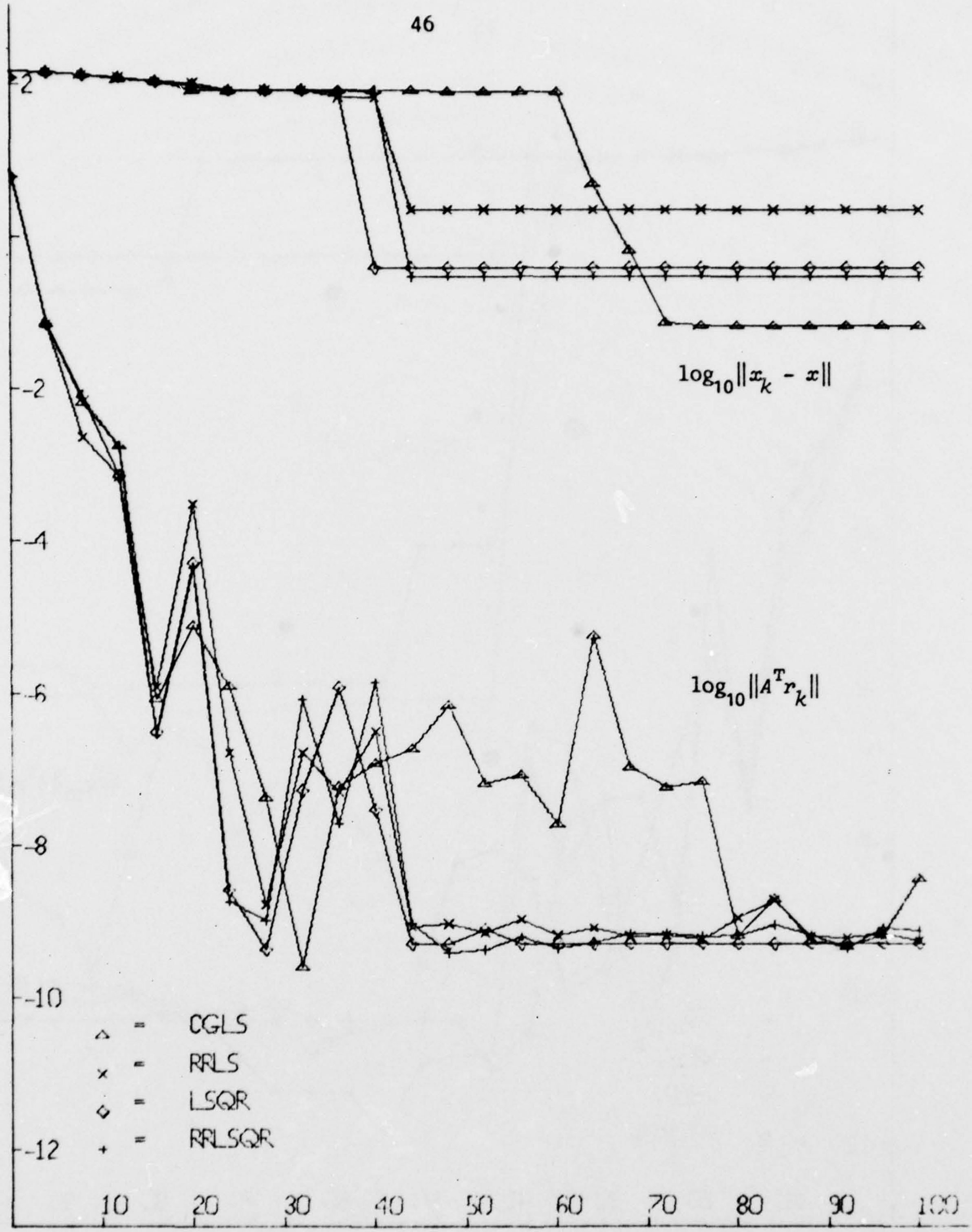


Figure 4. An ill-conditioned problem, $\min\|Ax-b\|$, $m = 80$, $n = 40$, $\text{cond}(A) = 10^6$.
All methods obtain a satisfactory solution, although
CGLS exhibits slower convergence and undesirable fluctuations in $\|A^T r_k\|$.

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) $\mathrm{cond}(A) = \sigma_{\max}/\sigma_{\min}$ is reduced, where $\sigma_{\min}$ is the smallest nonzero singular 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 $A = [B \quad C]$ 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 $\mathrm{cond}(A)$ may be large primarily because $\mathrm{cond}(B)$ is large. We now show that as long as B has full column rank, we can always transform the problem $\min\|[B \quad C]\begin{bmatrix} y \\ z \end{bmatrix} - b\|$ 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

$$\min_z \|TCz - Tb\| \tag{9.1}$$

for some matrix T and then obtain y using a direct method on the problem

$$\min_y \|By - (b - Cz)\| . \tag{9.2}$$

The form of T will depend on how much is known about B, but in all cases we shall have $\mathrm{cond}(TC) \le \mathrm{cond}(A)$.

In general, let the QR factorization of B be

$$B = [Y \quad Z]\begin{bmatrix} R \\ 0 \end{bmatrix} = YR$$

where $Q^T \equiv [Y \quad Z]$ 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):

$$\begin{aligned} T_1 &= Z^T \quad , & T_z &= I - YY^T \quad , \\ T_2 &= ZZ^T \quad , & T_4 &= I - B(B^TB)^{-1}B^T = I - BR^{-1}R^{-T}B^T \, . \end{aligned}$$

In the first case, given that Q is orthogonal, we have

$$\begin{aligned} \min\|Ax - b\| &= \min\|QAx - Qb\| \\ &= \min\left\|\begin{bmatrix} R & Y^TC \\ 0 & Z^TC \end{bmatrix}\begin{bmatrix} y \\ z \end{bmatrix} - \begin{bmatrix} Y^Tb \\ Z^Tb \end{bmatrix}\right\| \\ &= \min\|Z^TCz - Z^Tb\| \quad , \end{aligned}$$

since we can obtain y from $Ry + Y^TCz = Y^Tb$ for any z . (This is equivalent to 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:

$$\bar{Q} = \begin{bmatrix} 0 & Y^T \\ Y & ZZ^T \end{bmatrix} \quad ;$$

$$\min\|Ax - b\| = \min\|\bar{Q}\begin{bmatrix} 0 \\ A \end{bmatrix} x - \bar{Q}\begin{bmatrix} C \\ b \end{bmatrix}\|$$

$$= \min\|\begin{bmatrix} R & Y^{\mathrm{T}}C \\ 0 & ZZ^{\mathrm{T}}C \end{bmatrix}\begin{bmatrix} y \\ z \end{bmatrix} - \begin{bmatrix} Y^{\mathrm{T}}b \\ ZZ^{\mathrm{T}}b \end{bmatrix}\| \; .$$

Again it is clear that cond$(A) \geq$ cond$(ZZ^{\mathrm{T}}C)$. In fact the operator $T_2 = ZZ^{\mathrm{T}}$ 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 $T_1 = Z^{\mathrm{T}}$ should be used whenever possible, particularly if 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 $T = T^2 = T^{\mathrm{T}}$ 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 TCv_i and just $C^{\mathrm{T}}u_i$ (not $C^{\mathrm{T}}Tu_i$).

Note also that the product TCv_i may be regarded as the original matrix A operating on a vector, thus

$$TCv_i = Cv_i - Bz_i = A\begin{bmatrix} -z_i \\ v_i \end{bmatrix}$$

where z_i solves the least-squares problem $\min\|Bz_i - Cv_i\|$. Each iteration is therefore essentially the same as for the original problem $\min\|Ax - b\|$ 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 $Mz_i = r_i$ (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, $A = [e \quad C]$ where e is a column of 1's. Writing $x = \begin{bmatrix} \mu \\ z \end{bmatrix}$, we would then use $T = (I - \frac{1}{m} ee^{\mathrm{T}})$ in (9.1), solve for z and then estimate the mean using $\mu = e^{\mathrm{T}}(b - Cz)/m$. This is nothing more than the well known practice of "subtracting off the mean" (note that $Tb = b - \overline{\mu}e$, where $\overline{\mu} = e^{\mathrm{T}}b/m$) 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 $(B^{\mathrm{T}}B + C^{\mathrm{T}}C)x = [B^{\mathrm{T}} \quad C^{\mathrm{T}}]b$ to obtain the equivalent system

$$(I + B^{-\mathrm{T}}C^{\mathrm{T}}CB^{-1})y = [I \quad B^{-\mathrm{T}}C^{\mathrm{T}}]b$$

(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

$$\min \left\| \begin{bmatrix} I \\ CB^{-1} \end{bmatrix} y - b \right\|$$

by an algorithm such as CGLS or LSQR

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

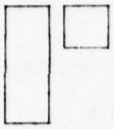
The simplest form of preconditioning is column scaling, in which the original problem is replaced by

$$\min\|ADy - b\|, \qquad x = Dy \tag{9.3}$$

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

$A = LU =$ 

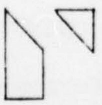
in which the left-hand factor L has better condition than A, and the right-hand factor U is nonsingular. We assume that U will be a sparse matrix and that systems of equations involving U and U^T can be solved efficiently. There are now two equivalent but distinct possibilities:

$$\min\|Ly - b\|, \qquad \text{solve } Ux = y \; ; \tag{9.4a}$$

$$\min\|AU^{-1}y - b\|, \quad \text{solve } Ux = y \; . \tag{9.4b}$$

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:

$P_1AP_2 = LU =$  .

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 cond(L) being kept small so that the normal equations $L^TLy = L^Tb$ corresponding to (9.4a) could be solved without serious numerical difficulty. In the present context it is again 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 preconditioning 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 subset of the parameters x may be required to sum to zero.) Thus with appropriate C and d it will be known in advance that

$$\min\|Ax - b\| = \min\left\|\begin{bmatrix} A \\ C \end{bmatrix} x - \begin{bmatrix} b \\ d \end{bmatrix}\right\| \quad ,$$

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

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

$$R^{\mathrm{T}}y = d \qquad , \qquad c = b - AYy ,$$

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

$$\min\|AZz - c\| \tag{9.6}$$

can be solved by a conjugate-gradient method, and the solution to the original problem is then $x = Q^{\mathrm{T}}\begin{bmatrix} y \\ z \end{bmatrix}$. 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 $Z = Q^{\mathrm{T}}\begin{bmatrix} 0 \\ I \end{bmatrix}$. Similarly, $c = b - AQ^{\mathrm{T}}\begin{bmatrix} y \\ 0 \end{bmatrix}$.

This procedure reduces the dimension of the least-squares problem to be solved, and any iterate z_k will give a point $x_k = Q^{\mathrm{T}}\begin{bmatrix} y \\ z_k \end{bmatrix}$ that satisfies $Cx_k = d$. If the constraints are nearly dependent, the solution of $R^{\mathrm{T}}y = d$ 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 cond$(AZ) \le$ cond(A) regardless of cond(C).

If C is large and sparse it would be more efficient to compute a stable triangular factorization of the form

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

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 $Z = L^{\mathrm{T}}\begin{bmatrix} 0 \\ I \end{bmatrix}$ and the bound on the condition of the reduced problem is weakened to cond$(AZ) \leq$ cond(A) cond$(Z) \leq$ cond(A) cond(L). This may not be of serious consequence since in practice the interchange strategy can usually keep 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

$$\min\|Ax - b\| , \qquad A = A_O D \tag{10.1}$$

in which A_O 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_O and to exclude some containing insufficient data. Ignoring excluded columns, A is effectively 1033 by 320 and of full rank. Since $\|b\| \simeq 6000$ and $\|r\| = \|b - Ax\| \simeq 0.75$ 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 cond(A). After 1600 iterations the run was terminated with $\|r_k\|$ and $\|A^T r_k\|/(\|A\|\ \|r_k\|)$ 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 cond$(A) \simeq 10^5$ 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 $\|r_k\|$, $\|A^T r_k\|$ and $\|x_k\|$ were correct to eight, five, and eight digits respectively, a very satisfactory result. The value $\|B_k\|_F \simeq 56$ over-estimated $\|A\|_F = \sqrt{320} \simeq 18$ 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

$$\ldots L_3 L_2 L_1 \begin{bmatrix} 0 \\ A \end{bmatrix} = \begin{bmatrix} U \\ 0 \end{bmatrix} \qquad (10.2)$$

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

$$\begin{bmatrix} 1 & \\ \lambda_j & 1 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} & 1 \\ 1 & \lambda_j \end{bmatrix} \qquad (10.3)$$

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

$$L = [0 \quad I]L_1^{-1}L_2^{-1}L_3^{-1}\ldots\begin{bmatrix} I \\ 0 \end{bmatrix}.$$

The multipliers were constrained to satisfy $|\lambda_j| \leq 1/\tau$ 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 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.

Operator	Condition (estimate)	Total iterations	Time per iteration	Total time
A	$\geq 10^5$	≥ 2000	1.0	≥ 2000
L	3500	390	2.4	920
AU^{-1}	3500	400	1.3	520

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, cond(L) is not directly related to cond(A). (For example, L itself is independent of the column scale of A.) In this case, cond$(L) \simeq 3500$ 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 λ_j would still be of order 1 the reduction in cond(L) could prove to be only slight.

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

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 cond$(AU^{-1}) \leq$ cond(A) cond$(U) \simeq 10^8$ 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 cond(A) and 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_o and to the intrinsic simplicity of the relevant subroutines. This could be expected in other practical applications.

Figure 5 compares the decrease of $\|r_k\|$ 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_o that the nonunit coefficients could be translated to give a more favorable distribution around zero (Woodward [28]). Algebraically, the columns of A_o occurred in pairs of the form $[e \quad a]$ in which the vectors e and a possessed the same sparsity pattern (with e containing unit coefficients). The shift $\bar{a} = a - \mu e$ was applied to each such pair, where $\mu = e^T a/e^T e$ was the mean of the nonzeros in a. This amounted to a column transformation

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

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

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

$$\min\|\bar{A}\bar{x} - b\| \ , \qquad \bar{A} = \bar{A}_o\bar{D} \tag{10.4}$$

proved to be greatly accelerated, as shown in Figure 6. Compared to the original problem (10.1), the reduction in cond$(\bar{A})$ was clearly important. From the sharp drops in $\|r_k\|$ it seems that another contributing factor was an increased repetition of singular values. Unfortunately a further "automatic" preconditioning via $\bar{A} = \bar{L}\bar{U}$ appeared to negate the latter effect, as shown in Figure 6. The smooth decrease in $\|r_k\|$ 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.

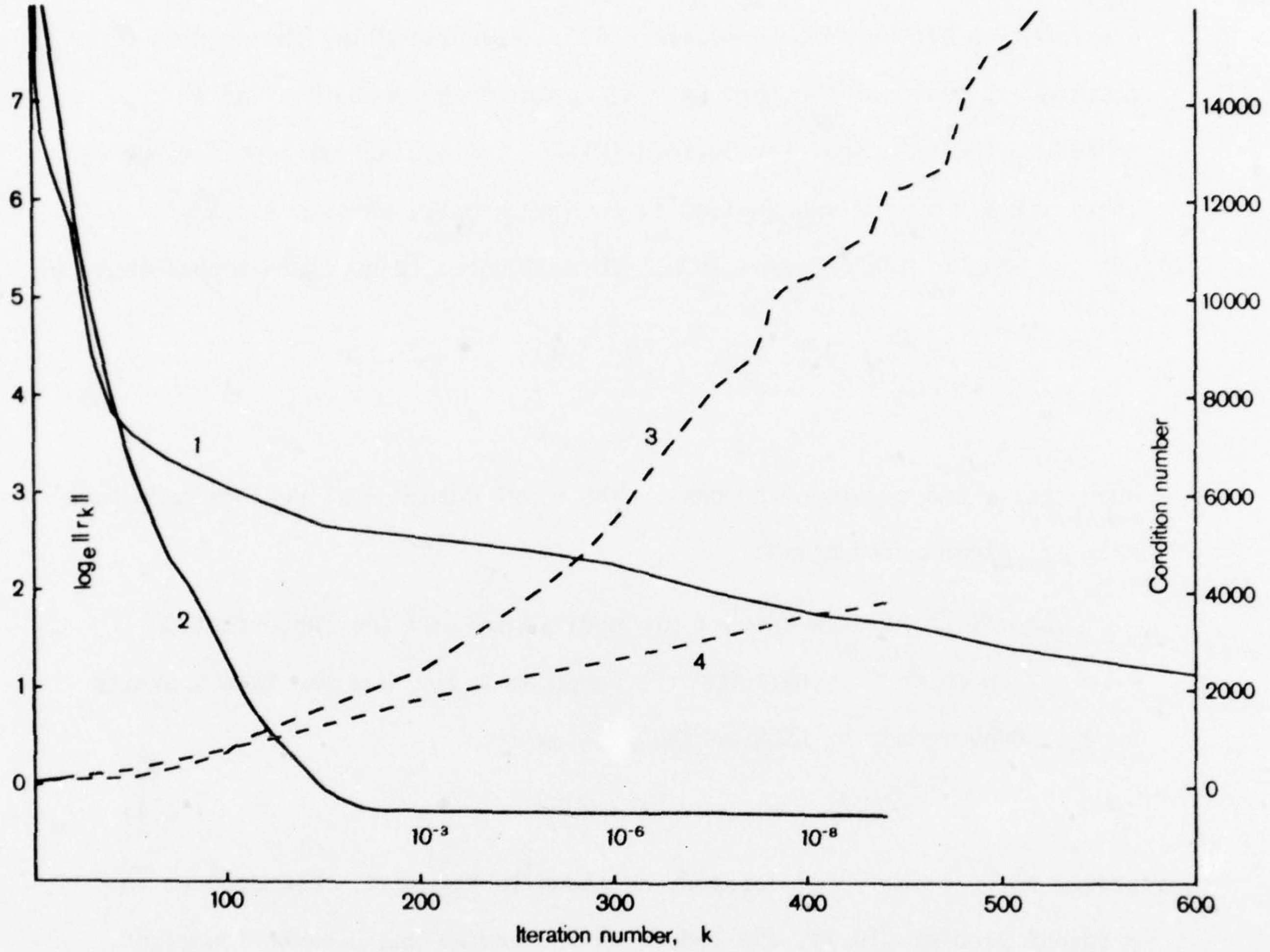


Figure 5. Comparison of $\min\|Ax - b\|$ and $\min\|AU^{-1}y - b\|$ for problem (10.1).

1 = $\log_e\|r_k\|$ for operator A.

2 = $\log_e\|r_k\|$ for operator AU^{-1}.

3 = estimate of cond(A).

4 = estimate of 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}.

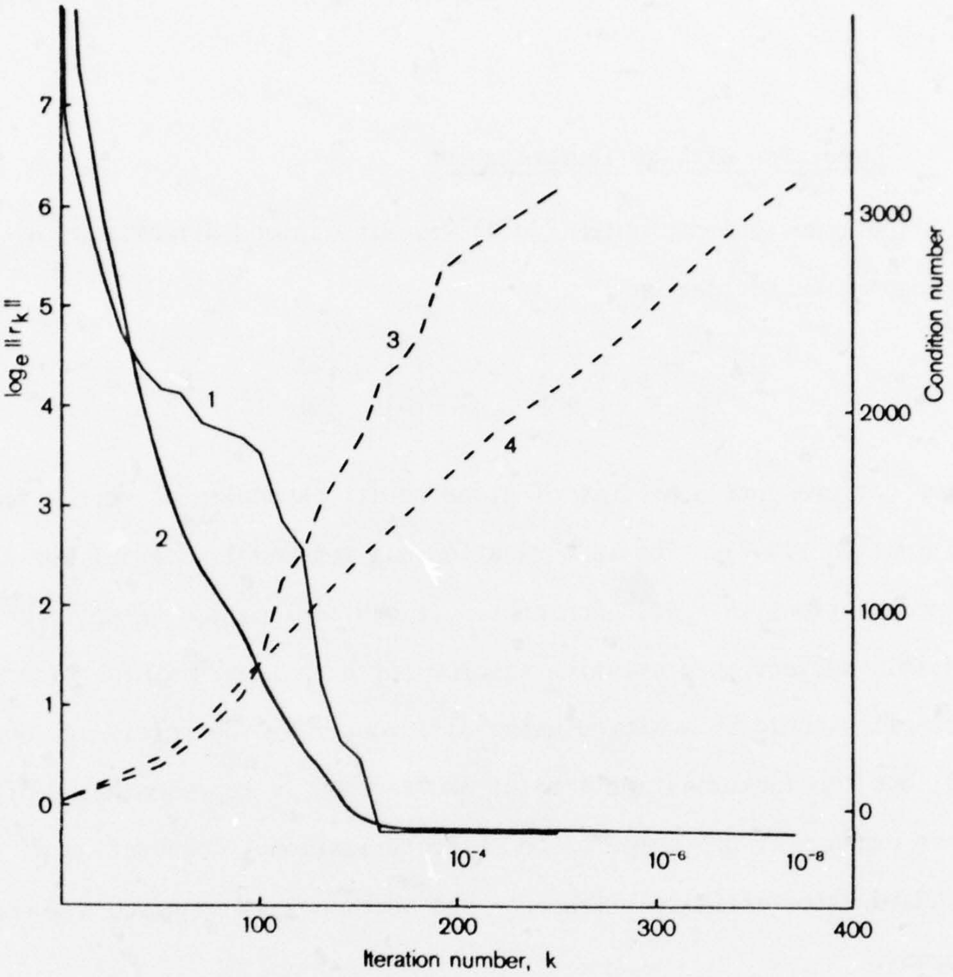


Figure 6. Similar comparison for the transformed problem (10.2).

1 = $\log \|r_k\|$ for operator $\bar{A}$.

2 = $\log \|r_k\|$ for operators $\overline{AU}^{-1}$, $\bar{L}$.

3 = estimate of cond$(\bar{A})$.

4 = estimate of cond$(\overline{AU}^{-1})$ = cond$(\bar{L})$.

Operator	Condition	Total iterations	Time per iteration	Total time
$\bar{A}$	*3100*	*250*	*1.0*	*250*
$\bar{L}$	*3200*	*376*	*2.4*	*880*
$\bar{A}\bar{U}^{-1}$	*3100*	*371*	*1.3*	*460*

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

$$Q \begin{bmatrix} 0 \\ \\ \bar{A} \end{bmatrix} = \begin{bmatrix} R \\ \\ 0 \end{bmatrix} , \qquad Q \begin{bmatrix} 0 \\ \\ b \end{bmatrix} = \begin{bmatrix} c \\ \\ d \end{bmatrix} , \qquad R\bar{x} = c ,$$

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 $\bar{A}$ in their natural order. Some ecconomies 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 $\bar{A}$ in Table 2 shows that LSQR furnished an equally accurate $\bar{x}$ 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 *825*.

In solving this case, LSQR obtained the estimate cond$(\bar{A}) \approx 3200$, 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 $\|A\|$, 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

1. BJÖRCK, Å. Methods for sparse linear least squares problems. In BUNCH, J.R. and ROSE, D.J. (Eds.), Sparse Matrix Computations, Academic Press, New York, 1976, 177-199.

2. BJÖRCK, Å. AND ELFVING, T. Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations. Research Report LiTH-MAT-R-1978-5, Department of Mathematics, Linköping University, Sweden, 1978.

3. CHANDRA, R. Conjugate gradient methods for partial differential equations. Research Report 129, Department of Computer Science, Yale University, 1978.

4. CHEN, Y.T. Iterative methods for linear least-squares problems. Research Report CS-75-04, Department of Computer Science, University of Waterloo, 1975.

5. CONCUS, P., GOLUB, G.H. AND O'LEARY, D.P. A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations. In BUNCH, J.R. and ROSE, D.J. (Eds.), Sparse Matrix Computations, Academic Press, New York, 1976, 309-332.

6. ELFVING, T. On the conjugate gradient method for solving linear least-squares problems. Research Report LiTH-MAT-R-1978-3, Department of Mathematics, Linköping University, Sweden, 1978.

7. FADDEEV, D.K. AND FADDEEVA, V.N. Computational Methods of Linear Algebra. Freeman and Co., London, 1963.

8. GENTLEMAN, M.W. Regression problems and the QR-decomposition. Bulletin of the Institute of Mathematics and Its Applications 10 (1974), 195-197.

9. GOLUB, G.H. Numerical methods for solving linear least-squares problems. Numerische Mathematik 7 (1965), 206-216.

10. GOLUB, G.H. AND KAHAN, W. Calculating the singular values and pseudo-inverse of a matrix. SIAM J. Numer. Anal. 2 (1965), 205-224.

11. HESTENES, M.R. AND STIEFEL, E. Methods of conjugate gradients for solving linear systems. J. Res. Nat. Bur. Standards 49 (1952), 409-436.

12. HOUSEHOLDER, A.S. Terminating and non-terminating iterations for solving linear systems. SIAM J. Appl. Math. 3 (1955), 67-72.

13. LANCZOS, C. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. J. Res. Nat. Bur. Standards 45 (1950), 255-282.

14. LÄUCHLI, P. Jordan-elimination und Ausgleichung nach kleinsten Quadraten. Numerische Mathematik 3 (1961), 226-240.

15. LEWIS, J.G. Algorithms for sparse matrix eigenvalue problems. Research Report STAN-CS-77-595, Stanford University, 1977.

16. NASHED, M.Z. Aspects of generalized inverses in analysis and regularization. In NASHED, M.Z. (Ed.), Generalized Inverses and Applications, Academic Press, New York, 1976, 193-244.

17. NELDER, J.A. GLIM Manual. Numerical Algorithms Group, 13 Banbury Road, Oxford, 1975.

18. PAIGE, C.C. Bidiagonalization of matrices and solution of linear equations. SIAM J. Numer. Anal. 11 (1974), 197-209.

19. PAIGE, C.C. Error analysis of the Lanczos algorithm for tridiagonalizing a symmetric matrix. J. Inst. Maths. Applics. 18 (1976), 341-349.

20. PAIGE, C.C. AND SAUNDERS, M.A. Solution of sparse indefinite systems of equations and least-squares problems. Research Report STAN-CS-73-399, Stanford University, 1973.

21. PAIGE, C.C. AND SAUNDERS, M.A. Solution of sparse indefinite systems of linear equations. SIAM J. Numer. Anal. 12 (1975), 617-629.

22. PETERS, G. AND WILKINSON, J.H. The least-squares problem and pseudo inverses. Computer J. 13 (1970), 309-316.

23. REILLY, W.I. Adjustment of gravity meter observations. New Zealand J. of Geology and Geophysics 13 (1970), 697-702.

24. VAN DER SLUIS, A. Condition numbers and equilibration of matrices. Numerische Mathematik 14 (1969), 14-23.

25. STEWART, G.W. Research, development and LINPACK. In RICE, J.R. (Ed.), Mathematical Software III, Academic Press, New York, 1977, 1-14.

26. WILKINSON, G.N. A general recursive procedure for analysis of variance. Biometrika 57 (1970), 19-46.

27. WILKINSON, J.H. The Algebraic Eigenvalue Problem. Oxford University Press (Clarendon), London and New York, 1965.

28. WOODWARD, D.J. Private communication. Geophysics Division, DSIR, Wellington, 1978.

APPENDIX A: FORTRAN PROGRAMS

Listings are given here of the following Fortran routines:

1. LSQR - the principal subroutine inplementing the sparse least-squares algorithm.

2. Auxiliary routines required by LSQR.

   NORMLZ - 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 $p = (I - 2zz^{T})v$, where $||z|| = 1$.

   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:

ABS	<---->	DABS
REAL	<---->	REAL*8 or DOUBLE PRECISION
SQRT	<---->	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.

[1] LAWSON, C.L., HANSON, R.J., KINCAID, D.R. AND KROGH, F.T. Basic Linear Algebra Subprograms. Research Report SAND77-0898, Sandia Laboratories, Albuquerque, New Mexico, 1977.

[2] RYDER, B.G. AND HALL, A.D. The PFORT Verifier. Computing Science Technical Report 12, Bell Telephone Laboratories, Murray Hill, New Jersey, 1973 (revised 1975).

```
      SUBROUTINE LSQR( M,N,MAX,B,P,U,V,W,X,SE,
     1                 ATOL,BTOL,CONLIM,ITNLIM,NOUT,
     2                 ISTOP,RNORM,ANORM,ACOND )
C
      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,CONLIM,RNORM,ANORM,ACOND
C     ------------------------------------------------------------------
C
C     LSQR  FINDS A SOLUTION  X  TO THE FOLLOWING PROBLEMS...
C
C     (UNSYMMETRIC EQUATIONS)  SOLVE      A.X = B,
C
C     (LINEAR LEAST SQUARES)   MINIMIZE  NORM(R)  WHERE  R = B - A.X,
C
C     IN WHICH  A  IS A REAL MATRIX OF DIMENSIONS  M BY N  AND  B  IS A
C     GIVEN  M-VECTOR.  NORMALLY RANK(A) SHOULD BE  N,  WITH  M >= N,
C     BUT THIS IS NOT ESSENTIAL (E.G. SOME OF THE ROWS AND COLUMNS OF
C     A  MAY BE ZERO).
C
C     THE MATRIX  A  IS INTENDED TO BE LARGE AND SPARSE.  IT IS ACCESSED
C     BY MEANS OF TWO SUBROUTINE CALLS OF THE FORM
C
C               CALL ATIMES( V,P,M,N )
C               CALL ATRANS( U,P,M,N )
C
C     WHICH MUST RETURN THE PRODUCTS  P = A.V  AND  P = A(TRANSPOSE).U
C     FOR GIVEN INPUT VECTORS  V  AND  U.
C
C     SUBROUTINES  ATIMES  AND  ATRANS  ARE TO BE SUPPLIED BY THE USER.
C     THEY MUST NOT ALTER  V  OR  U  RESPECTIVELY.  THE LENGTH OF THE
C     OUTPUT VECTOR  P  IS RESPECTIVELY  M  AND  N,  BUT IN EITHER CASE
C     THE ARRAY PARAMETER  P  MAY BE DECLARED TO HAVE LENGTH  MAX(M,N).
C     (THIS MAY BE USEFUL IF  P  IS USED FOR WORKSPACE BEFORE EXIT.)
C
C     NOTE.  THE NUMBER OF ITERATIONS REQUIRED BY  LSQR  DEPENDS
C     CRITICALLY ON THE CONDITION NUMBER OF THE MATRIX  A.  POOR
C     SCALING OF THE COLUMNS OF  A  SHOULD BE AVOIDED.  THIS CAN
C     USUALLY BE TAKEN CARE OF WHEN PROGRAMMING SUBROUTINES  ATIMES
C     AND  ATRANS.  IN THE ABSENCE OF BETTER INFORMATION THE NONZERO
C     COLUMNS OF  A  SHOULD BE SCALED SO THAT THEY ALL HAVE THE SAME
C     EUCLIDEAN NORM (USUALLY  1.0).
C
C
C     PARAMETERS
C     ----------
C
C     M       INPUT      THE NUMBER OF ROWS IN  A.
C
C     N       INPUT      THE NUMBER OF COLUMNS IN  A.
C
C     MAX     INPUT      MAX(M,N).  USED ONLY TO SPECIFY THE LENGTH
C                        OF  P  IN THE DIMENSION STATEMENT ABOVE.
```

```
C
C     B(M)     INPUT       THE RHS VECTOR OF LENGTH  M.
C
C     P(MAX)   OUTPUT      RETURNS THE VECTOR  A(TRANSPOSE).R,  WHERE
C                          R = B - A.X  IS THE RESIDUAL VECTOR CORRES-
C                          PONDING TO THE COMPUTED SOLUTION  X.
C
C     U(M)     OUTPUT      RETURNS THE RESIDUAL VECTOR  R = B - A.X.
C
C     V(N)     WORKSPACE
C
C     W(N)     WORKSPACE
C
C     X(N)     OUTPUT      RETURNS THE COMPUTED SOLUTION  X.
C
C     SE(N)    OUTPUT      RETURNS STANDARD ERROR ESTIMATES FOR THE
C                          COMPONENTS OF  X.
C
C              NOTE.  THE FOLLOWING DISCUSSION OF TOLERANCES IS
C              IN TERMS OF A QUANTITY  EPS  WHICH STANDS FOR THE
C              RELATIVE PRECISION OF FLOATING-POINT ARITHMETIC
C              ON THE MACHINE BEING USED.  TYPICAL VALUES ARE
C
C              BURROUGHS B6700      SINGLE PRECISION,  EPS = 1.0E-11
C              CDC 6600,7600        SINGLE PRECISION,  EPS = 1.0E-14
C              IBM 360,370          SINGLE PRECISION,  EPS = 1.0E-6
C                                   DOUBLE PRECISION,  EPS = 1.0D-16
C              UNIVAC 1100 SERIES   SINGLE PRECISION,  EPS = 1.0E-8
C                                   DOUBLE PRECISION,  EPS = 1.0D-18
C
C              (ALL VALUES APPROXIMATE).
C
C     ATOL     INPUT       AN ESTIMATE OF THE RELATIVE ACCURACY OF
C                          THE DATA DEFINING THE MATRIX  A.
C                          ATOL  WILL NORMALLY BE IN THE RANGE
C                          EPS  TO  SQRT(EPS).
C                          SUGGESTED VALUE --   ATOL = 1000*EPS
C
C     BTOL     INPUT       AN ESTIMATE OF THE RELATIVE ACCURACY OF
C                          THE DATA DEFINING THE RHS VECTOR  B.
C                          BTOL  WILL NORMALLY BE IN THE RANGE
C                          EPS  TO  SQRT(EPS).
C                          SUGGESTED VALUE --   BTOL = 1000*EPS
C
C     CONLIM   INPUT       AN UPPER LIMIT ON  COND(A),  THE APPARENT
C                          CONDITION NUMBER OF THE MATRIX  A  (IGNORING
C                          KNOWN SINGULARITIES).  ITERATIONS WILL
C                          TERMINATE IF A COMPUTED ESTIMATE OF  COND(A)
C                          EXCEEDS  CONLIM.  THIS IS INTENDED TO PREVENT
C                          CERTAIN SMALL OR ZERO SINGULAR VALUES OF  A
C                          FROM COMING INTO EFFECT AND CAUSING UNWANTED
C                          GROWTH IN THE COMPUTED  X.  HENCE IT MAY ASSIST
C                          IN REGULARIZING ILL-CONDITIONED SYSTEMS.
```

```
C
C                        CONLIM  WILL NORMALLY BE IN THE RANGE
C                        1ØØØ  TO  1/EPS.
C                        SUGGESTED VALUE --
C                        CONLIM = 1/(1ØØØ*EPS)  FOR COMPATIBLE SYSTEMS,
C                        CONLIM = 1/(1Ø*SQRT(EPS))  FOR LEAST SQUARES.
C
C             NOTE.  IF THE USER IS NOT CONCERNED ABOUT THE PARAMETERS
C             ATOL, BTOL  AND  CONLIM,  ANY OR ALL OF THEM MAY BE SET
C             TO ZERO.
C
C     ITNLIM  INPUT      AN UPPER LIMIT ON THE NUMBER OF ITERATIONS.
C                        SUGGESTED VALUE --
C                        ITNLIM = N/2      FOR WELL-CONDITIONED SYSTEMS,
C                        ITNLIM = 4*N      OTHERWISE.
C
C     NOUT    INPUT      LINEPRINTER UNIT NUMBER.  IF POSITIVE,
C                        A SUMMARY WILL BE PRINTED ON UNIT  NOUT.
C
C     ISTOP   OUTPUT     AN INTEGER GIVING THE REASON FOR TERMINATION...
C
C               Ø        X = Ø  IS THE EXACT SOLUTION.
C                        NO ITERATIONS WERE PERFORMED.
C
C               1        THE EQUATIONS  A.X = B  ARE PROBABLY
C                        COMPATIBLE.  NORM(R)  IS SUFFICIENTLY SMALL
C                        GIVEN THE VALUES OF  ATOL  AND  BTOL.
C
C               2        THE SYSTEM  A.X = B  IS PROBABLY NOT
C                        COMPATIBLE.  A LEAST-SQUARES SOLUTION HAS
C                        BEEN OBTAINED, FOR WHICH  NORM(A.R)  IS
C                        SUFFICIENTLY SMALL GIVEN THE VALUE OF  ATOL.
C
C               3        THE SYSTEM IS MORE ILL-CONDITIONED THAN
C                        EXPECTED.  AN ESTIMATE OF  COND(A)  HAS
C                        EXCEEDED  CONLIM.
C
C               4        THE ITERATION LIMIT  ITNLIM  WAS REACHED.
C
C               5        THE EQUATIONS  A.X = B  ARE PROBABLY
C                        COMPATIBLE.  NORM(R)  IS AS SMALL AS SEEMS
C                        REASONABLE ON THIS MACHINE.
C
C               6        THE SYSTEM  A.X = B  IS PROBABLY NOT
C                        COMPATIBLE.  A LEAST-SQUARES SOLUTION HAS
C                        BEEN OBTAINED, FOR WHICH  NORM(A.R)  IS AS
C                        SMALL AS SEEMS REASONABLE ON THIS MACHINE.
C
C               7        COND(A)  SEEMS TO BE SO LARGE THAT THERE IS
C                        NOT MUCH POINT IN DOING FURTHER ITERATIONS,
C                        GIVEN THE PRECISION OF THIS MACHINE.
C
C     RNORM   OUTPUT     NORM(R) = SQRT(R.R),  THE NORM OF THE FINAL
C                        RESIDUAL VECTOR  R = B - A.X.
```

```
C
C     ANORM   OUTPUT     AN ESTIMATE OF THE FROBENIUS NORM OF  A.
C                        THIS IS THE SQUARE-ROOT OF THE SUM OF SQUARES
C                        OF THE ELEMENTS OF  A.  IF THE COLUMNS OF  A
C                        HAVE ALL BEEN SCALED TO HAVE LENGTH  1.0,
C                        ANORM  SHOULD INCREASE TO ROUGHLY  SQRT(N).
C                        A RADICALLY DIFFERENT VALUE FOR  ANORM  MAY
C                        INDICATE AN ERROR IN ONE OF THE SUBROUTINES
C                        ATIMES  OR  ATRANS.
C
C     ACOND   OUTPUT     AN ESTIMATE OF  COND(A),  THE CONDITION
C                        NUMBER OF  A.  A VERY HIGH VALUE OF  ACOND
C                        MAY AGAIN INDICATE AN ERROR IN SUBROUTINES
C                        ATIMES  OR  ATRANS.
C
C
C     TO CHANGE PRECISION
C     -------------------
C
C                IF SUBSTITUTE BLAS ROUTINES ARE IN USE, ALTER THE WORDS
C                        ABS, REAL  , SQRT
C                THROUGHOUT ROUTINES  LSQR, NORMLZ, SAXPY, SNRM2.
C
C                IF AUTHENTIC BLAS ROUTINES ARE IN USE, ALTER THE WORDS
C                        ABS, REAL  , SAXPY, SNRM2, SQRT
C                THROUGHOUT ROUTINES  LSQR, NORMLZ.
C
C
C     REFERENCE  A BIDIAGONALIZATION ALGORITHM FOR SPARSE
C     ---------  LINEAR EQUATIONS AND LEAST-SQUARES PROBLEMS,
C                TECHNICAL REPORT SOL 78-19, DEPARTMENT OF
C                OPERATIONS RESEARCH, STANFORD UNIVERSITY, 1978
C
C
C     AUTHORS    C.C. PAIGE                        M.A. SAUNDERS
C     -------    MCGILL UNIVERSITY, CANADA         DSIR, NEW ZEALAND
C
C
C     LSQR.      THIS VERSION DATED 2 OCTOBER 1978.
C
C     SUBROUTINES AND FUNCTIONS
C
C     USER       ATIMES,ATRANS
C     LSQR       NORMLZ
C     BLAS       SAXPY
C     FORTRAN    ABS,MOD,SQRT
C     -------------------------------------------------------------
```

```
C
C     FUNCTIONS AND LOCAL VARIABLES
C
      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,SQRT,T,TEST1,TEST2,TEST3,
     4           THETA,T1,T2,T3,XNORM,XXNORM,Z,ZBAR,ZERO
C
C
C     INITIALIZE
C
      ZERO = 0
      ONE  = 1
      IF (NOUT .GT. 0) WRITE(NOUT, 1000) M,N,ATOL,BTOL,CONLIM,ITNLIM
      CALL NORMLZ( B,U,M,BETA )
      CALL ATRANS( U,P,M,N )
      CALL NORMLZ( P,V,N,ALFA )
C
      DO 20 I = 1, N
         W(I) = V(I)
         X(I) = ZERO
        SE(I) = ZERO
   20 CONTINUE
C
      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
C
      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
```

```
C
C     -----------------------------------------------------------------
C     MAIN ITERATION LOOP
C     -----------------------------------------------------------------
C
C     PERFORM NEXT STEP OF THE BIDIAGONALIZATION
C     TO OBTAIN NEW  BETA, U, ALFA, V
C
  100 ITN = ITN+1
      CALL ATIMES( V,P,M,N )
      CALL SAXPY( M,(-ALFA),U,1,P,1 )
      CALL NORMLZ( P,U,M,BETA )
C
      BBNORM = BBNORM + ALFA**2 + BETA**2
C
      CALL ATRANS( U,P,M,N )
      CALL SAXPY( N,(-BETA),V,1,P,1 )
      CALL NORMLZ( P,V,N,ALFA )
C
C
C     COMPUTE NEXT PLANE ROTATION FOR THE QR-FACTORIZATION
C     OF THE LOWER BIDIAGONAL MATRIX  B  (I.E.  Q.B = R)
C
      RHO    = SQRT(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
C
C
C     UPDATE  X  AND THE STANDARD ERROR ESTIMATES
C
      T1 =     PHI/RHO
      T2 = -THETA/RHO
      T3 =     ONE/RHO
      CALL SAXPY( N,T1,W,1,X,1 )
C
      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
C
```

```
C
C      ESTIMATE  NORM(X)  USING AN LQ-FACTORIZATION
C      OF THE UPPER BIDIAGONAL MATRIX  R  (I.E.  R.Q = L)
C
       DELTA  =  SN2*RHO
       GBAR   = -CS2*RHO
       RHS    = PHI - DELTA*Z
       ZBAR   = RHS/GBAR
       XNORM  = SQRT(XXNORM + ZBAR**2)
       GAMMA  = SQRT(GBAR**2 + THETA**2)
       CS2    = GBAR/GAMMA
       SN2    = THETA/GAMMA
       Z      = RHS/GAMMA
       XXNORM = XXNORM + Z**2
C
C
C      TEST FOR CONVERGENCE
C
       ANORM  = SQRT(BBNORM)
       ACOND  = ANORM*SQRT(DDNORM)
       RTOL   = BTOL + ATOL*ANORM*XNORM/BNORM
       TEST1  = SINES
       TEST2  = ARNORM/(ANORM*RNORM)
       TEST3  = ONE/ACOND
C
C      THE FOLLOWING THREE TESTS ARE INDEPENDENT OF THE TOLERANCES
C      ATOL, BTOL  AND  CONLIM.  THEY ARE INTENDED TO GUARD AGAINST
C      ACCIDENTAL SETTING OF THOSE PARAMETERS TO EXTREME VALUES.
C      THEY ARE EQUIVALENT TO THE NORMAL TESTS USING THE VALUES
C      ATOL = EPS,  BTOL = EPS,  CONLIM = 1/EPS.
C
       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
C
C      ALLOW FOR TOLERANCES SET BY USER
C
       IF (ITN   .GE. ITNLIM) ISTOP = 4
       IF (TEST3 .LE. CTOL  ) ISTOP = 3
       IF (TEST2 .LE. ATOL  ) ISTOP = 2
       IF (TEST1 .LE. RTOL  ) ISTOP = 1
```

```
C     ============================================================
C
C     SEE IF IT IS TIME TO PRINT SOMETHING
C
      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
C
C     PRINT A LINE FOR THIS ITERATION
C
  400 CONTINUE
      WRITE(NOUT, 1400) ITN,X(1),RNORM,ARNORM,TEST1,TEST2,ANORM,ACOND
      IF (MOD(ITN,10) .EQ. 0) WRITE(NOUT, 1500)
C     ==============================================================
C
C
C     STOP IF POSSIBLE.
C     THE CONVERGENCE CRITERIA ARE REQUIRED TO BE MET ON  NCONV
C     CONSECUTIVE ITERATIONS, WHERE  NCONV  IS SET BELOW.
C     SUGGESTED VALUE --    NCONV = 1, 2  OR  3
C
  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
C     --------------------------------------------------------------
C     END OF ITERATION LOOP
C     --------------------------------------------------------------
C
```

```
C
C     PRINT SUMMARY OF FINAL SOLUTION STATUS
C
  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
C     COMPUTE FINAL RESIDUAL  R = B - A.X,  AND  A(TRANSPOSE).R
C
  800 T1 = RNORM
      T2 = ARNORM
      T3 = XNORM
      CALL ATIMES( X,P,M,N )
C
      DO 900 I = 1, M
         U(I) = B(I) - P(I)
  900 CONTINUE
C
      CALL NORMLZ( U,P,M,RNORM )
      CALL ATRANS( U,P,M,N )
      CALL NORMLZ( P,W,N,ARNORM )
      CALL NORMLZ( X,W,N,XNORM )
      IF (NOUT .GT. 0) WRITE(NOUT, 3000) BNORM,ANORM,ACOND,T1,T2,T3,
     1   RNORM,ARNORM,XNORM
C
C     FINISH OFF THE STANDARD ERROR ESTIMATES
C
      T = ONE
      IF (M .GT. N) T = M-N
      T = RNORM/SQRT(T)
C
      DO 950 I = 1, N
         SE(I) = T*SQRT(SE(I))
  950 CONTINUE
C
      RETURN
```

```
C     ----------------------------------------------------------------------
C
 1ØØØ 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, I6, 5H   BY, I6
     4   // 25X,  8HATOL    =, 1PE1Ø.2, 1ØX, 8HBTOL    =, 1PE1Ø.2
     5    / 25X,  8HCONLIM =, 1PE1Ø.2, 1ØX, 8HITNLIM =, I1Ø)
 12ØØ FORMAT(// 3X, 3HITN, 9X, 4HX(1), 14X, 7HNORM(R), 8X, 9HNORM(A.R),
     1   3X, 23HCOMPATIBLE INCOMPATIBLE, 4X, 7HNORM(A), 4X, 7HCOND(A) /)
 14ØØ FORMAT(I6, 1PE2Ø.1Ø, 1PE19.1Ø, 1PE12.3, 1P2E13.3, 1P2E11.2)
 15ØØ FORMAT(1X)
 19ØØ FORMAT(
     1    / 19H NO. OF ITERATIONS , I1Ø
     2   // 19H STOPPING CONDITION, I1Ø)
 2ØØØ FORMAT(1H+, 34X, 44H(THE EXACT SOLUTION IS  X = Ø)                    )
 21ØØ FORMAT(1H+, 34X, 44H(NORM(R) IS SMALL ENOUGH, GIVEN  ATOL, BTOL)      )
 22ØØ FORMAT(1H+, 34X, 44H(NORM(A.R) IS SMALL ENOUGH, GIVEN  ATOL)          )
 23ØØ FORMAT(1H+, 34X, 44H(COND(A) HAS EXCEEDED  CONLIM)                    )
 24ØØ FORMAT(1H+, 34X, 44H(ITERATION LIMIT REACHED)                         )
 25ØØ FORMAT(1H+, 34X, 44H(NORM(R) IS SMALL ENOUGH FOR THIS MACHINE)        )
 26ØØ FORMAT(1H+, 34X, 44H(NORM(A.R) IS SMALL ENOUGH FOR THIS MACHINE)      )
 27ØØ FORMAT(1H+, 34X, 44H(COND(A) IS TOO LARGE FOR THIS MACHINE)           )
 3ØØØ FORMAT(
     1      / 19H NORM(B)         TRUE, 1PE2Ø.1Ø,
     2    5X, 19H NORM(A)    ESTIMATE, 1PE15.5,
     3    5X, 19H COND(A)    ESTIMATE, 1PE15.5
     4     // 19H NORM(R)    ESTIMATE, 1PE2Ø.1Ø,
     5    5X, 19H NORM(A.R)  ESTIMATE, 1PE15.5,
     6    5X, 19H NORM(X)    ESTIMATE, 1PE15.7
     7      / 19H                 TRUE, 1PE2Ø.1Ø,
     8    5X, 19H                 TRUE, 1PE15.5,
     9    5X, 19H                 TRUE, 1PE15.7)
C     ----------------------------------------------------------------------
C     END OF LSQR
      END
      SUBROUTINE NORMLZ( P,V,N,BETA )
      INTEGER    N
      REAL       P(N),V(N),BETA
C
C     NORMLZ  IS REQUIRED BY SUBROUTINE LSQR.
C     IT NORMALIZES THE VECTOR  P  AND RETURNS THE RESULT IN  V.
C     ON RETURN,  BETA = NORM(P)   AND   BETA*V = P.
C
C     FUNCTIONS
C
C     BLAS       SNRM2
C
      INTEGER    I
      REAL       ONE,SNRM2,T,ZERO
C
C
      ZERO = 0
      ONE  = 1
      BETA = SNRM2( N,P,1 )
      IF (BETA .LE. ZERO) RETURN
      T = ONE/BETA
C
      DO 20 I = 1, N
         V(I) = P(I)*T
   20 CONTINUE
      RETURN
C
C     END OF NORMLZ
      END
```

```
      SUBROUTINE SAXPY( N,A,X,INCX,Y,INCY )
      REAL       A,X(N),Y(N)
C
C     THIS MAY BE REPLACED BY THE CORRESPONDING  BLAS  ROUTINE.
C     THE FOLLOWING IS A SIMPLE VERSION FOR USE WITH  LSQR.
C
      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)
C
C     THIS MAY BE REPLACED BY THE CORRESPONDING  BLAS  ROUTINE.
C     THE FOLLOWING IS A SIMPLE VERSION FOR USE WITH  LSQR.
C
      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
C     COMPUTE  P = A.V  FOR TEST MATRIX  A
C
      INTEGER     I,N1
      REAL        ZERO
      REAL        D,R,XTRUE,Y,Z
      COMMON     /LSCOMM/ D(1ØØ),R(1ØØ),XTRUE(1ØØ),Y(1ØØ),Z(1ØØ)
C
C
      CALL HPROD( V,P,Z,N )
      DO 1ØØ I=1,N
         P(I) = D(I)*P(I)
  1ØØ CONTINUE
C
      IF (M .LE. N) GO TO 5ØØ
      ZERO = Ø
      N1 = N+1
      DO 2ØØ I=N1,M
         P(I) = ZERO
  2ØØ CONTINUE
C
  5ØØ CALL HPROD( P,P,Y,M )
      RETURN
C
C     END OF ATIMES
      END
```

```
      SUBROUTINE ATRANS( U,P,M,N )
      INTEGER     M,N
      REAL        U(M),P(N)
C
C     COMPUTE  P = A(TRANSPOSE).U  FOR TEST MATRIX  A
C
      INTEGER     I
      REAL        D,R,XTRUE,Y,Z
      COMMON     /LSCOMM/ D(1ØØ),R(1ØØ),XTRUE(1ØØ),Y(1ØØ),Z(1ØØ)
C
C
      CALL HPROD( U,P,Y,M )
C
      DO 1ØØ I=1,N
         P(I) = D(I)*P(I)
  1ØØ CONTINUE
C
      CALL HPROD( P,P,Z,N )
      RETURN
C
C     END OF ATRANS
      END
      SUBROUTINE HPROD( V,P,Z,N )
      INTEGER    N
      REAL       P(N),V(N),Z(N)
C
C     APPLY HOUSEHOLDER TRANSFORMATION TO GET  P = (I - 2 Z Z').V
C
      INTEGER    I
      REAL       S
C
      S = Ø
      DO 1ØØ I=1,N
         S = Z(I)*V(I) + S
  1ØØ CONTINUE
C
      S = S + S
      DO 2ØØ I=1,N
         P(I) = V(I) - S*Z(I)
  2ØØ CONTINUE
C
      RETURN
C
C     END OF HPROD
      END



      SUBROUTINE LSTP( M,N,NDUPLC,NPOWER,B,ACOND,RNORM )
      INTEGER    M,N,NDUPLC,NPOWER
      REAL       B(M),ACOND,RNORM
C
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.
C
C     FUNCTIONS AND SUBROUTINES
C
C     TESTPROB   ATIMES,HPROD
C     LSQR       NORMLZ
C     FORTRAN    COS,FLOAT,SIN
C
      INTEGER    I,J,MN
      REAL       ALFA,BETA,COS,FLOAT,FOURPI,SIN,T
      REAL       D,R,XTRUE,Y,Z
      COMMON    /LSCOMM/ D(1ØØ),R(1ØØ),XTRUE(1ØØ),Y(1ØØ),Z(1ØØ)
C
C     --------------------
C     MAKE TWO VECTORS OF NORM 1.Ø, FOR HOUSEHOLDER TRANSFORMATIONS
C     --------------------
      FOURPI = 4.Ø*3.141592
      ALFA   = FOURPI/FLOAT(M)
      BETA   = FOURPI/FLOAT(N)
C
      DO 1ØØ I=1,M
         Y(I) = SIN(FLOAT(I)*ALFA)
  1ØØ CONTINUE
C
      DO 2ØØ I=1,N
         Z(I) = COS(FLOAT(I)*BETA)
  2ØØ CONTINUE
```

```
C
      CALL NORMLZ( Y,Y,M,ALFA )
      CALL NORMLZ( Z,Z,N,BETA )
C
C     --------------------
C     SET SINGULAR VALUES FOR DIAGONAL MATRIX  D
C     --------------------
      DO 300 I=1,N
         J = (I-1+NDUPLC)/NDUPLC
         T = J*NDUPLC
         T = T/FLOAT(N)
         D(I) = T**NPOWER
  300 CONTINUE
C
      ACOND = D(N)/D(1)
C
C     --------------------
C     SET SOLUTION  XTRUE
C     --------------------
      DO 400 I=1,N
         XTRUE(I) = N-I
  400 CONTINUE
C
C     --------------------
C     COMPUTE RHS  B
C     --------------------
      CALL ATIMES( XTRUE,B,M,N )
      IF (M .LE. N) RETURN
C
C     --------------------
C     FOR LEAST SQUARES, ADD RESIDUAL  R
C     --------------------
      DO 500 I=1,N
         R(I) = 0.0
  500 CONTINUE
C
      T  = 1
      MN = M-N
      DO 600 I=1,MN
         J    = N+I
         R(J) = T*FLOAT(I)/FLOAT(M)
         T    = -T
  600 CONTINUE
C
      CALL HPROD( R,R,Y,M )
C
      DO 700 I=1,M
         B(I) = B(I) + R(I)
  700 CONTINUE
C
      CALL NORMLZ( R,P,M,RNORM )
      RETURN
C
C     END OF LSTP
      END
      SUBROUTINE TEST( M,N,NDUPLC,NPOWER )
      INTEGER    M,N,NDUPLC,NPOWER
C
C     EXAMPLE DRIVER ROUTINE FOR TESTING  LSQR
C
      INTEGER    ISTOP,ITNLIM,J,NOUT
      REAL       B(100),P(100),U(100),V(100),
     1           W(100),X(100),SE(100),
     2           ATOL,BTOL,CONLIM,RNORM,ANORM,ACOND
C
C
C     GENERATE SPECIFIED TEST PROBLEM
C
      CALL LSTP( M,N,NDUPLC,NPOWER,B,ACOND,RNORM )
C
      NOUT = 6
      WRITE(NOUT, 1000) NDUPLC,NPOWER,ACOND,RNORM
C
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
C
      CALL LSQR( M,N,M,B,P,U,V,W,X,SE,
     1   ATOL,BTOL,CONLIM,ITNLIM,NOUT,ISTOP,RNORM,ANORM,ACOND )
C
C     OUTPUT RESULTS
C
      WRITE(NOUT, 2000)
      WRITE(NOUT, 4000)  (J,X(J), J=1,N)
      WRITE(NOUT, 3000)
      WRITE(NOUT, 4000)  (J,SE(J),J=1,N)
      RETURN
C
 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)  =, 1PE20.10)
 2000 FORMAT(/// 9H SOLUTION)
 3000 FORMAT(/ 16H STANDARD ERRORS)
 4000 FORMAT(5(I7, 1PE17.9))
C     END OF TEST
      END



C
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 $Ax = b$,

$P(80,40,4,2)$ - 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 $s_1 s_2 \ldots s_k$ 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 $\mathrm{cond}(A) = \|A\|_F \|A^+\|_F$.

```
LEAST-SQUARES TEST PROBLEM.

SINGULAR VALUES REPEATED  1  TIMES.          POWER FACTOR =  6

COND(A)   =   1.0000E+06          NORM(R)   =     0.


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

                          THE MATRIX  A  HAS DIMENSIONS    10   BY    10

                          ATOL   =  1.00E-10          BTOL   =  1.00E-10
                          CONLIM =  1.00E+10          ITNLIM =       100


  ITN        X(1)                 NORM(R)         NORM(A.R)   COMPATIBLE INCOMPATIBLE    NORM(A)     COND(A)

    0   0.                   2.1988640593E+00    2.052E+00

    1   -2.7042692029E-01   6.8442884975E-01    3.332E-01     3.113E-01     4.957E-01   9.82E-01    1.00E+00
    2   -2.9210120797E-01   2.9393864349E-01    2.777E-02     1.337E-01     8.351E-02   1.13E+00    2.42E+00
    3   -2.0897591689E-01   2.5614275400E-01    2.176E-02     1.165E-01     7.396E-02   1.15E+00    6.47E+00
    4   -1.6482707892E-01   2.0079135630E-01    9.416E-03     9.132E-02     4.020E-02   1.17E+00    1.27E+01
    5   -4.3450538422E-01   9.8536462677E-02    1.512E-03     4.481E-02     1.312E-02   1.17E+00    2.79E+01
    6   -1.1935063864E+00   2.6172194262E-02    6.629E-03     1.190E-02     2.166E-01   1.17E+00    7.97E+01
    7   -1.1964009429E+00   2.5314270652E-02    1.032E-04     1.151E-02     2.649E-03   1.54E+00    1.05E+02
    8   -1.1977951358E+00   2.5223397071E-02    1.130E-03     1.147E-02     2.910E-02   1.54E+00    1.10E+02
    9   -1.4087142112E+00   3.8117040781E-03    4.249E-06     1.733E-03     6.849E-04   1.63E+00    4.13E+02
   10   -1.4086990982E+00   3.8116443356E-03    3.639E-06     1.733E-03     5.822E-04   1.64E+00    4.16E+02

   11   -1.4063681329E+00   3.8034045946E-03    2.498E-04     1.730E-03     3.983E-02   1.65E+00    4.44E+02
   12   -9.8166255060E-01   1.6967019812E-03    1.746E-04     7.716E-04     5.335E-02   1.93E+00    2.43E+03
   13   -8.8371064353E-01   4.0540073308E-04    4.278E-06     1.844E-04     5.462E-03   1.93E+00    2.70E+03
   14   -8.8370769258E-01   4.0532069151E-04    1.686E-07     1.843E-04     2.076E-04   2.00E+00    2.80E+03
   15   -8.8370300344E-01   4.0530422373E-04    2.875E-08     1.843E-04     3.539E-05   2.00E+00    2.80E+03
   16   -8.8370299585E-01   4.0530421761E-04    6.436E-08     1.843E-04     7.763E-05   2.05E+00    2.85E+03
   17   -8.7819876218E-01   4.0354421149E-04    9.828E-06     1.835E-04     1.087E-02   2.24E+00    4.52E+03
   18   -4.4830117734E-01   2.2665093442E-04    2.943E-06     1.031E-04     5.759E-03   2.26E+00    2.94E+04
   19   -4.3095339796E-01   2.1661424492E-04    3.377E-05     9.851E-05     6.911E-02   2.26E+00    3.00E+04
   20   -2.4876416969E-01   1.3726150976E-05    3.958E-06     6.242E-06     1.245E-01   2.32E+00    3.63E+04

   21   -2.4869779837E-01   1.3097398302E-05    1.084E-06     5.956E-06     3.299E-02   2.51E+00    3.94E+04
   22   -2.4838195872E-01   9.6161025747E-06    3.765E-10     4.373E-06     1.550E-05   2.53E+00    3.96E+04
   23   -2.4838133275E-01   9.6159159701E-06    1.188E-08     4.373E-06     4.892E-04   2.53E+00    3.96E+04
   24   -2.4838053635E-01   9.6156667487E-06    3.149E-09     4.373E-06     1.289E-04   2.54E+00    3.98E+04
   25   -2.4838053393E-01   9.6156661046E-06    1.280E-09     4.373E-06     4.944E-05   2.69E+00    4.23E+04
   26   -2.4838053325E-01   9.6156658399E-06    3.565E-10     4.373E-06     1.335E-05   2.78E+00    4.36E+04
   27   -2.4838051374E-01   9.6156630369E-06    9.624E-12     4.373E-06     3.599E-07   2.78E+00    4.36E+04
   28   -2.4833962115E-01   9.6156417403E-06    2.389E-09     4.373E-06     8.933E-05   2.78E+00    4.40E+04
   29   -2.4826395733E-01   9.6156023209E-06    2.741E-08     4.373E-06     1.023E-03   2.79E+00    4.48E+04
   30   -1.9569041273E-01   9.5881720382E-06    4.905E-09     4.361E-06     1.730E-04   2.96E+00    2.28E+05
```

```
 31   -1.9568086220E-01   9.5881670474E-06   1.532E-09    4.361E-06    5.326E-05   3.00E+00   2.32E+05
 32   -1.9567656334E-01   9.5881647998E-06   9.431E-10    4.361E-06    3.264E-05   3.01E+00   2.33E+05
 33   -1.9521285898E-01   9.5879224696E-06   7.158E-10    4.360E-06    2.475E-05   3.02E+00   2.34E+05
 34   -1.9520969009E-01   9.5879208170E-06   5.584E-09    4.360E-06    1.929E-04   3.02E+00   2.34E+05
 35    8.0752496087E+00   3.0405792019E-06   1.351E-07    1.383E-06    1.398E-02   3.18E+00   3.02E+06
 36    8.4995138483E+00   2.2368657654E-06   8.016E-07    1.017E-06    1.127E-01   3.18E+00   3.09E+06
 37    8.9992703240E+00   8.5373339872E-08   1.008E-08    3.883E-08    3.662E-02   3.22E+00   3.22E+06
 38    8.9999073652E+00   3.0337501764E-08   2.525E-08    1.380E-08    2.580E-01   3.23E+00   3.23E+06
 39    8.9999796830E+00   1.4052497214E-08   3.516E-09    6.391E-09    7.417E-02   3.37E+00   3.37E+06
 40    8.9999979862E+00   3.8156898804E-09   1.523E-11    1.735E-09    1.178E-03   3.39E+00   3.39E+06

NO. OF ITERATIONS          40

STOPPING CONDITION          1       (NORM(R) IS SMALL ENOUGH, GIVEN  ATOL, BTOL)

NORM(B)         TRUE    2.1988640593E+00       NORM(A)   ESTIMATE     3.38644E+00       COND(A)   ESTIMATE     3.38686E+06

NORM(R)    ESTIMATE    3.8156898804E-09       NORM(A.R) ESTIMATE     1.52253E-11       NORM(X)   ESTIMATE  1.6881941E+01
                TRUE    3.8143764658E-09                      TRUE    1.52701E-11                     TRUE  1.6881946E+01


SOLUTION
     1  8.999997986E+00        2  8.000009492E+00        3  6.999996033E+00        4  6.000000609E+00        5  5.000004904E+00
     6  4.000001516E+00        7  2.999996032E+00        8  1.999996032E+00        9  1.000001516E+00       10  4.904386609E-06

STANDARD ERRORS
     1  3.668685944E-03        2  3.839756591E-04        3  3.817949531E-04        4  1.458249465E-04        5  4.718820577E-04
     6  1.458203521E-04        7  3.817600691E-04        8  3.817611543E-04        9  1.458185086E-04       10  4.718820199E-04
```

```
LEAST-SQUARES TEST PROBLEM.

SINGULAR VALUES REPEATED  4  TIMES.          POWER FACTOR =  2

COND(A)   =   1.0000E+02           NORM(R)    =      1.8599395152E+00

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

                          THE MATRIX  A  HAS DIMENSIONS    80    BY     40

                          ATOL    =  1.00E-10          BTOL    =  1.00E-10
                          CONLIM =  1.00E+05           ITNLIM =       100

   ITN        X(1)              NORM(R)          NORM(A.R)    COMPATIBLE INCOMPATIBLE    NORM(A)     COND(A)

    0    0.                   2.8085842421E+01    1.420E+01

    1    -2.8314582863E-01    1.9828879909E+01    9.783E+00    7.060E-01     6.203E-01    7.14E-01    1.00E+00
    2     9.2511600003E-01    1.4498340606E+01    5.553E+00    5.162E-01     3.609E-01    1.06E+00    2.43E+00
    3    -2.3143976900E+00    1.1093697190E+01    3.458E+00    3.950E-01     2.428E-01    1.28E+00    3.98E+00
    4    -5.4493230062E+00    8.6829373292E+00    2.192E+00    3.092E-01     1.769E-01    1.43E+00    5.71E+00
    5    -6.0275711966E+00    6.8719797239E+00    1.312E+00    2.447E-01     1.257E-01    1.52E+00    7.69E+00
    6    -4.3976106478E+00    5.2922883743E+00    6.604E-01    1.884E-01     7.957E-02    1.57E+00    1.03E+01
    7    -6.6415172885E-02    3.8633906426E+00    2.720E-01    1.376E-01     4.440E-02    1.59E+00    1.47E+01
    8     5.8127060346E+00    2.7225094402E+00    7.856E-02    9.694E-02     1.814E-02    1.59E+00    2.31E+01
    9     1.0608363669E+01    1.9925787310E+00    2.967E-02    7.095E-02     9.354E-03    1.59E+00    4.58E+01
   10     1.0609611660E+01    1.9923440647E+00    7.400E-03    7.094E-02     1.993E-03    1.86E+00    5.36E+01

   11     1.4920859464E+01    1.9727279215E+00    2.075E-01    7.024E-02     5.595E-02    1.88E+00    9.11E+01
   12     3.8760040089E+01    1.8610128863E+00    4.033E-02    6.626E-02     1.059E-02    2.05E+00    2.12E+02
   13     3.8999824899E+01    1.8599397490E+00    4.569E-04    6.622E-02     1.145E-04    2.14E+00    2.23E+02
   14     3.9000002111E+01    1.8599395154E+00    4.246E-06    6.622E-02     1.038E-06    2.20E+00    2.29E+02
   15     3.9000000085E+01    1.8599395153E+00    1.265E-07    6.622E-02     3.052E-08    2.23E+00    2.32E+02
   16     3.9000000018E+01    1.8599395153E+00    2.334E-07    6.622E-02     5.540E-08    2.27E+00    2.36E+02
   17     3.8999999925E+01    1.8599395153E+00    3.266E-08    6.622E-02     7.156E-09    2.45E+00    2.56E+02
   18     3.9000000001E+01    1.8599395153E+00    3.076E-09    6.622E-02     6.721E-10    2.46E+00    2.57E+02
   19     3.9000000001E+01    1.8599395153E+00    4.054E-10    6.622E-02     8.419E-11    2.59E+00    2.70E+02

NO. OF ITERATIONS          19

STOPPING CONDITION          2       (NORM(A.R) IS SMALL ENOUGH, GIVEN  ATOL)

NORM(B)         TRUE    2.8085842421E+01      NORM(A)    ESTIMATE      2.58873E+00      COND(A)    ESTIMATE     2.70207E+02

NORM(R)     ESTIMATE    1.8599395153E+00      NORM(A.R) ESTIMATE      4.05370E-10      NORM(X)    ESTIMATE  1.4331783E+02
                TRUE    1.8599395152E+00                     TRUE      4.95573E-10                    TRUE  1.4331783E+02
```

```
SOLUTION
     1  3.900000000E+01      2  3.800000000E+01      3  3.700000000E+01      4  3.600000000E+01      5  3.499999999E+01
     6  3.399999999E+01      7  3.299999998E+01      8  3.199999998E+01      9  3.100000001E+01     10  3.000000003E+01
    11  2.900000002E+01     12  2.800000003E+01     13  2.700000000E+01     14  2.600000000E+01     15  2.500000000E+01
    16  2.400000000E+01     17  2.300000000E+01     18  2.200000001E+01     19  2.100000001E+01     20  2.000000001E+01
    21  1.900000001E+01     22  1.800000001E+01     23  1.700000000E+01     24  1.600000000E+01     25  1.500000000E+01
    26  1.400000000E+01     27  1.300000000E+01     28  1.199999999E+01     29  1.099999999E+01     30  9.999999993E+00
    31  8.999999993E+00     32  7.999999994E+00     33  6.999999996E+00     34  5.999999998E+00     35  5.000000000E+00
    36  4.000000002E+00     37  3.000000005E+00     38  2.000000006E+00     39  1.000000007E+00     40  7.677679018E-09

STANDARD ERRORS
     1  1.174057969E+01      2  1.180840472E+01      3  1.212782669E+01      4  1.263417887E+01      5  3.935482636E+00
     6  3.777500596E+00      7  3.997619461E+00      8  4.377073447E+00      9  3.990530866E+00     10  4.155077390E+00
    11  3.958476812E+00     12  3.423109222E+00     13  2.847820222E+00     14  1.792520653E+00     15  1.186578306E+00
    16  1.636335294E+00     17  2.440620671E+00     18  3.462314579E+00     19  3.887230926E+00     20  4.075132291E+00
    21  3.880713999E+00     22  3.348173999E+00     23  2.416713985E+00     24  1.410733701E+00     25  4.557552121E-01
    26  1.351041163E+00     27  2.411464001E+00     28  3.301372852E+00     29  3.872808079E+00     30  4.070009380E+00
    31  3.870527642E+00     32  3.292844790E+00     33  2.401036162E+00     34  1.287646047E+00     35  4.649576757E-01
    36  1.272382778E+00     37  2.414297039E+00     38  3.292920575E+00     39  3.868551215E+00     40  4.070802635E+00
```

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER SOL 78-19	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A BIDIAGONALIZATION ALGORITHM FOR SPARSE LINEAR EQUATIONS AND LEAST-SQUARES PROBLEMS		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER SOL 78-19
7. AUTHOR(s) Christopher C. Paige and Michael A. Saunders		8. CONTRACT OR GRANT NUMBER(s) N00014-75-C-0267
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Operations Research -- SOL Stanford University Stanford, CA 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR-047-143
11. CONTROLLING OFFICE NAME AND ADDRESS Operations Research Program -- ONR Department of the Navy, 800 N. Quincy Street Arlington, VA 22217		12. REPORT DATE October 1978
		13. NUMBER OF PAGES 89
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

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

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

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

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

SEE ATTACHED

OVER

A BIDIAGONALIZATION ALGORITHM FOR SPARSE LINEAR EQUATIONS AND LEAST-SQUARES PROBLEMS
SOL 78-19
Christopher C. Paige and Michael A. Saunders

min value of (Ax-b) sub 2

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.

END
3-79