SOLUTION OF SPARSE INDEFINITE SYSTEMS OF EQUATIONS AND LEAST SQUARES PROBLEMS

by

C. C. Paige and M. A. Saunders

STAN-CS-73-399
November 1973

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

SOLUTION OF SPARSE INDEFINITE SYSTEMS OF EQUATIONS AND LEAST SQUARES PROBLEMS

C. C. Paige* and M. A. Saunders**

Abstract

The method of conjugate gradients for solving systems of linear equations with a symmetric positive definite matrix A is given as a logical development of the Lanczos algorithm for tridiagonalizing A. This approach suggests numerical algorithms for solving such systems when A is symmetric but indefinite. The new methods can be applied to linear least squares problems with or without constraints, with simplifications when there are no constraints. These methods have advantages when A is large and sparse. Fortran subroutines are included.

* School of Computer Science, McGill University, Montreal, Canada. Supported in part by National Research Council of Canada grant A8652, and National Science Foundation grant GJ 29988X.

** Applied Mathematics Division, D.S.I.R., Wellington, New Zealand. Supported in part by AEC grant AT(04-3)-326 PA # 30.

1. Introduction

Here some methods are considered for solving

$$(1.1) \qquad Ax = b$$

when the $n \times n$ real symmetric matrix A is large and sparse. The special case that arises when a linear least squares problem is transformed to a larger problem of the form (1.1) is also examined. Unlike matrix factorization, the methods given here for solving (1.1) regard A as an operator and only require matrix-vector products, building up x as a combination of vectors derived from a Krylov sequence. Some basic theory for different methods of this type is given in Section 2.

An example of this type is the method of conjugate gradients [2], which is often useful for solving such problems when A is positive definite [11]. Although rounding errors cause the conjugate gradients method to depart significantly from its ideal path, it can still be very effective when regarded as an iterative method, and the solution can often be found to the required accuracy in far less than n steps.

The method developed by Lanczos [4] for tridiagonalizing A is directly related to the conjugate gradients method, as is explained in [5] and [3], and the rounding error properties of both methods are closely related. A description of the Lanczos process is given in Section 3, and the method of conjugate gradients is developed from it in Section 4. This gives computational insights into the method, and leads to two new algorithms that may be used when A has both positive and negative eigenvalues; these are described in Section 5 and 6. The method in Section 6 can also be used if A is singular and (1.1) is not a consistent set of equations, and some properties of this method are developed in Section 7.

When linear least squares problems are put in the form (1.1), as in equation (8.2), the symmetric matrix A will be indefinite with some zero sub-blocks. This is true for unconstrained problems [12], and also for problems with linear equality constraints [1]. If these problems are large and sparse then the new methods given here can be used. When there are no constraints the algorithms can be simplified to take advantage of the special form of A , saving storage and computation. The algorithm in Section 5 is extended to take this into account in Section 8, where the resulting algorithm is shown to be closely related to that given in [10, Section 4].

Computational results for the new algorithms are discussed in Section 9, indicating that they give accurate results; but while the methods can often take much less than n steps, there are cases where they take a great deal more. A rounding error analysis of these algorithms will be given in a later report.

Fortran subroutines for the new method in Section 5 and its extension to the least squares problem in Section 8 are given in the Appendix.

The methods given here for symmetric indefinite systems would appear to be superior to those suggested by Luenberger [7], [8], as the latter present some difficult unsettled questions when routine practical application is considered. These particular problems do not arise here, since the development of the algorithms from the Lanczos process allows a good understanding of their numerical properties, and so some possible numerical instabilities have been avoided.

In the text upper case Roman letters denote matrices, lower case Roman denote vectors, and lower case Greek denote scalars. The exceptions are c and s used to denote cosine and sine. The symbol $\|\cdot\|$ denotes the 2-norm of a vector or matrix.

2. General Theory

Given the set of equations

(2.1) $$r + Ax = b \quad , \quad Ar = 0$$

where A is a real $n \times n$ symmetric matrix which may be both indefinite and singular, we will consider computing various approximations to x of the form $V_k y$, $V_k \equiv [v_1, v_2, \ldots, v_k]$, where the v_i are a given set of linearly independent vectors. In particular we will look for solutions $x_k \equiv V_k y_k$ which give stationary values to

(2.2) $$f_k(y) \equiv (AV_k y - b)^T B(AV_k y - b)$$

where B is some symmetric matrix; thus $f_k(y)$ will be a norm of the residual if B is positive definite. Note that this is just a theoretical tool that will lead to different methods, and that B will not be required explicitly.

The function $f_k(y)$ has a stationary value at y_k if

(2.3) $$V_k^T ABA V_k y_k = V_k^T A B b \quad ,$$

that is if

(2.4) $$V_k^T A B r_k = 0 \quad , \quad r_k \equiv b - Ax_k \quad ,$$

and the methods to be considered will essentially try to solve (2.3). Since the second derivative of $f_k(y)$ is $2V_k^T ABAV_k$, it follows that if ABA is positive definite there is a unique y that minimizes $f_k(y)$. If ABA is only positive semidefinite then the minimizing y is not necessarily unique, while if ABA is indefinite we only have a stationary point of $f_k(y)$. In any case x_k is an approximation to the solution in the sense that the residual is restricted to the null space of $V_k^T AB$, and V_k can be chosen to reduce the dimension of this null space with increasing k .

An obvious choice for B is A^m for some integer m , and we will restrict ourselves to this case. Choosing $m = -2$ would essentially require a knowledge of x on the right hand side of (2.3), and for $m \leq -2$ solving (2.3) would appear to require at least as much knowledge as solving the original problem. The choices $m = -1, 0$ appear to be the most useful, and will now be considered.

Case (a). Taking $m = -1$ would give $B = A^{-1}$, but to allow for the more general case of singular A , we take $B = A^-$, where A^- is a generalized inverse of A such that AA^- is the orthogonal projector onto $\mathcal{R}(A)$, the range of A . With this choice we have from (2.1)

$$AA^-b = AA^-r + AA^-Ax = Ax$$

and (2.3) becomes

$$(2.5) \qquad V_k^T A V_k y_k = V_k^T A x = V_k^T b - V_k^T r \quad ,$$

which cannot be solved directly for y_k unless a value for $V_k^T r$ is known. We will only consider the case $V_k^T r = 0$, so the method will be applicable if $r = 0$ or $v_i \in \mathcal{R}(A)$, $i = 1,\ldots,k$, and then

$$(2.6) \qquad V_k^T A V_k y_k = V_k^T b \quad , \quad x_k = V_k y_k$$

gives a stationary value of (2.2) with $B = A^-$. If the columns of V_k span $\mathcal{R}(A)$ then we have the least squares solution of minimum length.

Case (b). Taking $m = 0$ gives $B = I$ and we can minimize $\|r_k\|$ by solving

$$(2.7) \qquad V_k^T A^2 V_k u_k = V_k^T A b \quad , \quad x_k = V_k u_k \quad ,$$

where y has been replaced by u to avoid confusion with (2.6). Furthermore, if $v_1,\ldots,v_k$ span $R(A)$ then x_k is the minimum length least squares solution of (2.1). We will call methods based on (2.7) minimum residual methods. A possible danger with these is that if A is poorly conditioned for solutions of equations, then the condition of the problem (2.7) can be much worse. Values of $m > 0$ would lead to more poorly conditioned problems still, and will not be examined here.

3. The Lanczos Vectors

If the vectors $v_1,\dots,v_k$ in Section 2 are computed by the Lanczos algorithm [4], then some important and computationally useful simplifications result. In particular, algorithms arise which are useful for large sparse matrices: for example the method of conjugate gradients.

The initial vector we will use in the Lanczos algorithm will be

$$(3.1) \qquad v_1 = b/\beta_1 \quad , \quad \beta_1 \equiv \|b\| \quad ;$$

there are indications that this choice, and possibly $v_1 = Ab/\|Ab\|$, are the most computationally viable ones for solving large problems of the form (2.1), and there is also some theoretical justification for (3.1) but this has not been shown rigorously. If we have an initial approximation x_0 to x in (2.1) then we change the problem to

$$r + Ag = r_0 \equiv b - Ax_0 \quad , \quad Ar = 0 \quad , \quad x = x_0 + g \quad ,$$

and proceed as before. With the choice of v_1 in (3.1) we restrict ourselves to the case of $r = 0$ in Section 2, case (a), though there is no such restriction on case (b).

A satisfactory computational variant of the Lanczos algorithm, [9], has as its j-th step, defining $v_0 \equiv 0$,

$$(3.2) \qquad \beta_{j+1}v_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1} \quad , \quad \alpha_j = v_j^T A v_j$$

with $\beta_{j+1} \geq 0$ chosen so that $\|v_{j+1}\| = 1$. After the k-th step

$$AV_k = V_kT_k + \beta_{k+1}v_{k+1}e_k^T \quad , \quad T_k \equiv \begin{bmatrix} \alpha_1 & \beta_2 & & \\ \beta_2 & \alpha_2 & \beta_3 & \\ & \cdot & \cdot & \cdot \\ & & \beta_k & \alpha_k \end{bmatrix} ,$$

(3.3)

$$V_k^TV_k = I \equiv [e_1,\ldots,e_k] \quad , \quad V_k^Tv_{k+1} = 0 \quad .$$

The process will be terminated at the first zero β_j , so from now on we can assume that $\beta_j \neq 0$, $j = 1,\ldots,k$.

From (3.3) and (3.1) we have $V_k^TAV_k = T_k$ and $V_k^Tb = \beta_1e_1$, and with this choice of vectors in Section 2(a) equation (2.6) becomes

(3.4) $\quad T_ky_k = \beta_1e_1 \quad , \quad x_k^c = V_ky_k \quad ,$

where the superscript c indicates that it is the solution that would be obtained using the method of conjugate gradients, as will now be explained.

4. Derivation of the Conjugate Gradients Method from the Lanczos Process

The conjugate gradients method [2] can be developed in a straightforward manner from the Lanczos process. The purpose of giving this development here is to divide the conjugate gradients method into separate computational algorithms whose numerical properties are more clearly understood; this leads to some new and useful methods.

If A is positive definite then so is $T_k = V_k^T A V_k$ in (3.4), and hence the Cholesky factorization

$$(4.1) \qquad T_k = \mathcal{L}_k \mathcal{D}_k \mathcal{L}_k^T$$

exists. Here $\mathcal{D}_k$ is diagonal with positive elements and $\mathcal{L}_k$ is unit lower bidiagonal, and these can be developed as k increases. Unfortunately y_k in (3.4) changes fully with each increase in k, and so $V_k y_k$ cannot be accumulated as k increases. This difficulty can be overcome if we define $p_k \equiv \mathcal{L}_k^T y_k$ and $C_k \equiv V_k \mathcal{L}_k^{-T}$ so that (3.4) becomes

$$(4.2) \qquad \mathcal{L}_k \mathcal{D}_k p_k = \beta_1 e_1 \quad , \quad x_k^c = C_k p_k \quad .$$

The columns of C_k can be found in ascending order by solving

$$(4.3) \qquad \mathcal{L}_k C_k^T = V_k^T$$

for the rows of C_k^T, and since p_k can be developed similarly from (4.2), it follows that $x_k^c = C_k p_k$ can be accumulated as the algorithm progresses and the columns of V_k and C_k need not be kept once they have been used in the Lanczos algorithm and in forming x_k^c. The columns of C_k are A-conjugate, since

$$(4.4) \qquad C_k^T A C_k = \mathcal{L}_k^{-1} T_k \mathcal{L}_k^{-T} = \mathcal{D}_k \quad ,$$

and a comparison with [3] shows that this method is mathematically equivalent to the method of conjugate gradients. The approach here is computationally a little different, for example involving unnecessary normalization of the Lanczos vectors, but the advantages are that it emphasizes how the method is based on the Lanczos algorithm, with the eigenvalues, and therefore the spectral condition number, of T_k approaching those of A as k increases. Furthermore the role of the Cholesky decomposition becomes apparent, with the subsequent need for A to be positive definite to ensure numerical stability.

If A is an indefinite symmetric matrix, then the factorization (4.1) can still be tried, often with success, but it does not always exist and can no longer be relied upon numerically.

5. An Algorithm for Indefinite Symmetric Systems

The possible failure of the method of conjugate gradients in problems involving indefinite symmetric matrices leaves a need for a numerically stable method based on the Lanczos vectors. Several such methods are possible, using various stable factorizations of T_k in (3.4), but the method we found to be most theoretically and numerically satisfying is that based on the orthogonal factorization

$$(5.1) \qquad T_k = \bar{L}_k Q_k \quad , \quad Q_k^T Q_k = I \quad ,$$

with $\bar{L}_k$ lower triangular. The bar is used to indicate that $\bar{L}_k$ differs from the $k \times k$ leading part of $\bar{L}_{k+1}$ in the (k,k) element only. As before y_k in (3.4) need not be computed; instead, if we define

$$(5.2) \qquad \bar{W}_k \equiv [w_1, \ldots, w_{k-1}, \bar{w}_k] \equiv V_k Q_k^T$$

$$(5.3) \qquad \bar{z}_k \equiv (\zeta_1, \ldots, \zeta_{k-1}, \bar{\zeta}_k)^T \equiv Q_k y_k$$

then (3.4) becomes

$$(5.4) \qquad \bar{L}_k \bar{z}_k = \beta_1 e_1 \quad , \quad x_k^c = \bar{W}_k \bar{z}_k \quad ,$$

and it turns out once more that the v_i and w_i can be formed, used, and discarded one by one. This gives mathematically the same solution as does conjugate gradients, but here the factorization is numerically stable even when T_k is indefinite.

The factorization (5.1) is best obtained by a series of orthonormal matrices $Q_{i,i+1}$ each of which differs from the unit matrix only in the elements $q_{ii} = -q_{i+1,i+1} = c_i = \cos \theta_i$, $q_{i,i+1} = q_{i+1,i} = s_i = \sin \theta_i$. Thus

$$(5.5)\qquad T_kQ_{1,2}\cdots Q_{k-1,k} \equiv T_kQ_k^T = \bar{L}_k \equiv \begin{bmatrix} \gamma_1 & & & & \\ \delta_2 & \gamma_2 & & & \\ \epsilon_3 & \delta_3 & \gamma_3 & & \\ \cdot & \cdot & \cdot & & \\ & & \epsilon_k & \delta_k & \bar{\gamma}_k \end{bmatrix} ,$$

where in the next step we compute

$$(5.6)\qquad \gamma_k = \left(\bar{\gamma}_k^2 + \beta_{k+1}^2\right)^{1/2} , \quad c_k = \bar{\gamma}_k/\gamma_k , \quad s_k = \beta_{k+1}/\gamma_k \quad .$$

In the following discussion we will use L_k to denote $\bar{L}_k$ with $\bar{\gamma}_k$ replaced by γ_k . Similarly, following (5.2) and (5.3), we define $z_k \equiv (\zeta_1,\ldots,\zeta_k)^T$ and $W_k \equiv [w_1,\ldots,w_k]$, where z_k is found from

$$(5.7)\qquad L_kz_k = \beta_1e_1 \quad ,$$

so from (5.4) and (5.6)

$$(5.8)\qquad \zeta_k = \bar{\gamma}_k\bar{\zeta}_k/\gamma_k = c_k\bar{\zeta}_k \quad ;$$

finally from (5.2) and the form of $Q_{k,k+1}$ in (5.5) we have

$$(5.9)\qquad [\bar{w}_k,v_{k+1}]\begin{bmatrix} c_k & s_k \\ s_k & -c_k \end{bmatrix} = [w_k,\bar{w}_{k+1}] \quad , \text{ with } \bar{w}_1 \equiv v_1 \, .$$

The algorithm defined by (5.1) to (5.4) should not be implemented directly, since it is wasteful to update x_k^c fully each step in (5.4), while if $\bar{L}_k$ is singular in (5.4) then $\bar{z}_k$ is undefined. Instead we see from (5.5) and (5.6) that L_k is nonsingular if $\beta_{k+1} \neq 0$, so z_k is defined in (5.7), and rather than updating x_k^c each step we update

(5.10) $$x_k^L = W_k z_k = x_{k-1}^L + \zeta_k w_k$$

where L indicates we are using L_k rather than $\bar{L}_k$. Since (5.4) and (5.10) show that

(5.11) $$x_{k+1}^c = x_k^L + \bar{\zeta}_{k+1} \bar{w}_{k+1} \quad ,$$

we are always able to obtain x_{k+1}^c if it is needed. Because L_k has better condition than $\bar{L}_k$, solving (5.7) will probably also give better numerical results than solving (5.4).

In theory the Lanczos iteration will stop with some $\beta_{k+1} = 0$, and then $x_k^c = x_k^L = x$, but in practice it is rare to have even a very small β_{k+1} , and some other stopping criterion must be used; here x_k^c and x_k^L will be different, and the one which gives the smaller residual would usually be chosen. x_k^c is often a much better approximation to x than x_k^L , and so (5.11) is usually carried out at the end of the iteration; there is no facility for doing this in the version of our algorithm described by Lawson [6], but it is included in the Fortran subroutine SYMMLQ in this report. Note that W_k has orthonormal columns, so that if

(5.12) $$d_k^L \equiv x - x_k^L \quad , \quad d_k^c \equiv x - x_k^c \quad ,$$

then d_k^L , but not d_k^c , must decrease in 2-norm every step. Thus x_k^L is the best approximation to x lying in the space spanned by $w_1, \ldots, w_k$, and is monotonically increasing in size every step; apparently this space is usually not as good an approximation space as that spanned by $w_1, \ldots, w_{k-1}, \bar{w}_k$.

For the algorithm using (5.7) and (5.10) to be theoretically well defined it is still necessary to show that there is no possibility of L_k being singular. Now from the discussion following equation (2.5) and the choice of v_1 in (3.1), we see that methods based on (3.4) will only be useful when $r = 0$ in (2.1), in which case (3.2) shows that $v_i \in \mathcal{R}(A)$, $i = 1,\dots,k$; but the only possibility of L_k being singular is if $\beta_{k+1} = 0$, giving $AV_k = V_kT_k$ in (3.3), from which we see that T_k cannot be singular. We see then that $L_k = \bar{L}_k = T_kQ_k^T$ cannot be singular in (5.7), and therefore $z_k = \bar{z}_k$ must be well defined at the final step, even if $\beta_{k+1} = 0$.

In any practical computation we will be interested in monitoring the size of the residual, usually to decide when to terminate, so when $\bar{L}_k$ is nonsingular we see from (3.1), (3.4), and (3.3), that

$$(5.13) \qquad r_k^c = b - Ax_k^c = \beta_1v_1 - AV_ky_k$$

$$= \beta_1v_1 - V_kT_ky_k - \beta_{k+1}v_{k+1}e_k^Ty_k \;\; = -\beta_{k+1}\eta_{kk}v_{k+1}$$

where η_{kk} is the k-th element of y_k . The vector y_k is not directly available in the computation, but since $T_k = T_k^T = Q_k^T\bar{L}_k^T$, equation (3.4) gives

$$(5.14) \qquad \bar{L}_k^Ty_k = \beta_1Q_ke_1$$

and from the last element of each side

$$(5.15) \qquad \bar{\gamma}_k\eta_{kk} = \beta_1s_1s_2\cdots s_{k-1}$$

so with (5.6)

$$(5.16) \qquad r_k^c = -(\beta_1s_1s_2\cdots s_k/c_k)v_{k+1} \quad .$$

Thus $\| r_k^c \|$ is directly available without ever forming x_k^c , and in fact it will be shown in a later paper that when rounding errors are present the norm of the residual using (5.16) is within $O(\varepsilon)\|A\|\,\|x\|$ of the true residual norm corresponding to the computed x_k^c , where ε specifies the relative accuracy of floating-point computation.

A slightly longer algebraic manipulation shows that

$$(5.17) \qquad r_k^L = b - Ax_k^L = \gamma_{k+1}\zeta_{k+1}v_{k+1} - \varepsilon_{k+2}\zeta_k v_{k+2}$$

so that

$$(5.18) \qquad \| r_k^L \|^2 = \gamma_{k+1}^2\zeta_{k+1}^2 + \varepsilon_{k+2}^2\zeta_k^2$$

is directly available during the computation, and may be used to decide whether to exit with x_k^c or x_k^L . Finally it is theoretically interesting to compare these two approximations to x . From (5.11), (5.10), and (5.8) it follows that

$$x_k^c = x_k^L + \zeta_k(\bar{w}_k / c_k - w_k)$$

but from (5.9) $\bar{w}_k = c_k w_k + s_k \bar{w}_{k+1}$, giving

$$(5.19) \qquad x_k^c = x_k^L + (\zeta_k s_k / c_k)\bar{w}_{k+1} \quad ,$$

and since from (5.2) $W_k^T \bar{w}_{k+1} = 0$ we see from (5.10) that $\bar{w}_{k+1}$ and x_k^L are orthogonal, so that

$$(5.20) \qquad \| x_k^L \| \le \| x_k^c \| \quad .$$

For later reference we shall call the method of this section Algorithm SYMMLQ.

6. The Minimum Residual Method

We will now examine the simplifications that result when we use the Lanczos vectors in the minimum residual method described in Section 2 (b). From (3.3) we see that

$$(6.1) \qquad V_k^T A^2 V_k = T_k^2 + \beta_{k+1}^2 e_k e_k^T \quad ,$$

$$(6.2) \qquad V_k^T A b = \beta_1 V_k^T A v_1 = \beta_1 T_k e_1 \quad .$$

The matrix in (6.1) is pentadiagonal and at least positive semidefinite, and so could be used directly in (2.7) with the Cholesky decomposition, in a very similar manner to the method of conjugate gradients. Forming the matrix in (6.1) and then factorizing would lead to an unnecessary loss of accuracy, but fortunately there is a simpler approach.

If we carry out the orthogonal factorization in (5.5) and use (5.6) we see that

$$(6.3) \qquad T_k^2 + \beta_{k+1}^2 e_k e_k^T = \bar{L}_k \bar{L}_k^T + \beta_{k+1}^2 e_k e_k^T = L_k L_k^T \quad ,$$

so that we have the Cholesky factor directly from T_k . In (2.7) we then have to solve

$$(6.4) \qquad L_k L_k^T u_k = \beta_1 \bar{L}_k Q_k e_1 \quad .$$

But since from (5.5) and (5.6)

$$(6.5) \qquad \bar{L}_k = L_k D_k \quad , \quad D_k \equiv \mathrm{diag}(1,1,\ldots,1,c_k) \quad ,$$

and since L_k is non-singular, (6.4) gives

$$(6.6) \qquad L_k^T u_k = \beta_1 D_k Q_k e_1 \equiv (\tau_1,\ldots,\tau_k)^T \equiv t_k \quad ,$$

$$(6.7) \qquad \tau_1 = \beta_1 c_1 \quad , \quad \tau_i = \beta_1 s_1 s_2 \cdots s_{i-1} c_i \quad , \quad i = 2,\ldots,k \quad ,$$

so there is minimal error in computing $L_k^T u_k$. Clearly u_k cannot be found until the algorithm is completed, but it is not really needed; instead we form

$$(6.8) \qquad M_k = [m_1,\ldots,m_k] = V_k L_k^{-T}$$

column by column (cf. (4.3)), and then in (2.7)

$$(6.9) \qquad x_k^M = V_k u_k = V_k L_k^{-T} L_k^T u_k = M_k t_k$$

where t_k is developed in (6.7), and the superscript M shows this is the vector which gives the minimum residual. Again it can be seen that previous vectors need not be held, and this is ideal for very large sparse matrices.

Note that much of the ill-conditioning suggested by (2.7) has been avoided, but nevertheless, as k increases the condition number of L_k in (6.8) approaches that of A , so that if A is ill-conditioned then some of the vectors m_k arising in (6.8) could be very large and somewhat in error, leading to errors in x_k^M in (6.9). In fact this minimum residual method has been found to suffer a little computationally on very poorly conditioned problems, whereas no such trouble has been found with the method in Section 5. This is probably because the vectors w_i in (5.10) are theoretically orthonormal.

The minimum residual method could also have been derived by considering solving (3.4) using a QR factorization of T . For since from (5.1) $T_k^T = T_k = Q_k^T \bar{L}_k^T$, (3.4) becomes

(6.10) $\tilde{L}_k^T y_k = \beta_1 Q_k e_1 \quad , \quad x_k^c = V_k(\bar{L}_k)^{-T} \tilde{L}_k^T y_k$

and a small change to make the computation slightly faster by using (6.5) leads to

(6.11) $x_k^M = V_k L_k^{-T}(\beta_1 D_k Q_k e_1) \quad .$

In fact x_k^c could be found from (6.10), or via (6.11), but neither way is as accurate as the method in Section 5, for the same reasons that the minimum residual method is suspect.

The minimum residual method will later be referred to as method MINRES.

7. Some Properties of the Minimum Residual Method

The minimum residual method described in Section 6 does not give as accurate results as the method in Section 5 when the problem is very ill-conditioned, but it still appears to give very good results in other cases. It can also be used for inconsistent equations whereas the method as described in Section 5 cannot. Furthermore, as the only way we have of deciding when to terminate the iteration is by testing the size of the residual, the method which minimizes this is likely to take fewer iterations than other methods. The other methods occasionally took a significant percentage more iterations than the minimum residual method, and so it will be of interest to examine the latter further.

It is straightforward to compare x_k^M with x_k^c, for (3.4) and (6.8) give

$$x_k^c = V_k L_k^{-T} L_k^T y_k = M_k L_k^T y_k \quad ,$$

$$Q_k T_k y_k = \bar{L}_k^T y_k = \beta_1 Q_k e_1 \quad ,$$

so with (6.5) and (6.6)

$$(7.1) \qquad x_k^c = M_k D_k^{-1} \bar{L}_k^T y_k = M_k D_k^{-2} t_k$$

$$= x_k^M + \tau_k (c_k^{-2} - 1) m_k = x_k^M + \tau_k (s_k / c_k)^2 m_k \quad .$$

Note that x_k^c can easily be obtained during the computation of x_k^M, but the reverse is not true. The m_k and w_k are related in a simple manner, for if the Lanczos process stops with $\beta_{m+1} = 0$, then $AV_m = V_m T_m$, $T_m = L_m Q_m$, and $M_m = V_m L_m^{-T}$, so

$$(7.2) \qquad AM_m = V_m T_m L_m^{-T} = V_m Q_m^T = W_m \quad .$$

Using this result with (7.1) gives

$$(7.3) \qquad r_k^M - r_k^c \equiv A(x_k^c - x_k^M) = \tau_k(s_k/c_k)^2 w_k$$

so that with (5.16) and (6.7)

$$r_k^M \equiv b - Ax_k^M = \beta_1 s_1 s_2 \dots s_k(s_k w_k - v_{k+1})/c_k$$

but from (5.9) $v_{k+1} = s_k w_k - c_k \bar{w}_{k+1}$, so

$$(7.4) \qquad r_k^M = \beta_1 s_1 s_2 \dots s_k \bar{w}_{k+1} \quad , \quad \|\bar{w}_{k+1}\| = 1 \quad ,$$

which shows clearly how the residual norm decreases each step. Also using (5.16)

$$(7.5) \qquad \|r_k^M\| = |c_k| \, \|r_k^c\| < \|r_k^c\|$$

except at the last step, where theoretically $s_k = 0$.

It is interesting to note from (5.16) and (7.4) that the size of the residuals r_k^c and r_k^M are given directly by the size of b and the LQ decomposition of T_{k+1} , and so are immediately available whichever of these algorithms is used. Equation (5.18) shows that $\|r_k^L\|$ is also available if (5.7) is solved.

8. The Linear Least Squares Problem

For general matrices A and C, the constrained least squares problem

$$\min \|b - Ax\|_2$$

subject to

$$Cx = d$$

can be converted to the symmetric indefinite system of linear equations

$$(8.1) \qquad \begin{bmatrix} I & & A \\ & & C \\ A^T & C^T & \end{bmatrix} \begin{bmatrix} r \\ \ell \\ x \end{bmatrix} = \begin{bmatrix} b \\ d \\ 0 \end{bmatrix}$$

where r is the vector of residuals and ℓ is the set of Lagrange multipliers. If A is $m \times n$ and C is $p \times n$, system (8.1) has a unique solution if $\text{rank}(C) = p$ and $\text{rank}\begin{pmatrix} A \\ C \end{pmatrix} = n$. In this case the procedure SYMMLQ can be applied directly, and should be efficient if the matrices A and C are large and sparse.

If there are no constraints, system (8.1) reduces to

$$(8.2) \qquad \begin{bmatrix} I & A \\ A^T & \end{bmatrix} \begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

which again could be solved by SYMMLQ given any starting point (r_0, x_0). However, computation and storage requirements can be halved by using a special choice of r_0 as follows. If we define

$$r_0 = b - Ax_0$$

for any given x_0, then the solution of (8.2) is given by

$$r = r_0 + \delta r$$
$$x = x_0 + \delta x$$

where

$$(8.3) \qquad \begin{bmatrix} I & A \\ A^T & \end{bmatrix} \begin{bmatrix} \delta r \\ \delta x \end{bmatrix} = - \begin{bmatrix} 0 \\ A^T r_0 \end{bmatrix} \quad .$$

When the Lanczos process is started with the vector $\begin{bmatrix} 0 \\ A^T r_0 \end{bmatrix}$ of (8.3) it can readily be shown that the orthogonal matrix V has zeros alternately in the top and bottom halves of its columns, while the tridiagonal matrix T has alternately 0 and 1 on its diagonal. The resulting simplification in the Lanczos process leads directly to the bidiagonalization procedure used by Paige [10, Section 4]. A corresponding simplification occurs in the factorization $T = LQ$ as performed in SYMMLQ, and this leads to a new algorithm for solving the least squares problem.

Without loss of generality we shall henceforth take $x_0 = 0$, $r_0 = b$. The bidiagonalization procedure, using $A^T b$ as initial vector, is defined as follows:

$$(8.4) \qquad \begin{array}{lll} \text{(a)} & \beta_1 u_1 = A^T b \ , & \alpha_1 v_1 = Au_1 \ ; \\ \text{(b)} & \beta_i u_i = A^T v_{i-1} - \alpha_{i-1} u_{i-1} \ , & \alpha_i v_i = Au_i - \beta_i v_{i-1} \ , \\ & (i = 2,3,\ldots,k) \ . & \end{array}$$

The scalars $\alpha_i > 0$, $\beta_i > 0$ are chosen so that $\|u_i\|_2 = \|v_i\|_2 = 1$. It is shown in [10, Section 2] that if

$$U = [u_1, \ldots, u_k] \ , \qquad J = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ & \alpha_2 & \beta_3 & & \\ & & \cdot & \cdot & \\ & & & \alpha_{k-1} & \beta_k \\ & & & & \alpha_k \end{bmatrix} \ ,$$

$$V = [v_1, \ldots, v_k] \ ,$$

then

$$(8.5)\qquad \begin{aligned} U^TU &= V^TV = I \ , \\ AU &= VJ \ , \\ A^TV &= UJ^T + \beta_{k+1}u_{k+1}e_k^T \end{aligned}$$

and in [10, Section 4] it is shown that none of the α_i can be zero, and that the minimum-length least squares solution of (8.2) is given by the equations

$$(8.6)\qquad J^Ty = \beta_1e_1 \ , \quad Jp = y \ , \quad x = Up \ , \quad r = b - Vy$$

where k is the first integer for which $\beta_{k+1} = 0$ in (8.5). With $\begin{pmatrix} \delta r \\ \delta x \end{pmatrix} = \begin{pmatrix} -Vy \\ x \end{pmatrix}$ this also solves equations (8.3). Now the elements of y can be computed as the bidiagonalization proceeds but p cannot be found similarly. In [10, Section 4] Paige computed successive columns of $M = UJ^{-1}$ by forward substitution, giving the algorithm implied by the following equations:

<u>Algorithm LSCG</u>

$$(8.7)\qquad J^Ty = \beta_1e_1 \ , \quad J^TM^T = U^T \ , \quad x = My \ .$$

Here x is built up progressively as a linear combination of the columns of M, the k-th approximation x_k minimizing the 2-norm of the residual $r_k = b - Ax_k$ over all vectors x_k lying in the space spanned by the first k columns of U.

LSCG is very straightforward and economical, and Lawson in his survey of algorithms of this type [6] has indicated his intent to test it more widely. It does have one possible failing, and that is if J in $M = UJ^{-1}$ is poorly conditioned then the columns of M could be very large, and serious cancellation could occur in forming $x = My$. This is the same sort of trouble that occurred with the use of algorithm MINRES in Section 6, and led us to favor SYMMLQ for symmetric systems. For similar reasons, SYMMLQ adapted for least squares may well be more numerically stable than LSCG, so it will be examined further.

The simplification resulting when SYMMLQ is applied to the least squares problem leads us to the orthogonal factorization

$$(8.8) \qquad J = LQ \quad , \quad Q^TQ = I \quad , \quad L \text{ lower-bidiagonal} \quad ,$$

and hence to the algorithm implied by the following equations:

Algorithm LSLQ

$$(8.9) \qquad J^Ty = \beta_1 e_1 \quad , \quad Lz = y \quad , \quad W = UQ^T \quad , \quad x = Wz$$

where x can still be built up from the columns of W as the algorithm progresses. This algorithm takes a little more storage and computation per step than algorithm LSCG, but it has the advantage that $W^TW = I$ which ensures that there is negligible cancellation in forming $x = Wz$.

The quantities involved in the k-th step of algorithm LSLQ are exactly analogous to those given in equations (5.1), ..., (5.11). We have

$$J_k = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ & \alpha_2 & \beta_3 & & \\ & & \cdot & \cdot & \\ & & & \alpha_{k-1} & \beta_k \\ & & & & \alpha_k \end{bmatrix} = \bar{L}_k Q_k \quad ,$$

$$\bar{L}_k = \begin{bmatrix} \gamma_1 & & & & \\ \delta_2 & \gamma_2 & & & \\ & \delta_3 & \gamma_3 & & \\ & & \cdot & \cdot & \\ & & & \delta_k & \bar{\gamma}_k \end{bmatrix} \quad , \tag{8.10}$$

$$Q_k^T = Q_{1,2} Q_{2,3} \cdots Q_{k-1,k} \quad , \quad Q_k^T Q_k = I \quad ,$$

$$U_k = [u_1, \ldots, u_{k-1}, u_k] \quad ,$$

$$\bar{W}_k = [w_1, \ldots, w_{k-1}, \bar{w}_k] = U_k Q_k^T \quad ,$$

$$W_k = [w_1, \ldots, w_{k-1}, w_k] \quad ,$$

and L_k is defined as the $k \times k$ leading part of $\bar{L}_{k+1}$. The system

$$J_k^T y_k = \beta_1 e_1 \tag{8.11}$$

is solved by forward substitution, and the points that would be computed by algorithm LSCG in (8.7) are given by

$$\bar{L}_k \bar{z}_k = y_k \quad , \quad x_k^c = \bar{W}_k \bar{z}_k \quad , \tag{8.12}$$

but as with SYMMLQ it is more efficient to work with the sequence defined by

(8.13) $$L_k z_k = y_k \quad , \quad x_k^L = W_k z_k$$

since x_k^L can be obtained cheaply from x_{k-1}^L as in (5.10). At all stages it is possible to move from x_k^L to x_{k+1}^c as in (5.11).

In order to find a stopping criterion it is natural to define

$$r_k^c \equiv b - Ax_k^c \quad , \quad r_k^L \equiv b - Ax_k^L$$

and to monitor the size of the vectors $A^T r_k^c$, $A^T r_k^L$. From equations (8.4), (8.10), ..., (8.13) we can show that

$$A^T r_k^c = -\beta_{k+1}\eta_k u_{k+1} \quad ,$$

$$A^T r_{k-1}^L = \alpha_k \bar{\gamma}_k \bar{\zeta}_k u_k - \beta_{k+1}\delta_k \zeta_{k-1} u_{k+1}$$

where $y_k \equiv (\eta_1, \ldots, \eta_k)^T$ and $\bar{z}_k \equiv (\zeta_1, \ldots, \zeta_{k-1}, \bar{\zeta}_k)^T$, so that

(8.14) $$\|A^T r_k^c\|^2 = (\beta_{k+1}\eta_k)^2 \quad ,$$

(8.15) $$\|A^T r_{k-1}^L\|^2 = (\alpha_k \bar{\gamma}_k \bar{\zeta}_k)^2 + (\beta_{k+1}\delta_k \zeta_{k-1})^2 \quad .$$

There is no obvious relationship between (8.14) and (8.15), but both can be readily calculated at the (k+1)-st stage of the bidiagonalization.

The final requirement is to estimate a reasonable size for these quantities, given that the computation will be carried out with some finite machine precision ε . In the case of SYMMLQ (Section 5) we would expect the vector x_k^L to be an acceptable approximation to the solution of $Ax = b$ if $\|b - Ax_k^L\|$ were smaller than $\|A\|\|x\|\varepsilon$. Since $V_k^T A V_k = T_k$ and $x_k^L = W_k z_k$ with V_k and W_k orthogonal, we can use $\|T_k\|$ and $\|z_k\|$ as estimates of $\|A\|$ and $\|x\|$ respectively, and in practice we terminate if either $\|r_k^L\|$ or $\|r_{k+1}^c\|$ (as estimated from (5.13), (5.18)) is smaller than $\|T_k\|_1 \|z_k\|_2 \varepsilon$.

Similarly with LSLQ, the residuals for system (8.3) involve Ax and $A^T\delta r$, where the approximations to δr are $\delta r_k^c = \delta r_k^L = -V_k y_k$. Since $V_k^T A U_k = J_k$ and $x_k^L = W_k z_k$ with U_k, V_k and W_k orthogonal, we can use

$$\|J_k\| \quad , \quad \|z_k\| \quad , \quad \|y_k\|$$

to estimate

$$\|A\| \quad , \quad \|x\| \quad , \quad \|\delta r\|$$

respectively. In practice we terminate if either $\|A^T r_k^L\|$ or $\|A^T r_{k+1}^c\|$ (as estimated from (8.15), (8.14)) is smaller than $\|J_k\|_1(\|y_k\|_2^2 + \|z_k\|_2^2)^{1/2}\varepsilon$.

9. Computational Experience

Algorithms SYMMLQ, MINRES and LSLQ have been programmed and tested on various systems of equations in order to obtain an impression of their numerical properties. A comparison has also beeen made in some cases with Reid's version 2 of the conjugate gradients method (CGM) [11]. We make the following observations.

(1) On symmetric positive definite systems, SYMMLQ gives essentially the same results as CGM. For example, the problem involving the Laplacian matrix of order 4080 $(15 \times 16 \times 17$ grid) was solved with SYMMLQ under the same conditions as described by Reid [11] for CGM, vis. single precision on the IBM System/360. A graph of $\|x - x_k^L\|$ lagged markedly behind the curve for CGM shown in [11, Figure 3], but SYMMLQ terminated at the same point as CGM by taking a final step from x_k^L to x_{k+1}^c as in equation (5.11). For test purposes all points x_{k+1}^c were computed from the points x_k^L, and the quantities $\|x - x_k^c\|$ were seen to follow the curve in [11, Figure 3] almost exactly.

(2) Although SYMMLQ obtains the same final point as CGM, it is clear that for positive definite systems CGM is to be preferred as it is more efficient.

(3) The variant of CGM described in Section 4 gave almost identical results for positive definite matrices as CGM in [1]. This confirms that the derivation of CGM from the Lanczos vectors and the Cholesky factorization of T_k is computationally similar to CGM, aside from their mathematical equivalence.

(4) Algorithm MINRES has behaved well on some examples involving the 2-dimensional Laplacian matrix, giving a rapid and very smooth decrease

in both $\|r_k^M\|$ and $\|x - x_k^M\|$. On other very ill-conditioned problems the estimate of $\|r_k^M\|$ in (7.4) decreased steadily but departed from the true $\|r_k^M\|$ and thus caused premature termination. In such cases it was also observed that if iterations were continued, the true $\|r_k^M\|$ stayed essentially constant while the true error $\|x - x_k^M\|$ continued to decrease until it reached quite an acceptably low level.

(5) An excellent application of SYMMLQ and MINRES is in solving symmetric systems of the form $(A - \mu I)x = b$ in the style of inverse iteration, since if μ is near an interior eigenvalue λ of A , the matrix $A - \mu I$ is indefinite. If μ is sufficiently close to λ and b is chosen appropriately then the computed x will be a good approximation to an eigenvector of A , and in practice it appears that the number of iterations required by SYMMLQ or MINRES is very small.

(6) Figure 1 illustrates the behavior of SYMMLQ on a symmetric system $(B^2 - \mu I)x = b$ of order $n = 50$, where $\mu = \sqrt{3}$ is <u>not</u> near an eigenvalue of B^2 but was chosen to make the system indefinite. The matrix B is tridiagonal with typical non-zero row elements $(-1, 2, -1)$, so that B^2 is pentadiagonal with typical row $(1, -4, 6, -4, 1)$. Computation was performed on a Burroughs B6700 with floating-point precision $\epsilon = 8^{-12} = 1.455 \times 10^{-11}$.

Estimates of the size of the residual vectors r_k^L , r_k^c and r_k^M are all available from SYMMLQ, and these were used to give estimates of $\log_{10}\|r_k^L\|$, $\log_{10}\|r_k^c\|$, $\log_{10}\|r_k^M\|$ which are plotted in Figure 1 against iteration number k . Of interest is the sharp reduction in residual obtained by taking a final step from x_{32}^L to the CGM point x_{33}^c (see dotted line in Figure 1). Note also the sharp jumps in $\|r_k^c\|$ at $k = 11$ and 24. These indicate regions of instability in the CGM sequence x_k^c, as described more fully in the next section, and if the standard method of conjugate gradients were used to compute the points x_k^c it is to be expected that the iterates $\|r_k^c\|$ would diverge from the path shown.

The final residual norm obtained was 7.83×10^{-9}, while the computed estimate of $\|r_{33}^c\|$ was 7.65×10^{-9}. This illustrates that the computed estimate of $\|r_k^c\|$ remains a good measure of the residual for the *computed* point x_k^c, in spite of the fact that the computed points are significantly different from those that would be obtained with exact computation. The same is true of the computed estimate of $\|r_k^L\|$, and similarly for the estimates of $\|A^T r_k^c\|$, $\|A^T r_k^L\|$ in algorithm LSLQ.

(7) Algorithm LSLQ has been tested on least squares problems of the form

$$\begin{bmatrix} r_1 \\ r_2 \end{bmatrix} + \begin{bmatrix} D \\ D \end{bmatrix} x = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

where $D = \mathrm{diag}(d_i)$ is an $n \times n$ diagonal matrix with $d_i = (i/n)^p$ for some integer $p > 0$. The simple form of the problem allows the condition number n^p to be altered easily, and the exact solution to be found from $Dx = (b_1 + b_2)/2$. The system is compatible iff $b_1 = b_2$, but incompatibility has no computational effect since the bidiagonalisation is essentially the same as if the problem were the compatible system $2Dx = b_1 + b_2$.

On a series of problems with $n = 20$ and $p = 3,4,\ldots,8$ the number of iterations taken by LSLQ (machine precision $\varepsilon = 10^{-11}$ approximately) were 78, 79, 77, 71, 64, 60, which shows a distinct *decrease* as the condition number increases from 10^4 to 10^{10} . This may or may not be typical of realistic problems, but it illustrates a tendency of the algorithm to ignore singular values which are smaller than $\epsilon^{1/2}$ (in this case, to ignore any $d_i < 10^{-5}$), and to converge on the minimum-length solution of a modified problem of correspondingly lower rank. This effect has also been observed by C. L. Lawson (private communication). In the above sequence of problems the number of negligible d_i was increasing steadily, and it could be expected that the number of iterations should decrease with the effectively decreasing rank.

10. Summary

We can now distinguish two reasons why the method of conjugate gradients (CGM) may fail to solve the symmetric system $Ax = b$ if A is not positive (or negative) definite. Recall that CGM attempts to compute a sequence of approximations $\{x_k^c\}$ satisfying

$$(10.1) \qquad x_k^c = V_k T_k^{-1} \beta_1 e_1 \quad , \quad k = 1,2,\ldots,m$$

for some $m > 0$, where $V_k^T V_k = I_k$, $V_k^T A V_k = T_k$ and the matrices T_k are tridiagonal, with T_{k+1} having T_k as its $k \times k$ principal submatrix. Recall also that CGM effectively computes the Cholesky factorization of each T_k . The basic problem we must contend with is the following:

> If A is indefinite, it is possible for some T_k to be singular or nearly singular $(k < m)$, <u>even if T_m is well conditioned</u>.

Now if T_k is nearly singular it happens that the Cholesky factorization of T_j is poorly determined numerically for all $j > k$. Even more seriously, if T_k is singular the corresponding point x_k^c in (10.1) is not properly defined. Thus we see that CGM's use of equation (10.1) is doubly doomed to failure.

The main features of algorithms SYMMLQ and MINRES can now be put into perspective. First of all, the orthogonal factorization $T_k = \bar{L}_k Q_k$ is well defined regardless of any near-singularities in T_j for $j \leq k$. In fact, as equations (5.11) and (5.19) show, we could compute the CGM sequence of points using

$$(10.2) \qquad x_k^c = x_{k-1}^c + (\bar{\zeta}_k - \zeta_{k-1} s_{k-1}/c_{k-1})\bar{w}_k$$

without the aid of the Cholesky factorization, but the more fundamental difficulty remains that x_k^c does not exist if T_k and hence $\bar{L}_k$ are singular. In such cases $\bar{\zeta}_k$ in (10.2) is undefined.

Secondly, then, instead of using $\bar{L}_k$ to compute the CGM sequence x_k^c, we define two new sequences x_k^L and x_k^M in terms of a matrix L_k which is the $k \times k$ principal submatrix of $\bar{L}_{k+1}$ and is guaranteed to be non-singular. By this means we effectively step around any irrelevant intermediate singularities in the CGM sequence (10.1). Some near-singularities are shown by the peaks in $\|r_k^c\|$ in Figure 1. We see from (5.6) and (7.5) that $\|r_k^c\| = \|r_k^M\| \cdot |\gamma_k| / |\bar{\gamma}_k|$ so we will get a large jump in $\|r_k^c\|$ when T_k is nearly singular but A is not.

Finally we note that the CGM points x_k^c are not to be discarded completely, since at least half of them <u>are</u> well defined by (10.1). This can be seen from the fact that if both T_k and T_{k+1} are singular then so are all T_j, $j \geq k$; hence if A is non-singular there cannot be two singular T_k's in a row. (In fact the limiting case is attained when A is the symmetric least squares matrix in (8.3), since in this case only the even numbered matrices $T_2, T_4, \ldots$ are non-singular.) Thus in algorithms SYMMLQ and LSLQ provision is made to terminate iterations at a CGM point whenever advantageous.

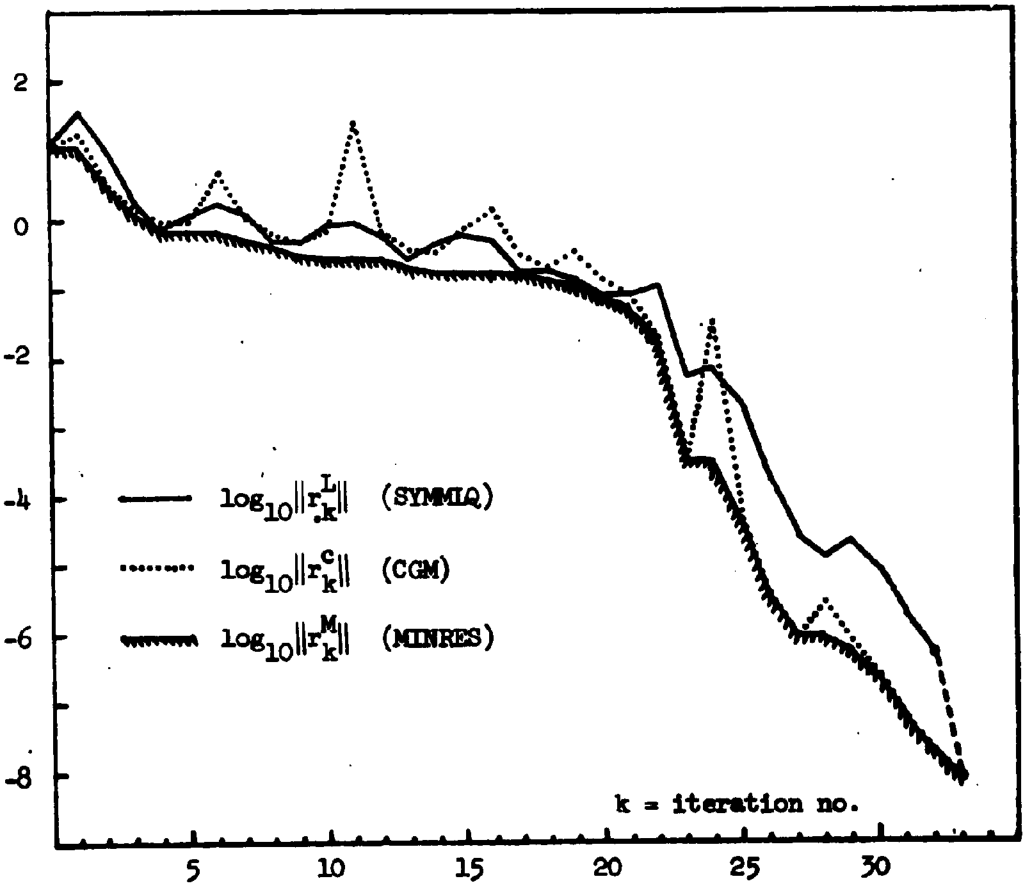

Figure 1. Solution of an indefinite symmetric system of equations $(B^2 - \mu I)x = b$, using subroutine SYMMLQ.

Notes

1. Dimension of system is $n = 50$; μ is not close to an eigenvalue of B^2 .
2. r_k^L , r_k^c , r_k^M are residual vectors for iteration paths taken by algorithms SYMMLQ, CGM, MINRES respectively. Estimates of the norms of these quantities are all computed by subroutine SYMMLQ.
3. Note large jumps in the size of $\|r_k^c\|$, reflecting intermediate near-singularities which would cause the standard method of conjugate gradients to break down.

Acknowledgments

We would like very much to thank John Reid for discussions about this work, and Gene Golub for his comments on this work and his hospitality at Stanford. We also gratefully acknowledge use of the computing facilities at the Stanford Linear Accelerator Center where earlier versions of our programs were developed.

Keywords

linear equations
least squares
sparse matrices
conjugate gradients
Lanczos process
bidiagonalization

AMS 1970 subject classifications

65F10
65F20

References

[1] Å. Björck and G. H. Golub, "Iterative Refinement of Linear Least Squares Solutions by Householder Transformations," BIT 7 (1967), pp. 322-337.

[2] M. R. Hestenes and E. Stiefel, "Methods of Conjugate Gradients for Solving Linear Systems," J. Res. Nat. Bur. Standards, 49, pp. 409-436, 1952.

[3] A. S. Householder, The Theory of Matrices in Numerical Analysis, Blaisdell Publishing Co., 1964, pp. 139-141.

[4] C. Lanczos, "An iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators," J. Res. Nat. Bur. Standards, 45, pp. 255-282, 1950.

[5] C. Lanczos, "Solution of Systems of Linear Equations by Minimized Iterations," J. Res. Nat. Bur. Standards, 49, pp. 33-53, 1952.

[6] C. L. Lawson, "Sparse Matrix Methods Based on Orthogonality and Conjugacy," Tech. Memo. 33-627, Jet Propulsion Laboratory, Pasadena, California, June 1973.

[7] D. G. Luenberger, "Hyperbolic Pairs in the Method of Conjugate Gradients," SIAM J. Appl. Math., 17 (1969), pp. 1263-1267.

[8] D. G. Luenberger, "The Conjugate Residual Method for Constrained Minimization Problems," SIAM J. Numer. Anal. 7 (1970), pp. 390-398.

[9] C. C. Paige, "Computational Variants of the Lanczos Method for the Eigenproblem," J. Inst. Maths Applics, 10 (1972), pp. 373-381.

[10] C. C. Paige, "Bidiagonalization of Matrices and Solution of Linear Equations," Technical Report No. CS 295, Computer Science Dept., Stanford University, 1972. SIAM J. Numer. Anal., Vol. 11, No. 1, March 1974.

[11] J. K. Reid, "On the Method of Conjugate Gradients for the Solution of Large Sparse Systems of Linear Equations," Proceedings of a conference on "Large Sparse Sets of Linear Equations," Academic Press, 1971.

[12] I. H. Siegel, "Deferment of Computation in the Method of Least Squares," Maths Comput., 19 (1965), pp. 329-331.

Appendix

The following are listings of Fortran subroutines SYMMLQ and LSLQ, along with subroutine NORM which is used by both. These routines were developed on a Burroughs B6700 at the Victorial University of Wellington. For machines with shorter word-length the routines should preferably be converted to double precision. This can be achieved by changing REAL to REAL*8 or DOUBLE PRECISION, ABS to DABS, and SQRT to DSQRT throughout.

As noted in the listings, it is assumed that subroutines ATIMES and ATRANS are available for computing products of the form Au and A^Tv respectively. These subroutines could be included as parameters to SYMMLQ and LSLQ, with appropriate use of the EXTERNAL statement in the calling program.

A positive value of the parameter ISTOP indicates that iteration was terminated at a point x_k^c (see text). A negative value indicates that the final solution is a point x_k^L.

```
      SUBROUTINE SYMMLQ( N,X,B,P,V1,V2,W,MACHEP,ACCY,ITNMAX,ISTOP )
      INTEGER  N, ITNMAX, ISTOP
      REAL     X(N), B(N), P(N), V1(N), V2(N), W(N), MACHEP, ACCY
      REAL     ALPHA, BETA, GAMMA, DELTA, EPSLN, CS, SN, D1, D2, Z,
     *         GBAR, DBAR, ZBAR, OLDB, EPS, EPSA, EPSX, S, T,
     *         NORMA, NORMX2, LQNORM, CGNORM, QRNORM, BESTNM
C     ------------------------------------------------------------------
C
C     SOLVES THE SYSTEM OF LINEAR EQUATIONS
C              A*X = B
C     WHERE  A  IS AN  N*N  MATRIX WHICH IS SYMMETRIC
C     BUT NOT NECESSARILY POSITIVE DEFINITE.
C     FOR EFFICIENCY  A  SHOULD BE SPARSE.
C
C     PARAMETERS:
C
C     N        THE DIMENSION OF  A.
C     X        AN N-VECTOR, CONTAINING AN INITIAL APPROXIMATION
C              TO  X  ON ENTRY (USUALLY  X = 0), AND THE FINAL
C              APPROXIMATION TO  X  ON EXIT.
C     B        AN N-VECTOR CONTAINING THE RHS VECTOR  B.
C     P,V1,V2,W
C              N-VECTORS FOR WORK-SPACE.
C     MACHEP   THE MACHINE PRECISION.
C     ACCY     A USER-SPECIFIED TOLERANCE.  ITERATION IS
C              TERMINATED IF IT APPEARS THAT  NORM(R) <= ACCY,
C              WHERE  R  IS THE RESIDUAL VECTOR  B - A*X.
C     ITNMAX   LIMIT ON THE NUMBER OF ITERATIONS.
C     ISTOP    INDICATES THE REASON FOR TERMINATION.
C              ABS(ISTOP)  RETURNS ONE OF THE FOLLOWING VALUES:
C          1 => NORM(R)  WAS REDUCED BELOW THE TOLERANCE  ACCY.
C          2 => NORM(R)  WAS REDUCED TO A REASONABLE LEVEL.
C          3 => THE LIMIT ON ITERATIONS WAS REACHED BEFORE THE
C              PREVIOUS CRITERIA WERE SATISFIED.
C
C     THE STATEMENT
C              CALL ATIMES( X, P, N )
C     SHOULD GIVE THE PRODUCT
C              P = A*X.
C     ------------------------------------------------------------------
```

```
C
C
      WRITE(6, 1000) N
      EPS = 8.0*MACHEP
C
C
C     COMPUTE RESIDUAL VECTOR  B - A*X
C     AND INITIATE THE LANCZOS PROCESS
C
      CALL ATIMES( X, P, N )
      DO 10 I = 1, N
         V1(I) = B(I) - P(I)
   10 CONTINUE
      CALL NORM( V1, N, EPS, D1 )
      BRNORM = D1
C
C
C     SECOND ITERATION OF LANCZOS
C
      CALL ATIMES( V1, P, N )
      ALPHA = 0.0
      DO 20 I = 1, N
         W(I) = V1(I)
         ALPHA = V1(I)*P(I) + ALPHA
   20 CONTINUE
C
      DO 30 I = 1, N
         V2(I) = P(I) - ALPHA*V1(I)
   30 CONTINUE
      CALL NORM( V2, N, EPS, BETA )
C
C
C     INITIALIZE OTHER QUANTITIES
C
      GBAR = ALPHA
      DBAR = BETA
      D2 = 0.0
      NORMX2 = 0.0
      NORMA = ABS(ALPHA) + BETA
      EPSA = NORMA*EPS
      EPSX = EPSA
      ITN = 0
      ISTOP = 0
```

```
C
C     -------------------------------------------------
C     MAIN ITERATION LOOP
C     -------------------------------------------------
C
C
C     TEST FOR CONVERGENCE
C
   50 LQNORM = SQRT(D1**2 + D2**2)
      CGNORM = QRNORM*BETA/(ABS(GBAR)+EPSA)
      BESTNM = AMIN1(LQNORM, CGNORM)
      IF (ITN .EQ. ITNMAX)  ISTOP = 3
      IF (BESTNM .LE. EPSX) ISTOP = 2
      IF (BESTNM .LE. ACCY) ISTOP = 1
      IF (ISTOP .NE. 0) GO TO 100
      WRITE(6, 1010) ITN, X(1), LQNORM, CGNORM
C
C
C     COMPUTE THE NEXT COLUMN OF  V  (LANCZOS)
C
      CALL ATIMES( V2, P, N )
      ALPHA = 0.0
      DO 60 I = 1, N
         ALPHA = V2(I)*P(I) + ALPHA
   60 CONTINUE
C
      DO 70 I = 1, N
         T = V2(I)
         V2(I) = P(I) - ALPHA*T - BETA*V1(I)
         V1(I) = T
   70 CONTINUE
      OLDB = BETA
      CALL NORM( V2, N, EPSA, BETA )
C
C
C     COMPUTE PLANE ROTATION
C
      GAMMA = SQRT(GBAR**2 + OLDB**2)
      CS = GBAR/GAMMA
      SN = OLDB/GAMMA
      DELTA = CS*DBAR + SN*ALPHA
      GBAR  = SN*DBAR - CS*ALPHA
      EPSLN = SN*BETA
      DBAR = -CS*BETA
      QRNORM = SN*QRNORM
```

```
C
C
C     UPDATE APPROXIMATION TO  X
C
      Z = D1/GAMMA
      S = Z*CS
      T = Z*SN
      DO 80 I = 1, N
         X(I) = (W(I)*S + V1(I)*T) + X(I)
         W(I) = W(I)*SN - V1(I)*CS
   80 CONTINUE
C
C
C     ESTIMATE  NORM(A),  GO ROUND AGAIN
C
      S = OLDB + ABS(ALPHA) + BETA
      IF (NORMA .LT. S) NORMA = S
      NORMX2 = Z**2 + NORMX2
      EPSA = NORMA*EPS
      EPSX = SQRT(NORMX2)*EPSA
      D1 = D2 - DELTA*Z
      D2 = -EPSLN*Z
      ITN = ITN+1
      GO TO 50
C
C     ------------------------------------------------
C     END OF MAIN ITERATION LOOP
C     ------------------------------------------------
C
C
C     TEST FOR MOVE TO CG POINT
C
  100 IF (LQNORM .LE. CGNORM) ISTOP = -ISTOP
      IF (ISTOP .LT. 0) GO TO 120
      ZBAR = D1/GBAR
      DO 110 I = 1, N
         X(I) = W(I)*ZBAR + X(I)
  110 CONTINUE
```

```
C
C
C     DISPLAY STATUS AT END OF ITERATIONS
C
  120 WRITE(6, 1010) ITN, X(1), LQNORM, CGNORM
      WRITE(6, 1020) ITN, ISTOP, ACCY, EPSX, BESTNM
      WRITE(6, 1030) CGNORM, LQNORM, QRNORM
      RETURN
C
C
C
 1000 FORMAT( /, ' SYMMLQ.   DIMENSION OF SYSTEM:', I6, // )
 1010 FORMAT( I8, 1PE20.10, 1P6E15.5 )
 1020 FORMAT( '1',
     *   /, ' NO. OF ITERATIONS:       ', 18X, I10,
     *   /, ' STOPPING CONDITION WAS:', 18X, I10,
     *   /, ' NORM OF RESIDUAL WAS REQUIRED TO BE:', 1PE15.5,
     *   /, ' ESTIMATE OF REASONABLE NORM:         ', 1PE15.5,
     *   /, ' ESTIMATE OF NORM ACTUALLY OBTAINED: ', 1PE15.5 )
 1030 FORMAT( '0ESTIMATES OF NORM OF FINAL RESIDUAL:',
     *   /, ' COMPLETED  LQ:', 1PE15.5,
     *   /, ' INCOMPLETE LQ:', 1PE15.5,
     *   /, ' QR:            ', 1PE15.5 )
C
C     END OF SYMMLQ
      END
      SUBROUTINE NORM( V, N, EPS, BETA )
      INTEGER  N
      REAL     V(N), EPS, BETA, S
C
C     NORMALIZES THE VECTOR  V  AND RETURNS THE NORM
C     AS  BETA.  CALLED BY SUBROUTINES  SYMMLQ  AND  LSLQ.
C
      S = 0.0
      DO 10 I = 1, N
         S = V(I)**2 + S
   10 CONTINUE
      BETA = SQRT(S)
      IF (BETA .LT. EPS) BETA = EPS*0.5
C
      S = 1.0/BETA
      DO 20 I = 1, N
         V(I) = V(I)*S
   20 CONTINUE
      RETURN
C
C     END OF NORM
      END
      SUBROUTINE LSLQ( M,N,X,B,P,V,U,W,MACHEP,ACCY,ITNMAX,ISTOP )
      INTEGER  M, N, ITNMAX, ISTOP
      REAL     X(N), B(M), P(M), V(M), U(N), W(N), MACHEP, ACCY
      REAL     ALPHA, BETA, GAMMA, DELTA, CS, SN, D1, D2, Y, Z,
     *         GBAR, ZBAR, EPS, EPSA, EPSX, S, T,
     *         NORMA, NORMX2, LQNORM, CGNORM, BESTNM
C     ----------------------------------------------------------------
C
C     SOLVES THE LINEAR LEAST SQUARES PROBLEM
C              MINIMIZE  R'R,     R = B - A*X
C     WHERE  A  IS AN  M*N  MATRIX,  M>=N,  AND SHOULD BE SPARSE.
C
C     PARAMETERS:
C
C     M, N     DIMENSIONS OF THE MATRIX  A.
C     X        AN N-VECTOR, CONTAINING AN INITIAL APPROXIMATION
C              TO  X  ON ENTRY (USUALLY  X = 0), AND THE FINAL
C              APPROXIMATION TO  X  ON EXIT.
C     B        AN M-VECTOR CONTAINING THE RHS VECTOR  B.
C     P, V     TWO M-VECTORS FOR WORK-SPACE.
C     U, W     TWO N-VECTORS FOR WORK-SPACE.
C     MACHEP   THE MACHINE PRECISION.
C     ACCY     A USER-SPECIFIED TOLERANCE.  ITERATION IS
C              TERMINATED IF IT APPEARS THAT  NORM(A'R) <= ACCY.
C     ITNMAX   A LIMIT ON THE NUMBER OF ITERATIONS TO BE DONE.
C     ISTOP    INDICATES THE REASON FOR TERMINATION.
C              ABS(ISTOP)  RETURNS ONE OF THE FOLLOWING VALUES:
C           1 => NORM(A'R)  WAS REDUCED BELOW THE TOLERANCE  ACCY.
C           2 => NORM(A'R)  WAS REDUCED TO A REASONABLE LEVEL.
C           3 => THE LIMIT ON ITERATIONS WAS REACHED BEFORE THE
C              PREVIOUS CRITERIA WERE SATISFIED.
C
C     THE STATEMENTS
C              CALL ATIMES( U, P, M, N )
C              CALL ATRANS( V, P, M, N )
C     SHOULD GIVE THE PRODUCTS
C              P = A*U
C              P = A(TRANSPOSE)*V
C     RESPECTIVELY.
C     ----------------------------------------------------------------
```

```
C
C
      WRITE(6, 1000) M, N
      EPS = 8.0*MACHEP
C
C
C     COMPUTE RESIDUAL VECTOR  B - A*X
C     AND INITIATE THE BIDIAGONALIZATION
C
      CALL ATIMES( X, P, M, N )
      DO 10 I = 1, M
         P(I) = B(I) - P(I)
   10 CONTINUE
C
      CALL ATRANS( P, U, M, N )
      CALL NORM( U, N, EPS, BETA )
C
      CALL ATIMES( U, V, M, N )
      CALL NORM( V, M, EPS, ALPHA )
C
C
C     INITIALIZE OTHER QUANTITIES
C
      DO 20 I = 1, N
         W(I) = U(I)
   20 CONTINUE
      BBAR = ALPHA
      Y = BETA/ALPHA
      D1 = Y
      D2 = 0.0
      NORMX2 = 0.0
      NORMA = ALPHA
      EPSA = NORMA*EPS
      EPSX = EPSA
      ITN = 0
      ISTOP = 0
```

```
C
C     ----------------------------------------------------------
C     MAIN ITERATION LOOP
C     ----------------------------------------------------------
C
C
C     FIND NEW COLUMN OF  U
C
   50 CALL ATRANS( V, P, M, N )
      DO 60 I = 1, N
         U(I) = P(I) - ALPHA*U(I)
   60 CONTINUE
      CALL NORM( U, N, EPSA, BETA )
C
C
C     TEST FOR CONVERGENCE
C
      LQNORM = SQRT((ALPHA*D1)**2 + (BETA*D2)**2)
      CGNORM = ABS(BETA*Y)
      BESTNM = AMIN1(LQNORM, CGNORM)
      IF (ITN .EQ. ITNMAX)  ISTOP = 3
      IF (BESTNM .LE. EPSX) ISTOP = 2
      IF (BESTNM .LE. ACCY) ISTOP = 1
      IF (ISTOP .NE. 0) GO TO 100
      WRITE(6, 1010) ITN, X(1), LQNORM, CGNORM
C
C
C     FIND NEW COLUMN OF  V
C
      CALL ATIMES( U, P, M, N )
      DO 70 I = 1, M
         V(I) = P(I) - BETA*V(I)
   70 CONTINUE
      CALL NORM( V, M, EPSA, ALPHA )
C
C
C     COMPUTE PLANE ROTATION
C
      GAMMA = SQRT(GBAR**2 + BETA**2)
      CS = GBAR/GAMMA
      SN = BETA/GAMMA
      DELTA = SN*ALPHA
      GBAR = -CS*ALPHA
```

```
C
C
C     UPDATE APPROXIMATION TO  X
C
      Z = D1/GAMMA
      S = Z*CS
      T = Z*SN
      DO 80 I = 1, N
         X(I) = (W(I)*S + U(I)*T) + X(I)
         W(I) = W(I)*SN - U(I)*CS
   80 CONTINUE
C
C
C     ESTIMATE  NORM(A),  GO ROUND AGAIN
C
      IF (NORMA .LT. ALPHA+BETA) NORMA = ALPHA+BETA
      NORMX2 = Y**2 + Z**2 + NORMX2
      EPSA = NORMA*EPS
      EPSX = SQRT(NORMX2)*EPSA
      Y = -BETA*Y/ALPHA
      D2 = DELTA*Z
      D1 = Y - D2
      ITN = ITN+1
      GO TO 50
C
C     ------------------------------------------------------------
C     END OF MAIN ITERATION LOOP
C     ------------------------------------------------------------
C
C
C     TEST FOR MOVE TO CG POINT
C
  100 IF (LQNORM .LE. CGNORM) ISTOP = -ISTOP
      IF (ISTOP .LT. 0) GO TO 120
      ZBAR = D1/GBAR
      DO 110 I = 1, N
         X(I) = W(I)*ZBAR + X(I)
  110 CONTINUE
```

```
C
C
C     DISPLAY STATUS AT END OF ITERATIONS
C
  120 WRITE(6, 1010) ITN, X(1), LUNORM, CGNORM
      WRITE(6, 1020) ITN, ISTOP, ACCY, EPSX, BESTNM
      RETURN
C
C
C
 1000 FORMAT( /, ' LSLQ.   DIMENSIONS OF SYSTEM:', 2I8, // )
 1010 FORMAT( I8, 1PE20.10, 1P6E15.5 )
 1020 FORMAT( '1',
     *   /, ' NO. OF ITERATIONS:      ', 18X, I10,
     *   /, ' STOPPING CONDITION WAS:', 18X, I10,
     *   /, ' NORM OF  A(**T)R  WAS REQD TO BE:    ', 1PE15.5,
     *   /, ' ESTIMATE OF REASONABLE NORM:         ', 1PE15.5,
     *   /, ' ESTIMATE OF NORM ACTUALLY OBTAINED:  ', 1PE15.5 )
C
C     END OF LSLQ
      END
```