MATHEMATICS OF COMPUTATION, VOLUME 26, NUMBER 120, OCTOBER 1972

Modification Methods for Inverting Matrices and Solving Systems of Linear Algebraic Equations

By D. Goldfarb*

Abstract. Modification methods for inverting matrices and solving systems of linear algebraic equations are developed from Broyden's rank-one modification formula. Several algorithms are presented that take as few, or nearly as few, arithmetic operations as Gaussian elimination and are well suited for the handling of data. The effect of rounding errors is discussed briefly.

Some of these algorithms are essentially equivalent to, or "compact" forms of, such known methods as Sherman and Morrison's modification method, Hestenes' biorthogonalization method, Gauss-Jordan elimination, Aitken's below-the-line elimination method, Purcell's vector method, and its equivalent, Pietrzykowski's projection method, and the bordering method. These methods are thus shown to be directly related to each other.

Iterative methods and methods for inverting symmetric matrices are also given, as are the results of some computational experiments.

1. Introduction. This paper is concerned with the related problems of inverting matrices and solving linear nonhomogeneous algebraic systems of equations by a class of direct methods which we shall call *modification methods*. The adjective *modification* is used to describe these methods, for they are all based upon the modification of a matrix and its inverse by a matrix of rank one. Most of these methods are direct—that is, a solution to the problem is obtained by using a finite number of elementary arithmetic operations. The general formula that these methods are based upon, however, is iterative in nature, and iterative methods are also given.

Modification methods have been developed and published by other authors. These include the methods of Sherman and Morrison [30], [31], and [32], and computational schemes and extensions based upon their work due to Woodbury [37], Wilf [34], [35], Zielke [38], Ershov [10] (as reported in Faddeev and Faddeeva [11]), and Kron [23]. Methods developed by the author are found in Section 2 (for inverting matrices), in Section 6 (for inverting symmetric matrices), in Section 7 (iterative methods), and in Sections 8, 9, and 10 (for solving simultaneous linear equations). At first it was thought that all of these methods were new. Subsequent comparisons with the literature, however, revealed that, although the theoretical development underlying these methods was new, most of them were essentially equivalent to, or were a compact form of, known techniques. These techniques include Sherman and Morrison's modification method [31], Hestenes' biorthogonalization method [17], [18], Gauss-Jordan elimination, Aitken's below-the-line elimina-

---

Received April 3, 1971.

AMS 1969 *subject classifications.* Primary 6535, 1510, 1515; Secondary 6510, 6580, 1525.

* This research was supported in part by City University Faculty Research Grant No. 1085 and the IBM New York Scientific Center.

tion method [1], Purcell's vector method [29] and its equivalent, Pietrzykowski's projection method [26], and the bordering method.

The relationships between these methods and those developed by the author are explored in Sections 4, 5, 9, and 11. As a result of this analysis several of these methods are shown to be related that were previously thought not to be.

Brief discussions of the computational aspects of the modification methods developed here are found in Sections 3 and 12. Two algorithms, algorithm (2.8) for inverting matrices and algorithm (10.2) for solving systems of linear equations, are particularly interesting, since both require no more arithmetic operations than Gaussian elimination. Both of these algorithms were coded in APL and were tested computationally on several numerical examples. These results are found in Section 13.

The purpose of this paper is not to encourage the rejection of Gaussian elimination in favor of modification methods. Many fine computer codes for the former method, incorporating scaling, pivoting, and other error-reducing procedures, are in use and are readily available. Rather the purpose of this paper is to present a comprehensive exposition of modification methods.

2. Generalized Modification Methods for Matrix Inversion. Consider the following algorithm based upon a general rank-one modification formula due to Broyden [4].

(0) Choose, as an initial approximation to A^{-1}, some nonsingular $n \times n$ matrix $B^{(0)}$. Let $x_1 \neq 0, \cdots, x_n \neq 0$ be some set of mutually orthogonal n-dimensional column vectors.

(i) Compute

$$B^{(k)} = B^{(k-1)} - \frac{(B^{(k-1)}Ax_{i_k} - x_k)x_k'B^{(k-1)}}{x_k'B^{(k-1)}Ax_{i_k}} \tag{2.1}$$

for $k = 1, 2, \cdots, n$ where $i_1, i_2, \cdots, i_n$ is some permutation of $1, 2, \cdots, n$.

The main result of this section is given by the following theorem.

THEOREM 2.1. *If* $x_k'B^{(k-1)}Ax_{i_k} \neq 0$, *for* $k = 1, 2, \cdots, n$, *then*

$$A^{-1} = RB^{(n)} \quad \textit{where } R = \sum_{k=1}^{n} x_{i_k}x_k'/||x_k||^2.$$

Proof. Substituting $Rx_k = x_{i_k}$ in (2.1) gives

$$B^{(k)} = B^{(k-1)} - \frac{(B^{(k-1)}AR - I)x_kx_k'B^{(k-1)}}{x_k'B^{(k-1)}ARx_k}, \qquad k = 1, \cdots, n. \tag{2.2}$$

Using the well-known formula, due to Bartlett [2],

$$(M + uv')^{-1} = M^{-1} - M^{-1}uv'M^{-1}/(1 + v'M^{-1}u) \tag{2.3}$$

relating rank-one modifications of a matrix and its inverse, we obtain

$$(B^{(k)})^{-1} = (B^{(k-1)})^{-1} + \frac{(AR - (B^{(k-1)})^{-1})x_kx_k'}{||x_k||^2}, \qquad k = 1, \cdots, n.$$

Hence,

$$(B^{(k)})^{-1} - AR = ((B^{(k-1)})^{-1} - AR)P_k \tag{2.4}$$

where $P_k = I - x_k x_k'/||x_k||^2$ is an orthogonal projection matrix. It follows that

$$(B^{(n)})^{-1} - AR = ((B^{(0)})^{-1} - AR)\cdot P_1 \cdots P_n. \tag{2.5}$$

But $P_1 \cdots P_n = 0$, where 0 is a matrix of all zeros. Therefore,

$$B^{(n)} = (AR)^{-1} = R^{-1}A^{-1}$$

and the theorem is proved.

The matrix R behaves very much like a permutation matrix and in fact, is one if the orthogonal set of vectors $\{x_i\}$ is the set of unit coordinate vectors $\{e_i\}$. If $x_{i_k} = x_k$ for all k, then R is obviously the identity I. If x_k and x_{i_k} are interchanged in (2.1), then $A^{-1} = R^{-1}B^{(n)}$, where $R^{-1} = \sum_{k=1}^{n} x_k x_{i_k}'/||x_{i_k}||^2$.

If algorithm (2.1) is to be at all useful, we should be able to choose an ordering of the orthogonal set of vectors $\{x_i\}$ and a permutation $i_1, i_2, \cdots, i_n$ so that the denominator in (2.1) does not vanish on any step. With this in mind we prove the following two lemmas. The first is a restatement of a result proved by Dennis [6, Theorem 5].

LEMMA 2.1. *If $B^{(k-1)}$ is a nonsingular $n \times n$ matrix and if $x_k'B^{(k-1)}Ax_{i_k} \neq 0$, then $B^{(k)}$ is nonsingular.*

Proof. Assume that $B^{(k)}$ is singular, i.e., $B^{(k)}u = 0$ for some $u \neq 0$. From (2.1) we have that

$$0 = x_k'B^{(k)}u = x_k'B^{(k-1)}u - x_k'\frac{(B^{(k-1)}Ax_{i_k} - x_k)x_k'B^{(k-1)}u}{x_k'B^{(k-1)}Ax_{i_k}},$$

$$0 = \frac{||x_k||^2 \cdot x_k'B^{(k-1)}u}{x_k'B^{(k-1)}Ax_{i_k}}.$$

Since the denominator is bounded and $x_k \neq 0$, $x_k'B^{(k-1)}u$ must equal zero. But this implies from (2.1) that $B^{(k)}u = B^{(k-1)}u = 0$, for $u \neq 0$ contradicting the hypothesis that $B^{(k-1)}$ is nonsingular. Hence, $B^{(k)}$ must also be nonsingular.

LEMMA 2.2. *$(B^{(k)}AR - I)U_{1,k} = 0$ where $U_{i,j} = [u_i, u_{i+1}, \cdots, u_j]$ and, $u_i = x_i/||x_i||$. $U_{i,j}$ and 0 are $n \times (j - i + 1)$ matrices, the latter having all zero elements.*

This result is an immediate consequence of (2.5) for $n = k$.

THEOREM 2.2. *If A is nonsingular, there exists an ordering of the vectors $\{x_j\}$ and a permutation $i_1, i_2, \cdots, i_n$ such that (2.1) is well defined (i.e., the denominator $x_k'B^{(k-1)}Ax_{i_k}$ does not vanish at the kth step).*

Proof. Let $U = U_{1,n} = [U_{1,k-1} \mid U_{k,n}] = [U_1 \mid U_2]$.

It is assumed that (2.1) was well defined on all steps preceding the kth step. $U'B^{(k-1)}ARU$ is nonsingular since U, A, R and $B^{(k-1)}$ are nonsingular. $B^{(k-1)}$ is nonsingular by Lemma 2.1 so long as $B^{(0)}$ is chosen nonsingular.

$$U'B^{(k-1)}ARU = \begin{bmatrix} U_1'B^{(k-1)}ARU_1 & U_1'B^{(k-1)}ARU_2 \\ \hline U_2'B^{(k-1)}ARU_1 & U_2'B^{(k-1)}ARU_2 \end{bmatrix}$$

$$= \begin{bmatrix} U_1'U_1 & U_1'B^{(k-1)}ARU_2 \\ \hline U_2'U_1 & U_2'B^{(k-1)}ARU_2 \end{bmatrix} = \begin{bmatrix} I_{k-1} & U_1'B^{(k-1)}ARU_2 \\ \hline 0 & U_2'B^{(k-1)}ARU_2 \end{bmatrix}$$

by Lemma 2.2, where I_{k-1} is a $(k-1)$st order identity matrix.

The possible denominators in (2.1) on the kth step, allowing for reordering the vectors $x_k, \cdots, x_n$ and the permutation $i_k, \cdots, i_n$, are just the elements of

$U_2'B^{(k-1)}ARU_2$. These cannot all be zero, for if they were, $U'B^{(k-1)}ARU$ would be singular. Since it is not singular, there must be some choice of $x_k, \cdots, x_n$ and some permutation that gives a nonvanishing denominator.

Actually, Theorem 2.2 is more restrictive than is necessary. On the kth step, if x_{i_k} is fixed there must be some choice x_l among the vectors $x_k, \cdots, x_n$ such that $x_l'B^{(k-1)}Ax_{i_k} \neq 0$. Otherwise the lth row of $U'B^{(k-1)}ARU$ would equal zero, contradicting the nonsingularity of A.

The most natural choice for the set of orthogonal vectors is the set of unit vectors $\{e_i\}$. With this choice, algorithm (2.1) becomes

(0) Choose an $n \times n$ nonsingular matrix $B^{(0)}$.

(i) Compute

$$B^{(k)} = B^{(k-1)} - \frac{(B^{(k-1)}A_k - e_{i_k})e_{i_k}'B^{(k-1)}}{e_{i_k}'B^{(k-1)}A_k}, \qquad k = 1, 2, \cdots, n, \tag{2.6}$$

where A_k denotes the kth column of A. If we let $b_{ij}^{(k)}$ and a_{ij} denote the elements of $B^{(k)}$ and A respectively, and if on the kth step $i_k = k$, (i) becomes

(i)′ Compute

$$c_i^{(k)} = \sum_{j=1}^{n} b_{ij}^{(k-1)}a_{jk}, \qquad i = 1, \cdots, n. \tag{2.6a}$$

(ii)′ Compute

$$b_{kj}^{(k)} = b_{kj}^{(k-1)}/c_k^{(k)}, \qquad j = 1, 2, \cdots, n, \tag{2.6b}$$

and

$$b_{ij}^{(k)} = b_{ij}^{(k-1)} - c_i^{(k)}b_{kj}^{(k)}, \qquad i, j = 1, 2, \cdots, n, \ i \neq k. \tag{2.6c}$$

In step (ii)′, $c_k^{(k)}$ may equal zero. Theorem 2.2 and the discussion following it only guarantee that not all the $c_i^{(k)}$, $i = k, \cdots, n$, computed in step (i)′ will be zero if A is nonsingular. Thus, row interchange may be necessary. In practice, the actual interchange of rows may be avoided through the use of a permutation vector. Rather than using $c_k^{(k)}$ as the denominator in (ii)′, it is advisable to use $c_{i_k}^{(k)}$ as the divisor, where

$$|c_{i_k}^{(k)}| = \max |c_i^{(k)}|, \qquad i \neq i_1, \cdots, i_{k-1}. \tag{2.7}$$

The subscript k would then be replaced by i_k and $i \neq k$ would be replaced by $i \neq i_k$ in (ii)′. This is the analog of *partial pivoting* in Gaussian elimination, i.e., of choosing as the pivot, an element of largest absolute value in the kth column. Henceforth, we shall assume that $i_k = k$ in (2.7).

The most economical choice for $B^{(0)}$ both in terms of computer storage and the number of operations performed is the identity I. This leads to the following algorithm.

For $k = 1, 2, \cdots, n$,

(i) compute

$$c_i^{(k)} = \sum_{j=1}^{k-1} b_{ij}^{(k-1)}a_{jk}, \qquad i = 1, \cdots, k-1, \tag{2.8a}$$

$$c_i^{(k)} = \sum_{j=1}^{k-1} b_{ij}^{(k-1)} a_{jk} + a_{ik}, \qquad i = k, \cdots, n. \tag{2.8b}$$

If $c_k^{(k)} = 0$, terminate. (A is singular.) Otherwise

(ii) compute

$$b_{kk}^{(k)} = 1/c_k^{(k)}, \tag{2.8c}$$

$$b_{kj}^{(k)} = b_{kk}^{(k)} b_{kj}^{(k-1)}, \qquad j = 1, \cdots, k-1, \tag{2.8d}$$

$$b_{ij}^{(k)} = b_{ij}^{(k-1)} - c_i^{(k)} b_{kj}^{(k)}, \qquad i = 1, \cdots, n;\ i \neq k,\ j = 1, \cdots, k-1 \tag{2.8e}$$

and

$$b_{ik}^{(k)} = -c_i^{(k)} b_{kk}^{(k)}, \qquad i = 1, \cdots, n;\ i \neq k. \tag{2.8f}$$

As in algorithm (2.6), *partial pivoting* may be used (with appropriate permutations of integers). It will be shown later that the numbers $c_j^{(k)}$, $j = k, \cdots, n$, are equal to the elements u_{kj} of the upper triangular matrix U of Gaussian elimination assuming exact arithmetic is used. Choosing i_k from (2.7) thus corresponds exactly to partial pivoting in Gaussian elimination.

Analogs of partial (row) pivoting and complete pivoting are also possible. These procedures require the computation of the inner products of all remaining columns of A (i.e., $n - k + 1$ columns at the kth step), with the kth row of $B^{(k-1)}$ in the former case and with the last $n - k + 1$ rows of $B^{(k-1)}$ in the latter case. As these greatly increase the amount of work required by the algorithm, they are not recommended.

An alternate form of algorithm (2.8) is obtained by replacing steps (2.8d)–(2.8f) by

$$b_{ik}^{(k)} = -c_i^{(k)} \cdot b_{kk}^{(k)}, \qquad i = 1, \cdots, n;\ i \neq k, \tag{2.8d'}$$

$$b_{ij}^{(k)} = b_{ij}^{(k-1)} + b_{ik}^{(k)} \cdot b_{kj}^{(k-1)}, \qquad i = 1, \cdots, n;\ i \neq k,\ j = 1, \cdots, k-1 \tag{2.8e'}$$

and

$$b_{kj}^{(k)} = b_{kk}^{(k)} \cdot b_{kj}^{(k-1)}, \qquad j = 1, \cdots, k-1. \tag{2.8f'}$$

These formulas require exactly the same number of multiplications, reciprocals, and additions and subtractions as formulas (2.8d)–(2.8f).

3. Computational Aspects. There are basically three important criteria for judging methods for inverting matrices on a computer. They are the number of arithmetic operations required, the storage and data handling requirements, and the susceptibility to rounding errors. A fourth, though less important criterion, is algorithmic complexity. Throughout this section, unless stated otherwise, we shall assume that $|c_k^{(k)}| \geqq |c_i^{(k)}|$ for all k and $i > k$; i.e., $i_k = k$. This is important for any analysis of numerical error, but has practically no effect on the storage and arithmetical requirements of the algorithms except for the general case of algorithm (2.1).

With regard to the first criterion, each iteration of algorithm (2.1) requires $4n^2 + n$ multiplications, n divisions, and $4n^2 - 2n$ additions and subtractions. Each iteration of algorithm (2.6) requires $2n^2 - n$ multiplications, n divisions, and $2n^2 - 2n$ additions and subtractions. Algorithms (2.1) and (2.6), respectively, require approxi-

mately 4 and 2 times as many operations as Gaussian elimination. Algorithm (2.6), however, is competitive with Hotelling's iterative formula [19], if used to improve an approximate inverse, obtained by it or some other method.

The kth iteration of algorithm (2.8) requires $n(2k-1)-1$ multiplications, 1 reciprocal, and $2(n-1)(k-1)$ additions and subtractions. For the complete inversion the totals are n^3-n multiplication, n reciprocals, and n^3-2n^2+n additions and subtractions. These operational counts are identical to those for Gaussian elimination. Although the method of modification has been discussed by several authors, this fact seems to have gone unnoticed.

The handling of data for these methods is particularly convenient, since single columns of A (or rows of A if we work with A') are needed on each iteration. The standard forms of the Gauss and Gauss-Jordan methods require the entire matrix on each iteration, while the original versions of the Crout [5] and Doolittle [7] methods require, on the kth iteration, all as yet uncalled for components of the kth column and row. An algorithm can also be given for decomposing A into LU that changes only the kth column of A on the kth step. (For example, see Fox [12, p. 103].)

Only some brief comments will be made here with regard to the effect of roundoff. A more complete analysis will appear in a later paper. On each iteration, the effect of the algorithm is to postmultiply the matrix difference $B^{(k-1)^{-1}}-A$ by an orthogonal projection operator. This suggests that the algorithm is not overly susceptible to rounding errors. Limited numerical experience (see Section 13) has tended to support this. On the other hand, even if partial pivoting is used, some of the ratios $c_i^{(k)}/c_k^{(k)}$ may be greater than one, as in Gauss-Jordan elimination, making it difficult to obtain error bounds. As discussed in Section 5, algorithm (2.8) and Gauss-Jordan elimination are very directly related. As in the Crout algorithm, the numbers $c_i^{(k)}$ in algorithms (2.6) and (2.8) should be computed as double-precision accumulated inner products before final rounding to single precision. This helps to keep the rounding errors small.

Lastly, the algorithms developed here are very simple to program and implement on a computer.

4. Relationship to Sherman and Morrison's Modification Method. Algorithm (2.1) is directly related to the method of modification [30], [31]. In fact, formula (2.6) is identical to the one given by Sherman and Morrison [31].

The method of modification (or reinforcement** as it is called by Faddeev and Faddeeva [11]) is based upon Bartlett's formula (2.3). We can invert a matrix A by considering it as the last matrix in a sequence of invertible matrices, $A^{(0)}, A^{(1)}, \cdots, A^{(n)}=A$, each differing from the preceding one by a matrix of rank one, and using formula (2.3) to compute $B^{(k)}=A^{(k)^{-1}}$ from $B^{(k-1)}=A^{(k-1)^{-1}}$. If $A^{(k)}$ and $A^{(k-1)}$ differ only in the kth column, and we choose the kth column of $A^{(k)}$ to equal the kth column of A, then $u=(A-A^{(k-1)})e_k$ and $v=e_k$ on the kth application of (2.3). Making the appropriate substitutions yields

$$B^{(k)}=B^{(k-1)}-\frac{B^{(k-1)}(A-A^{(k-1)})e_k e_k' B^{(k-1)}}{1+e_k' B^{(k-1)}(A-A^{(k-1)})e_k}. \tag{4.1}$$

Since $B^{(k)}A^{(k)}=I$ and $e_k'e_k=1$, it is obvious that this method is equivalent to

** One of the referees has pointed out that "completion" is probably a better translation of the Russian "popolnenija".

algorithm (2.6). If we choose the rank-one matrix difference between A^{k-1} and $A^{(k)}$ to be

$$(AR - A^{(k-1)})x_k x_k'/||x_k||^2 = (Ax_{i_k} - A^{(k-1)}x_k)x_k'/||x_k||^2$$

we obtain the equivalent of the general algorithm (2.1) by application of (2.3) iteratively.

Although (4.1) and (2.6) with $i_k = k$ are identical if exact arithmetic is used, Powell [27] has shown, both theoretically and numerically, that the latter is to be preferred because it tends to suppress the contribution from any errors in $B^{(k-1)}$. A similar statement holds for the more general algorithms. Specifically, if $B^{(k)}$ and $B^{(k-1)}$ are approximations to $A^{(k)^{-1}}$ and $A^{(k-1)^{-1}}$, where

$$A^{(k)} = A^{(k-1)} + (A - A^{(k-1)})x_k x_k'/||x_k||^2$$

then one finds that $B^{(k)^{-1}} - A^{(k)} = [B^{(k-1)^{-1}} - A^{(k-1)}]P_k$ using the Broyden formula (2.1) and $B^{(k)^{-1}} - A^{(k)} = B^{(k-1)^{-1}} - A^{(k-1)}$ using the Bartlett formula.

5. Relationship to Other Methods. One may view algorithm (2.6), i.e., Sherman and Morrison's method of modification, as a method for forming from the rows of $B^{(0)}$, a basis that is dual or biorthonormal to the basis consisting of the columns of A, since after the kth step $B^{(k)}A_j = e_j$ for $j = 1, 2, \cdots, k$. This dual basis is given by the rows of $B^{(n)}$. It is not surprising, therefore, that algorithm (2.6) is also identical to Hestenes' biorthogonalization method [18]. If the kth row of $B^{(k)}$ is left unchanged on the kth iteration, $B^{(k)}A_j = \delta_j e_j$ for $j = 1, \cdots, k$, where δ_j is a scalar, and the version of Hestenes' method described by Fox [12] is obtained. A^{-1} is then equal to $D^{-1}B^{(n)}$, where D is a diagonal matrix whose kth diagonal element equals δ_k.

Hestenes recommends choosing $B^{(0)} = A'$ [18]. With this choice of $B^{(0)}$ or, in fact, any $B^{(0)}$ such that $B^{(0)}A$ is positive definite, he shows that the denominator in formula (2.6b), i.e., $c_k^{(k)}$, does not vanish so long as A is nonsingular and exact arithmetic is used. For numerical considerations, it is still preferable to employ partial pivoting.

Hestenes also shows that his algorithm (i.e., algorithm (2.6) is equivalent to applying Gauss-Jordan elimination with pivot row division by the pivot element to the matrix $B^{(0)}A$. This can be seen by writing (2.6), with $i_k = k$, as $B^{(k)} = (I - v^{(k)}e_k')B^{(k-1)}$, where $v^{(k)} = (B^{(k-1)}A - I)e_k/e_k'B^{(k-1)}A_k$. Now $v^{(k)}$ is chosen so that

$$B^{(k)}A_k = (I - v^{(k)}e_k')B^{(k-1)}A_k = e_k.$$

By Lemma 2.2, $B^{(k)}A_j = e_j$ for $j < k$ as well. Thus

$$A^{-1} = B^{(n)} = (I - v^{(n)}e_n') \cdots (I - v^{(1)}e_1')B^{(0)}.$$

If $B^{(0)} = I$, then we have A^{-1} expressed exactly as it is by the Gauss-Jordan algorithm. However, in modification algorithms, we do not record the elements of the "reduced" matrix $B^{(k)}A$ but only those of $B^{(k)}$; i.e., the modification algorithm (2.8) is a "compact" form of Gauss-Jordan.

The form of Gauss-Jordan elimination described above is operationally identical to Aitken's below-the-line elimination method [1], applied to the partitioned matrix

$$\left[\begin{array}{c|c} A & I \\ \hline -I & 0 \end{array}\right].$$

Thus, algorithm (2.8) may also be viewed as a compact method for computing A^{-1} in the lower right corner in the below-the-line method.

After k steps of algorithm (2.8),

$$B^{(k)} = \begin{bmatrix} A_{11}^{-1} & 0 \\ -A_{21}A_{11}^{-1} & I_{m-n} \end{bmatrix}$$

and

$$B^{(k)}A = \begin{bmatrix} I_k & A_{11}^{-1}A_{12} \\ 0 & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{bmatrix},$$

where A_{11}, A_{12}, A_{21} and A_{22} are the appropriate $k \times k$, $k \times (n-k)$, $(n-k) \times k$, and $(n-k) \times (n-k)$ submatrices of the partitioned matrix

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}.$$

The connection between algorithms (2.8) and (2.6) and the inversion of a matrix by either successively bordering its submatrices or by partitioning is easily demonstrated. The relationship between Sherman and Morrison's modification method and the bordering method was first pointed out by Dwyer [9, pp. 292–294].

6. Symmetric Matrices. Both Greenstadt [16] and Powell [28] have independently developed a symmetric analog of Broyden's rank-one modification formula. Using Powell's [28] formula for updating the corresponding inverse, we arrive at the following general algorithm for inverting a symmetric matrix.

(0) Choose, as an initial approximation to A^{-1}, some nonsingular symmetric $n \times n$ matrix $B^{(0)}$. Let $x_1 \neq 0, \cdots, x_n \neq 0$ be some set of mutually orthogonal vectors.

(i) Compute

$$B^{(k)} = B^{(k-1)} + \frac{1}{\delta}\{B^{(k-1)}x_k[\beta y_k' - \alpha x_k'B^{(k-1)}] + y_k[\beta x_k'B^{(k-1)} - \gamma y_k']\}, \tag{6.1}$$

$$k = 1, 2, \cdots, n,$$

where $y_k = B^{(k-1)}Ax_k - x_k$, $\alpha = y_k'Ax_k$, $\beta = x_k'B^{(k-1)}Ax_k$, $\gamma = x_k'B^{(k-1)}x_k$ and $\delta = \alpha\gamma - \beta^2$. For simplicity we have assumed that pivoting is not required.

Using the generalized modification formula

$$(M + UV')^{-1} = M^{-1} - M^{-1}U(I + V'M^{-1}U)^{-1}V'M^{-1} \tag{6.2}$$

where both U and V are $n \times 2$ matrices, we can show that if the denominator does not vanish on any step, then

$$B^{(k)^{-1}} - A = P_k \cdot P_{k-1} \cdots P_1(B^{(0)^{-1}} - A)P_1 \cdots P_{k-1}P_k \quad \text{and} \quad A^{-1} = B^{(n)}.$$

Identity (6.2) is usually attributed to Woodbury [37]. However, Zielke [38] notes that it first appeared in a paper by Duncan [8].

Each step of algorithm (6.1) requires $4n^2 + O(n)$ multiplications and divisions. If the set of orthogonal vectors $\{x_k\}$ are the coordinate vectors $\{e_k\}$, these requirements become $2n^2 + O(n)$, and formula (6.1) reduces to Zielke's formula [38].

Other modification methods that preserve the symmetry of $B^{(k)}$, the matrix approximation to A^{-1}, if A is symmetric, have been described by Householder [20], Wilf [34], [35], and Bodewig [3].

7. Improving an Approximate Inverse. Algorithms (2.1) or (2.6) can also be used in an iterative manner to improve an approximation to A^{-1} that has been computed using these algorithms or some other inversion technique. If $B^{(0)}$ is a fairly good approximation to A^{-1}, pivoting will not be necessary, for $B^{(0)}A \approx I$, and the remarks of Section 5, with regard to choices of $B^{(0)}$ such that $B^{(0)}A$ is positive definite, apply. Algorithms (2.1) and (2.6) can be applied in integral multiples of n iterations, as previously described in Section 3. The algorithms, however, may be terminated when some norm of $B^{(k-1)}A - I$ becomes small enough.

One variant of algorithm (2.1) that is entirely iterative is to compute, on the kth step, the eigenvector of $B^{(k-1)}A - I$ corresponding to the eigenvalue of maximum modulus. This vector can then be used in formula (2.1) in place of x_k (and x_{ik}). Caution must be used not to choose an eigenvector with eigenvalue -1, for this would cause the denominator in (2.1) to vanish. This could only happen if the matrix $B^{(k-1)}A$ has a zero eigenvalue; i.e., A is singular. The eigenvalue, ξ, need only be computed approximately, using, for example, the power method. After one iteration of algorithm (2.1) with $x_k = \xi$, $(B^{(k)}A - I)\xi = 0$, and since $||P_k|| \leqq 1$, $||B^{(k)-1} - A|| \leqq ||B^{(k)-1} - A||$ from Eq. (2.4). Thus by these criteria, $B^{(k)}$ is a better approximation to A^{-1} than is $B^{(k-1)}$. This is a time-consuming procedure. However, it might be worth performing one such iteration to improve an approximate inverse, $B^{(n)}$, obtained by algorithm (2.6), if the norm of the residual matrix corresponding to $B^{(n)}$ was greater than some acceptable value.

If $B^{(0)}A$ is symmetric, as would be the case for $B^{(0)} = I$ and A symmetric, or $B^{(0)} = A'$, then the above procedure results in an n-step algorithm for inverting A. (This assumes that the eigenvalue problem is solved exactly, at each step.) If $B^{(0)}A$ is symmetric, there exists an orthogonal set of eigenvectors of $B^{(0)}A$, $\xi_1, \xi_2, \cdots, \xi_n$ corresponding to eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_n$. Therefore, from (2.1),

$$B^{(1)} = B^{(0)} + \left(\frac{1 - \lambda_1}{\lambda_1}\right)\frac{\xi_1\xi_1'B^{(0)}}{\xi_1'\xi_1} \tag{7.1}$$

and

$$B^{(1)}A = B^{(0)}A + (1 - \lambda_1)\xi_1\xi_1'/\xi_1'\xi_1, \tag{7.2}$$

$B^{(1)}A$ is obviously symmetric and has eigenvectors $\xi_1, \xi_2, \cdots, \xi_n$ corresponding to eigenvalues $1, \lambda_2, \cdots, \lambda_n$. When formula (7.2), with sub and superscripts 0 and 1 replaced by $k - 1$ and k, is used in conjunction with the power method to determine the eigenvalues and eigenvectors of $(B^{(k-1)}A - I)$, and hence those of $B^{(k-1)}A$, for $k = 1, \cdots, n$, the result is obviously a variant of the orthogonalization method for determining the intermediate eigenvalues and eigenvectors of a matrix. Formula (7.1) iteratively builds up the inverse matrix as the eigenvalues and eigenvectors are determined.

8. Modification Methods for the Solution of Linear Equations. In this section, we shall present several algorithms for solving systems of n linear equations in n

unknowns that are based upon the rank-one modification formula (2.6). More general methods based upon the more general formula (2.1) can also be derived. However, these methods would require a much greater amount of computational effort than those that will be presented.

In Section 2, we proved that, allowing for row permutations of $B^{(k)}$, i.e., partial pivoting, and assuming that $B^{(0)}$ and A were both nonsingular, $B^{(n)} = R^{-1}A^{-1}$ where R is a permutation matrix. In order to simplify the following presentation, we shall assume that partial pivoting is unnecessary. This will be the case if all leading square submatrices of A are nonsingular. Under this assumption, the solution to the system of linear equations

$$Ax = b, \tag{8.1}$$

is given by $B^{(n)}b$.

Postmultiplying both sides of Eq. (2.6) (with $i_k = k$) by b and letting $B^{(k)}b = x^{(k)}$ for all k yields the following iterative formula:

$$x^{(k)} = x^{(k-1)} - (B^{(k-1)}A_k - e_k)x_k^{(k-1)}/e_k'B^{(k-1)}A_k \tag{8.2}$$

where $x_k^{(k-1)}$ is the kth component of the vector $x^{(k-1)}$. If we set $x^{(0)} = B^{(0)}b$, $x^{(n)}$ will equal $A^{-1}b$, the required solution. This, as it stands, is not a very efficient method since the full matrix $B^{(k-1)}$ is needed at the kth step. Even if the iterations were begun with $B^{(0)} = I$, and hence $x^{(0)} = b$, the method would still require approximately three times as much work as required by Gaussian elimination to solve (8.1).

In formula (8.2), we do not need the matrix $B^{(k-1)}$ on the kth step, but rather the vector $B^{(k-1)}A_k$. Defining the vectors $d_j^{(k)} = B^{(k)}A_j$, $0 \leqq k < j \leqq n$, and $d_{n+1}^{(k)} = x^{(k)} = B^{(k)}b$, $0 \leqq k \leqq n$, we obtain from (8.2) and (2.6) the following iterative scheme for computing the $d_j^{(k)}$:

$$d_j^{(k)} = d_j^{(k-1)} - \frac{(d_k^{(k-1)} - e_k)\, d_{kj}^{(k-1)}}{d_{kk}^{(k-1)}}, \tag{8.3}$$

$$j = k+1, \cdots, n+1, k = 1, \cdots, n,$$

where $d_{k,j}^{(k-1)}$ denotes the kth component of $d_j^{(k-1)}$. The solution to (8.1) is given by $x = d_{n+1}^{(n)}$. Finally, (8.3) can be written as

$$d_{kj}^{(k)} = d_{kj}^{(k-1)}/d_{kk}^{(k-1)}, \qquad j = k+1, \cdots, n+1, \tag{8.4a}$$

and

$$d_{ij}^{(k)} = d_{ij}^{(k-1)} - d_{ik}^{(k-1)} \cdot d_{kj}^{(k)}, \tag{8.4b}$$

$$j = k+1, \cdots, n+1, i = 1, \cdots, n; i \neq k.$$

These formulas are just those of Gauss-Jordan elimination with division of the pivot row by the pivot element, and are operationally equivalent to Aitken's below-the-line elimination method applied to

$$\left[\begin{array}{c|c} A & b \\ \hline -I & 0 \end{array}\right] \quad \text{or} \quad \left[\begin{array}{c|c} A & -b \\ \hline I & 0 \end{array}\right].$$

There is an alternative scheme that lies somewhere between computing the full matrix $B^{(k)}$ and computing the vectors $d_j^{(k)} = B^{(k)}A_j$ on the kth iteration. We can compute the last $n - k$ rows of $B^{(k)}$ on the kth step, and, at the same time, compute the first k rows of the matrix $B^{(k)}[A_{k+1}, A_{k+2}, \cdots, A_n] = [d_{k+1}^{(k)}, \cdots, d_n^{(k)}]$. This method [15], however, requires $\frac{2}{3}n^3 + \frac{1}{2}n^2 + \frac{5}{6}n$ multiplications and divisions, and $\frac{2}{3}n^3 - \frac{3}{2}n^2 + \frac{5}{6}n$ additions and subtractions, whereas the corresponding counts for Gauss-Jordan elimination, i.e., algorithm (8.4), are $\frac{1}{2}n^3 + \frac{1}{2}n^2$ and $\frac{1}{2}n^3 - \frac{1}{2}n$.

If we multiply both sides of formula (2.6) on the left by b' rather than on the right by b, we arrive at a method for computing the solution to $x'A = b'$ or equivalently, $A'x = b$, since $b'B^{(n)} = b'A^{-1}$. Specifically, if we let $x^{(k)\prime} = b'B^{(k)}$, for $k = 0, 1, \cdots, n$, we obtain

$$x^{(k)\prime} = x^{(k-1)\prime} - \frac{(x^{(k-1)\prime}A_k - b_k)e_k'B^{(k-1)}}{e_k'B^{(k-1)}A_k}, \qquad k = 1, \cdots, n. \tag{8.5}$$

We can, of course, use this iterative formula together with (2.6) to solve (8.1). We need only replace A by A' in both formulas. Furthermore, observe that only rows $k + 1, \cdots, n$ of $B^{(k)}$ are needed after the kth step. Therefore, after transposition of formulas (2.6) and (8.5), and using $B^{(k)}$ to denote what was previously indicated by $B^{(k)\prime}$, we obtain the following algorithm, starting with $B^{(0)} = I$:

(0) Set $x^{(0)} = b$.

(i) Compute

$$\begin{aligned} c_j^{(k)} &= \sum_{p=1}^{k-1} a_{kp}b_{pj}^{(k-1)} + a_{kj}, \qquad j = k, \cdots, n, \\ c_{n+1}^{(k)} &= \sum_{p=1}^{n} a_{kp}x_p^{(k-1)} - b_k, \end{aligned} \tag{8.6a}$$

$$b_{kj}^{(k)} = -c_j^{(k)}/c_k^{(k)}, \qquad j = k + 1, \cdots, n, \tag{8.6b}$$

$$b_{ij}^{(k)} = b_{ij}^{(k-1)} + b_{ik}^{(k-1)}b_{kj}^{(k)}, \qquad i = 1, \cdots, k - 1, j = k + 1, \cdots, n, \tag{8.6c}$$

$$x_k^{(k)} = b_k - c_{n+1}^{(k)}/c_k^{(k)} \tag{8.6d}$$

and

$$x_i^{(k)} = x_i^{(k-1)} - (c_{n+1}^{(k)}/c_k^{(k)})b_{ik}^{(k-1)}, \qquad i = 1, \cdots, k - 1, \tag{8.6e}$$

for $k = 1, 2, \cdots, n$.

This algorithm can be expressed more compactly by defining the $n \times (n + 1)$ and $(n + 1) \times (n + 1)$ partitioned matrices

$$\tilde{A} = (A \mid -b) \tag{8.7a}$$

and

$$\tilde{B}^{(k)} = \left[\begin{array}{c|c} B^{(k)} & x^{(k)} \\ \hline 0 & 1 \end{array}\right]. \tag{8.7b}$$

We can also make the algorithm more general by choosing, in step (0), $B^{(0)}$ to be any nonsingular matrix and $x^{(0)} = B^{(0)}b$. Step (i) then becomes

(i′) Compute

$$c_j^{(k)} = \sum_{p=1}^{n+1} \tilde{a}_{kp}\tilde{b}_{pj}^{(k-1)}, \qquad j = k, \cdots, n+1, \tag{8.8a}$$

and

$$\tilde{b}_{ij}^{(k)} = \tilde{b}_{ij}^{(k-1)} - \tilde{b}_{ik}^{(k-1)}(c_j^{(k)}/c_k^{(k)}), \qquad i = 1, \cdots, n,\ j = k+1, \cdots, n+1 \tag{8.8b}$$

for $k = 1, 2, \cdots, n$. After n iterations, the solution of (8.1) is given by $x_i^{(n)} = \tilde{b}_{i,n+1}^{(n)}$, $i = 1, \cdots, n$. Notice that only the columns of $B^{(k)}$ that are needed in subsequent steps are updated.

As in algorithms (2.6) and (2.8), incorporation of a *partial pivoting* scheme in the above two algorithms is advisable, and sometimes necessary. This is accomplished exactly as it is in the inversion algorithms.

Algorithm (8.6) requires a total of $\frac{1}{3}n^3 + \frac{3}{2}n^2 + \frac{1}{6}n$ multiplications and divisions and $\frac{1}{3}n^3 + n^2 + \frac{2}{3}n$ additions and subtractions to solve an $n \times n$ system. The more general algorithm (8.8) requires $n^3 + \frac{5}{2}n^2 + \frac{1}{2}n$ multiplications and divisions and $n^3 + \frac{3}{2}n^2 - \frac{1}{2}n$ additions and subtractions.

Klyuev and Kokovkin-Shcherbak proved that $\frac{1}{3}n^3 + n^2 - \frac{1}{3}n$ multiplications and divisions and $\frac{1}{3}n^3 + \frac{1}{2}n^2 - \frac{5}{6}n$ additions and subtractions are the minimal numbers of these operations that are required to solve an n by n system of linear equations when using methods which employ linear combinations of rows or columns [22]. Strassen has shown that one can do even better than this, if one does not restrict oneself to operations on rows and columns as a whole [33]. Algorithm (8.6) is in the class of methods treated by Klyuev and Kokovkin-Shcherbak and requires only $\frac{1}{2}n(n+1)$ more multiplications and divisions and $\frac{1}{2}n^2 + \frac{5}{6}n$ more additions and subtractions than the minimum amount required by such methods.

The more general algorithm (8.8) may be used instead of algorithm (8.6) at the expense of approximately tripling the number of operations that need be performed. If one chooses $B^{(0)} = A'$, it can be shown that $c_k^{(k)} \neq 0$, for $k = 1, 2, \cdots, n$, assuming that A is nonsingular and that exact arithmetic is used. (See the discussion of Hestenes' biorthogonalization method in Section 5.)

Let us consider algorithm (8.8) in greater detail. Each iteration can be expressed in the terminology of (8.7) as

$$\tilde{B}^{(k)} = \tilde{B}^{(k-1)} - \tilde{B}^{(k-1)}\tilde{C}^{(k)}, \qquad k = 1, 2, \cdots, n, \tag{8.9}$$

where $\tilde{C}^{(k)}$ is an $(n+1) \times (n+1)$ matrix, all of whose elements are zero except for the last $n - k + 1$ elements of the kth row which are

$$\tilde{c}_{kj}^{(k)} = c_j^{(k)}/c_k^{(k)}, \qquad j = k+1, k+2, \cdots, n+1. \tag{8.10}$$

The $c_j^{(k)}$, $j = k, \cdots, n+1$, are, of course, computed according to (8.8a).

Adding the n matrix equations, (8.9) yields

$$\tilde{B}^{(n)} = \tilde{B}^{(0)} - (\tilde{B}^{(0)}\tilde{C}^{(1)} + \tilde{B}^{(1)}\tilde{C}^{(2)} + \cdots + \tilde{B}^{(n-1)}\tilde{C}^{(n)}).$$

The matrix product $\tilde{B}^{(k-1)}\tilde{C}^{(k)}$ depends only upon the kth column of $\tilde{B}^{(k-1)}$ and affects only the last $n - k + 1$ columns of $\tilde{B}^{(k)}$ in Eq. (8.9). Therefore the kth columns of $\tilde{B}^{(k-1)}, \tilde{B}^{(k)}, \cdots, \tilde{B}^{(n)}$ are identical and $\tilde{B}^{(n)} = \tilde{B}^{(0)} - \tilde{B}^{(n)}(\tilde{C}^{(1)} + \tilde{C}^{(2)} + \cdots + \tilde{C}^{(n)})$

or

$$\tilde{B}^{(n)}\tilde{C} = \tilde{B}^{(0)} \tag{8.11}$$

where $\tilde{C}$ is unit upper triangular with elements $\tilde{c}_{ij} = c_i^{(i)}/c_i^{(i)}$, $1 \leqq i \leqq j \leqq n+1$.

The matrices $\tilde{B}^{(k)}$, $k = 1, 2, \cdots, n$, satisfy

THEOREM 8.1.

$$e_i'\tilde{A}\tilde{B}^{(k)}e_j = 0, \qquad 1 \leqq i \leqq k \leqq n,\ i < j \leqq n+1.$$

Proof. The theorem is vacuously true for $k = 0$. Assume that it is true for $k = p - 1$. Then, from (8.9),

$$\begin{aligned} e_i'\tilde{A}\tilde{B}^{(p)}e_j &= e_i'\tilde{A}\tilde{B}^{(p-1)}(I - \tilde{C}^{(p)})e_j \\ &= \begin{cases} e_i'\tilde{A}\tilde{B}^{(p-1)}e_j - (c_j^{(p)}/c_p^{(p)})e_i'\tilde{A}\tilde{B}^{(p-1)}e_p, & j > p, \\ e_i'\tilde{A}\tilde{B}^{(p-1)}e_j, & j \leqq p, \end{cases} \end{aligned}$$

for $1 \leqq i \leqq p \leqq n$ and $i < j \leqq n+1$, since $\tilde{C}^{(p)}e_j = (c_j^{(p)}/c_p^{(p)})e_p$ for $p < j \leqq n+1$ and is zero otherwise. By the induction hypothesis $e_i'\tilde{A}\tilde{B}^{(p-1)}e_j = 0$ for $1 \leqq i \leqq p - 1 \leqq n$ and $i < j \leqq n+1$ and hence, so is the above for these values of i and j. But for $i = p < j \leqq n+1$, we have from (8.8a) and the above that $e_p'\tilde{A}\tilde{B}^{(p)}e_j = \tilde{c}_j^{(p)} - (c_j^{(p)}/c_p^{(p)})c_p^{(p)} = 0$, concluding the proof.

From this theorem it follows that the kth approximation to the solution of (8.1), computed by algorithm (8.8), satisfies the first k equations in (8.1). In algorithm (8.6),

$$\tilde{B}^{(0)} = \left[\begin{array}{c|c} I & b \\ \hline 0 & 1 \end{array}\right], \tag{8.12}$$

all subsequent matrices $\tilde{B}^{(1)}, \cdots, \tilde{B}^{(n)}$ are unit upper triangular, and the last $n - k$ components of the kth approximate solution equal $b_{k+1}, \cdots, b_n$.

THEOREM 8.2. *If A is nonsingular, then at the kth step of algorithm* (8.8), *not all $c_p^{(k)}$, $p = k, \cdots, n$, equal zero.*

Proof. From the previous theorem and (8.7a) and (8.7b) we know that

$$\sum_{j=1}^{n+1} \tilde{a}_{ij}\tilde{b}_{jp}^{(k-1)} = \sum_{j=1}^{n} a_{ij}b_{jp}^{(k-1)} = 0 \quad \text{for } i = 1, \cdots, k-1,\ p = k, \cdots, n.$$

If

$$c_p^{(k)} = \sum_{j=1}^{n+1} \tilde{a}_{kj}\tilde{b}_{jp}^{(k-1)} = \sum_{j=1}^{n} a_{kj}b_{jp}^{(k-1)} = 0 \quad \text{for } p = k, \cdots, n,$$

then

$$\sum_{j=1}^{n} a_{ij}b_{jp}^{(k-1)} = 0 \quad \text{for } i = 1, \cdots, k,\ p = k, \cdots, n.$$

This implies that the first k rows of the matrix A are orthogonal to the last $n - k + 1$ columns of $B^{(k-1)}$. Since A is nonsingular this implies that $B^{(k-1)}$, and hence from (8.7b), that $\tilde{B}^{(k-1)}$ are singular. But as in the discussion preceding Eq. (8.11) we can show that $\tilde{B}^{(0)}$, which is nonsingular, can be expressed as the product of $\tilde{B}^{(k-1)}$ with a unit upper triangular matrix, contradicting the singularity of $\tilde{B}^{(k-1)}$. Therefore, at least one $c_p^{(k)} \neq 0$, $k \leqq p \leqq n$.

These two theorems guarantee that algorithm (8.8) and, in particular, algorithm (8.6), with partial pivoting will find a solution to (8.1) for A nonsingular.

9. Relationship to Gaussian Elimination. Although it is not obvious, algorithm (8.6) and Gaussian elimination are related. To demonstrate this, consider the $n \times (n+1)$ matrix

$$\tilde{L} = \tilde{A}\tilde{B}^{(n)} = [A \mid -b]\begin{bmatrix} B^{(n)} & x^{(n)} \\ 0 & 1 \end{bmatrix} \tag{9.1}$$

$$= [AB^{(n)} \mid Ax^{(n)} - b].$$

By Theorem 8.1, $L = AB^{(n)}$ is lower triangular and $Ax^{(n)} - b = 0$. Since $B^{(n)}$ is unit upper triangular and $b_{kj}^{(n)} = b_{kj}^{(j-1)}$, for $1 \leqq k \leqq j-1$, the nonzero elements of $\tilde{L}$ (and $AB^{(n)}$) are given by

$$\tilde{l}_{ij} = \sum_{k=1}^{j-1} a_{ik} b_{kj}^{(j-1)} + a_{ij}, \qquad i \geqq j. \tag{9.2}$$

A comparison with (8.6a) reveals that $\tilde{l}_{kk} = c_k^{(k)}$, $k = 1, \cdots, n$.

Now, A has a unique factorization $A = L_1 D U_1$, where L_1, D and U_1 are unit lower triangular, unit upper triangular, and diagonal matrices, respectively. Since $B^{(n)-1}$ exists and is unit upper triangular and $A = AB^{(n)}B^{(n)-1}$, $U_1 = B^{(n)-1}$ and $L_1 D = AB^{(n)}$. Hence, D has diagonal elements $\tilde{l}_{kk} = c_k^{(k)}$, $k = 1, \cdots, n$. (Notice that the determinant of A can be computed as $\prod_{k=1}^{n} c_k^{(k)}$.) Also from (8.11) we have that $\tilde{B}^{(n)}\tilde{C} = \tilde{B}^{(0)}$, i.e.,

$$\begin{bmatrix} B^{(n)} & x^{(n)} \\ 0 & 1 \end{bmatrix}\begin{bmatrix} C & \alpha \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} I & b \\ 0 & 1 \end{bmatrix}.$$

Thus $U_1 = B^{(n)-1} = C$ and $\alpha = (U_1 - L^{-1})b$.

In Gaussian elimination, the original linear system is transformed on the forward pass into

$$Ux = L_1^{-1}b, \tag{9.3}$$

where $U = DU_1$, and then (9.3) is solved by back substitution. Hence $U = DC$, and the (i, j)th element of U equals $c_j^{(i)}$ for $i \leqq j$ and is zero otherwise.

An alternate method that is suggested by this analysis is the following:

Compute $B^{(n)}$ from (8.6a)–(8.6c) and $L = AB^{(n)}$ from (9.2). ($\tilde{l}_{kk} = c_k^{(k)}$ and has already been computed while forming $B^{(n)}$.) Solve the system $Ly = b$ for y, and finally, compute $x = B^{(n)}y$. This algorithm requires a total of $\frac{1}{2}n^2(n+1)$ multiplications and divisions. This is just $\frac{1}{2}n(n+1)$ less than the number required by Gauss-Jordan elimination. However, it is approximately fifty percent more than either Gaussian elimination or algorithm (8.6) require for large n.

10. A More Efficient Method. In the case of algorithm (8.6) (i.e., unit upper triangular $B^{(k)}$), Theorem 8.1 shows that formulas (8.6a) through (8.6c) produce, on the kth step, solutions to the $n-k$ sets of k equations.

$$
\begin{aligned}
&a_{11}u_1 + \cdots + a_{1k}u_k + a_{1p} = 0 \\
&\quad\vdots \qquad\qquad\quad \vdots \qquad \vdots \\
&a_{k1}u_1 + \cdots + a_{kk}u_k + a_{kp} = 0
\end{aligned}
\qquad p = k+1, \cdots, n, \tag{10.1}
$$

in k unknowns $u_1, \cdots, u_k$. If we apply this approach iteratively to the $(n+1) \times n$ matrix $\tilde{A} = (A \mid -b)$, replacing n in (10.1) by $n+1$ we arrive at the following algorithm.

Compute

$$c_j^{(k)} = \sum_{p=1}^{k-1} \tilde{a}_{kp}\tilde{b}_{pj}^{(k-1)} + \tilde{a}_{kj}, \qquad j = k, \cdots, n+1, \tag{10.2a}$$

$$\tilde{b}_{kj}^{(k)} = -c_j^{(k)}/c_k^{(k)} \qquad j = k+1, \cdots, n+1, \tag{10.2b}$$

$$\tilde{b}_{ij}^{(k)} = \tilde{b}_{ij}^{(k-1)} + \tilde{b}_{ik}^{(k-1)}\tilde{b}_{kj}^{(k)}, \qquad i = 1, \cdots, k-1, j = k+1, \cdots, n+1, \tag{10.2c}$$

for $k = 1, \cdots, n$.

The solution to (8.1) is then given by $x_i = \tilde{b}_{i,n+1}^{(n)}$, $i = 1, \cdots, n$. As in previous algorithms, partial pivoting can be employed. Operational counts total to $\frac{1}{3}n^3 + n^2 - \frac{1}{3}n$ multiplications and divisions and $\frac{1}{3}n^3 + \frac{1}{2}n^2 - \frac{5}{6}n$ additions and subtractions. These figures are the same as those for Gaussian elimination and, as we have already mentioned, are minimal for this class of methods. Equations (8.9), (8.10), and (8.11) still hold, but with $\tilde{B}^{(0)} = I$ and $c_j^{(k)}$ defined by (10.2a). After k steps,

$$\tilde{B}^{(k)} = \begin{pmatrix}
1 & \tilde{b}_{12}^{(1)} & \tilde{b}_{13}^{(2)} & \cdots & \tilde{b}_{1,k+1}^{(k)} & \cdots & \tilde{b}_{1,n+1}^{(k)} \\
 & 1 & \tilde{b}_{23}^{(2)} & \cdots & \tilde{b}_{2,k+1}^{(k)} & \cdots & \tilde{b}_{2,n+1}^{(k)} \\
 & & & & \vdots & & \vdots \\
 & & & \ddots & \tilde{b}_{k,k+1}^{(k)} & \cdots & \tilde{b}_{k,n+1}^{(k)} \\
 & & & & 1 & & \\
 & & & & & \ddots & \\
 & & & & & & 1
\end{pmatrix}, \tag{10.3}$$

and corresponding to Theorem 8.1 we have the following theorem:

THEOREM 10.1. *In algorithm* (10.2),

$$\sum_{j=1}^{k} \tilde{a}_{ij}\tilde{b}_{jp}^{(k)} + \tilde{a}_{ip} = 0, \qquad 1 \leqq i \leqq k < p \leqq n+1.$$

Theorem 8.2 remains valid. Finally, it is easy to show that $B^{(n)-1} = U_1$ and $\alpha = -L^{-1}b$ where $L = AB^{(n)}$ and the components α_j of α are equal to $c_{n+1}^{(j)}/c_j^{(j)} = -b_{j,n+1}^{(j)}$. Geometrically, the kth approximation to the solution, i.e., $\tilde{b}_{i,n+1}^{(k)}$, $i = 1, \cdots, n$, is the point defined by the intersection of the hyperplanes corresponding to $x_{k+1} = 0, \cdots, x_n = 0$ and the first k equations in (8.1).

11. **Relationship to Other Methods.** We have already discussed the relationship between algorithm (8.6) and Gaussian elimination. Since the operations of algorithm (8.6) are performed according to a compact scheme, the previous discussion is perhaps most pertinent to a comparison with the Crout form of Gaussian elimination. Algorithm (8.6) is even more closely related to other known methods. In fact, it is essentially equivalent to Purcell's vector method [29], if one initiates that procedure with a set of $n + 1$ dimensional vectors, $e_1, \cdots, e_n, v_{n+1}$, where $v_{n+1} = (b_1, \cdots, b_n, 1)'$ and e_i, $i = 1, 2, \cdots, n$, are unit coordinate vectors. This equivalence is apparent from a comparison of the iterative formulas that define the two methods.

Algorithm (8.6) is also related to Hestenes' version of his biorthogonalization method [17] for solving systems of linear equations. In Hestenes' method, one applies the biorthogonalization algorithm to the matrix

$$\left(\begin{array}{c|c} -A & b \\ \hline 0 & 1 \end{array}\right). \tag{11.1}$$

The last column of the inverse of matrix (11.1) must be $\binom{x}{1}$ where x is the solution to (8.1). Only this column is sought, and therefore, the complete biorthogonalization algorithm is not needed. Algorithm (8.6) is similar in that not all of the matrix B is updated. Hestenes suggests using the transpose of matrix (11.1) as the initial estimate of the inverse. If, however, the matrix (8.12) is used, then Hestenes' method becomes essentially equivalent to algorithm (8.6). In Hestenes' method the term to the right of the parentheses in (8.5) is first divided by the denominator and this is then multiplied by the term within the parentheses, while in algorithms (8.6) and (8.8), division is first performed on the term within the parentheses. If one chooses $B^{(0)} = A'$ in algorithm (8.8),

$$\tilde{B}^{(0)} = \left(\begin{array}{c|c} A' & A'b \\ \hline 0 & 1 \end{array}\right).$$

This is not the same as Hestenes' algorithm which starts with $\tilde{B}^{(0)}$ equal to the transpose of matrix (11.1).

Algorithm (8.6) and (10.2) are very closely related. Therefore, it is not surprising that the latter is also equivalent to special cases of Purcell's and Hestenes' methods. These equivalences are obtained if one uses the identity, i.e., the coordinate vectors $e_1, e_2, \cdots, e_{n+1}$, as the set of basic vectors in the vector method. This corresponds to Purcell's choice. It is evident from the above discussion that Hestenes' and Purcell's methods are essentially equivalent. This fact, however, seems to have gone unnoticed.

Algorithms (10.2) and (8.6) can also be shown to be compact forms of Aitken's below-the-line elimination method, i.e., a forward pass of Gaussian elimination, applied to

$$\left(\begin{array}{c|c} A' & I \\ \hline -b' & 0 \end{array}\right) \tag{11.2}$$

and

$$\left[\begin{array}{c|c} A' & I \\ \hline b'A - b' & b' \end{array}\right] \tag{11.3}$$

respectively. A similar statement can be made with regard to the cases of Purcell's method and Hestenes' method that have already been discussed.

Aitken's method transforms a matrix $\bar{A}'$, whether it be the matrix (11.2) or (11.3), to a matrix $M^{(k)\prime}\bar{A}'$, after k steps, $1 \leqq k \leqq n$. The matrices $M^{(k)\prime}\bar{A}'$ and $M^{(k-1)\prime}\bar{A}'$, at the kth and $(k-1)$st steps, are related by

$$M^{(k)\prime}\bar{A}' = (I - v^{(k)}e_k')M^{(k-1)\prime}\bar{A}', \qquad 1 \leqq k \leqq n,$$

where

$$v_j^{(k)} = \frac{(M^{(k-1)\prime}\bar{A}')_{jk}}{(M^{(k-1)\prime}\bar{A}')_{kk}} = \frac{\sum_{p=1}^{n+1} \bar{a}_{kp}M_{pj}^{(k-1)}}{\sum_{p=1}^{n+1} \bar{a}_{kp}M_{pk}^{(k-1)}}, \qquad j = k+1, \cdots, n+1, \tag{11.4}$$

and is zero otherwise. At the start of the process $M^{(0)} = I$. Thus the matrices $M^{(k)}$ and $M^{(k-1)}$ satisfy

$$M^{(k)} = M^{(k-1)}(I - e_k v^{(k)\prime}), \qquad 1 \leqq k \leqq n. \tag{11.5}$$

If we choose $\bar{A}'$ equal to matrix (11.2), then it is easily verified, taking into account the zero elements of $M^{(k-1)}$, that formulas (11.4) and (11.5) are exactly those that are used in algorithm (10.2), where $M^{(k)}$ equals the matrix $B^{(k)}$ of (10.3) and $v_j^{(k)} = c_j^{(k)}/c_k^{(k)} = -\tilde{b}_{kj}^{(k)}$ of Eq. (10.2b).

If we apply Aitken's method to matrix (11.3),

$$M^{(0)\prime}\bar{A}' = \left[\begin{array}{c|c} A' & I \\ \hline b'A - b' & b' \end{array}\right] = \left[\begin{array}{c|c} I & 0 \\ \hline b' & 1 \end{array}\right]\left[\begin{array}{c|c} A' & I \\ \hline -b' & 0 \end{array}\right],$$

and if we associate matrix (8.12) with $M^{(0)}$ then formulas (11.4) and (11.5) are those of algorithm (8.6), where $M^{(k)}$ equals the matrix $\tilde{B}^{(k)}$ of (8.7b).

Algorithms (8.6) and (10.2) are also related to Morris' escalator method [24], [25] and its equivalent, the method of orthogonal vectors [13], especially when A is symmetric. In the escalator method, the matrix $\tilde{B}^{(n)}$ is also computed. The formulas used in this computation are different from those of algorithms (8.6) and (10.2), although not as different as one might think at first. The computations performed in algorithms (8.6) and (10.2) that give rise to the first n columns of $B^{(n)}$ can be reordered as shown by Goldfarb [15], so that they are very similar to those of the escalator method.

Algorithm (10.2) and, to a lesser extent, algorithm (8.6) are also very directly related to the bordering method. The bordering method is based upon the following easily proved lemma.

LEMMA 11.1. *Let the $k \times k$ square matrix $A_{(k)}$ be partitioned as*

$$A_{(k)} = \left[\begin{array}{c|c} A_{(k-1)} & u_{k-1} \\ \hline v_{k-1}' & a_{kk} \end{array}\right]$$

where $A_{(k-1)}$ is a nonsingular square matrix of order $k-1$, a_{kk} is a scalar, and u_{k-1} and v_{k-1} are $(k-1)$-dimensional column vectors. If $A_{(k)}$ is also nonsingular, then

$a_{kk} - v'_{k-1}A^{-1}_{(k-1)}u_{k-1} \neq 0$ *and*

$$A^{-1}_{(k)} = \begin{bmatrix} A^{-1}_{(k-1)} + \dfrac{A^{-1}_{(k-1)}u_{k-1}v'_{k-1}A^{-1}_{(k-1)}}{a_{kk} - v'_{k-1}A^{-1}_{(k-1)}u_{k-1}} & -\dfrac{A^{-1}_{(k-1)}u_{k-1}}{a_{kk} - v'_{k-1}A^{-1}_{(k-1)}u_{k-1}} \\ -\dfrac{v'_{k-1}A^{-1}_{(k-1)}}{a_{kk} - v'_{k-1}A^{-1}_{(k-1)}u_{k-1}} & \dfrac{1}{a_{kk} - v'_{k-1}A^{-1}_{(k-1)}u_{k-1}} \end{bmatrix}.$$

Conversely, if $a_{kk} - v'_{k-1}A^{-1}_{(k-1)}u_{k-1} = 0$, *then* $A_{(k)}$ *is singular.*

Let $A_{(k)}$ be the leading $k \times k$ square submatrix of A and recall from Theorem 10.1 that the column vectors $a_j^{(k)} = (\tilde{a}_{1j}, \cdots, \tilde{a}_{kj})'$ and $\beta_j^{(k)} = (\tilde{b}_{1j}^{(k)}, \cdots, \tilde{b}_{kj}^{(k)})'$ satisfy $A_{(k)}\beta_j^{(k)} + a_j^{(k)} = 0$ for $j = k+1, \cdots, n+1$. Therefore if $A_{(k)}$ and $A_{(k-1)}$ are non-singular, it follows that $\beta_j^{(k)} = -A^{-1}_{(k)}a_j^{(k)}$, for $j = k+1, \cdots, n+1$; that is,

$$\beta_j^{(k)} = \begin{bmatrix} \beta_j^{(k-1)} - \beta_k^{(k-1)} \dfrac{(v'_{k-1}\beta_j^{(k-1)} + \tilde{a}_{kj})}{a_{kk} + v'_{k-1}\beta_k^{(k-1)}} \\ -\dfrac{(v'_{k-1}\beta_j^{(k-1)} + \tilde{a}_{kj})}{a_{kk} + v'_{k-1}\beta_k^{(k-1)}} \end{bmatrix}, \qquad j = k+1, \cdots, n+1,$$

since $\beta_j^{(k-1)} = -A^{-1}_{(k-1)}a_j^{(k-1)}$, $j = k, \cdots, n+1$, and $u_{k-1} = a_k^{(k-1)}$. This last equation is equivalent to algorithm (10.2) for, by Eq. (10.2a), $c_j^{(k)} = v'_{k-1}\beta_j^{(k-1)} + a_{kj}$, $j = k, \cdots, n+1$.

12. Computational Aspects. Operational counts have already been given. Based upon this criterion, the modification methods described for solving systems of linear equations compare favorably with other known methods.

They also appear attractive with regard to data handling and storage requirements. If one is willing to write over the original matrix, A, and right-hand side, b, then the methods discussed here all require the same amount of storage and are not much different from Gaussian elimination in its regular or compact forms. If A cannot be written over, then algorithms (8.6) and (10.2) require less additional storage for intermediate computations than do any of the compact forms of Gaussian elimination. The amount of additional storage locations required by algorithm (10.2) can be shown to be equal to the greatest integer less than or equal to $\frac{1}{4}n^2 + n + 2$, as was first pointed out by Purcell [29]. The same result also holds for algorithm (8.6).

Input of data is very convenient for algorithms (8.6) and (10.2) since single rows of the matrix $A = [A \mid -b]$ are required on each iteration. (Algorithm (8.6) also requires the vector b initially.) These modification methods therefore, are particularly well suited for solving large problems with storage restrictions where handling of data may be a problem. The computations of algorithm (8.4), i.e., Gauss-Jordan elimination, can also be ordered so that only single columns of A need be input on each iteration.

A full discussion of the effect of rounding errors on the modification methods described above will be presented in a subsequent paper. However, some important points shall be mentioned here. If partial pivoting is employed in the usual sense—i.e., one determines which $c_j^{(k)}$, $j = k, \cdots, n$, is of maximum modulus, and uses it as the divisor in steps (8.6b) or (10.2b) with appropriate relabelling of the columns of A and $\tilde{B}^{(k)}$—one can only ensure that all multipliers, $\tilde{b}_{kj}^{(k)} = -c_j^{(k)}/c_k^{(k)}$, $j = k+1, \cdots, n$, have absolute value less than or equal to one. This is of great importance

in obtaining error bounds on the elements of the matrices $\tilde{B}^{(k)}$, $k = 1, \cdots, n$. However, if one wants error bounds on all elements of these matrices, one must also have that $|c_{n+1}^{(k)}/c_k^{(k)}| \leqq 1$ in (8.6d) and (8.6e) and $|b_{k,n+1}^{(k)}| \leqq 1$ in (10.2b). This problem is easily resolved by extending the normal partial pivoting scheme to include $c_{n+1}^{(k)}$. The solution vector, y, that is obtained by this process, assuming that pivoting never occurs in the pth column of A, is no longer the solution to the original set of Eqs. (8.1). If algorithm (10.2) is used, y is the solution to a set of equations obtained from (8.1) by interchanging the pth column of A, A_p, with the right-hand side b and multiplying both column vectors by -1. If algorithm (8.6) is used, y solves (8.1) with A_p, replaced by $b_p A_p - b$ and the right-hand side replaced by $-A_p$. If a solution is obtained with either algorithm, then $x_i = y_i/y_p$, $i \neq p$, and $x_p = 1/y_p$, in the former case, and $x_i = y_i/y_p$, $i \neq p$, and $x_p = b_p + 1/y_p$, in the latter. If $y_p = 0$, then the original matrix A is singular.

A scheme such as the one described can also be used with complete or maximal row pivoting in Gaussian elimination. As far as the author knows, this very simple idea has not been previously suggested. Further investigation of it is needed to determine its merits.

13. Numerical Examples. Algorithms (2.6), (2.8) and (10.2) were programmed in APL/360 and tested on several problems. In APL/360 all numbers are carried to a precision of approximately 16 decimal digits, which is also the precision of internal calculations. The output exhibited below, however, has been rounded, for print-out purposes only, to either 5 or 10 decimal digits. Neither code employed a pivoting strategy.

Algorithm (2.8) was first tried on the following symmetric matrix A, also used by Faddeeva [11] for illustrative purposes.

$$A = \begin{pmatrix} 1 & 0.42 & 0.54 & 0.66 \\ 0.42 & 1 & 0.32 & 0.44 \\ 0.54 & 0.32 & 1 & 0.22 \\ 0.66 & 0.44 & 0.22 & 1 \end{pmatrix}. \tag{13.1}$$

The computer output that follows should be self-explanatory.

```
INITIAL APPROXIMATION TO A INVERSE IS :

 1 0 0 0
 0 1 0 0
 0 0 1 0
 0 0 0 1
THE NEW APPROXIMATION TO A INVERSE IS :
    1        0        0        0
 ¯0.42       1        0        0
 ¯0.54       0        1        0
 ¯0.66       0        0        1
THE RESIDUAL AB-I IS:
    0        0.42     0.54     0.66
    0       ¯0.1764  ¯0.0932  ¯0.1628
    0        0.0932  ¯0.2916  ¯0.1364
    0        0.1628  ¯0.1364  ¯0.4356
THE NEW APPROXIMATION TO A INVERSE IS :
    1.2142  ¯0.50996   0        0
 ¯0.50996    1.2142   0        0
 ¯0.49247   ¯0.11316  1        0
 ¯0.57698   ¯0.19767  0        1
```

```
THE RESIDUAL AB-I IS:
 -6.9389E-17 -2.7756E-17  4.9247E-1   5.7698E-1
 -1.3878E-17 -5.5511E-17  1.1316E-1   1.9767E-1
  1.3878E-17  1.3878E-17 -3.0215E-1  -1.5482E-1
 -1.3878E-17  0.0000E0   -1.5482E-1  -4.6778E-1
THE NEW APPROXIMATION TO A INVERSE IS :
    1.5617      -0.4301      -0.7057       0
   -0.4301       1.2325      -0.16216      0
   -0.7057      -0.16216      1.433        0
   -0.68624     -0.22277      0.22186      1
THE RESIDUAL AB-I IS:
 -9.7145E-17 -4.1633E-17  0.0000E0    6.8624E-1
 -8.3267E-17 -4.3021E-16 -4.1633E-17  2.2277E-1
 -4.1633E-17 -1.3878E-17 -3.4694E-16 -2.2186E-1
  0.0000E0    0.0000E0    1.3878E-17 -5.0213E-1
THE NEW APPROXIMATION TO A INVERSE IS :
    2.5076      -0.12304     -1.0115      -1.3783
   -0.12304      1.3322      -0.26143     -0.44745
   -1.0115      -0.26143      1.5318       0.44561
   -1.3783      -0.44745      0.44561      2.0086
THE RESIDUAL AB-I IS:
  0.0000E0    0.0000E0    2.2204E-16  2.2204E-16
 -1.3878E-16 -5.2736E-16 -6.9389E-17 -1.3878E-16
 -2.2204E-16 -2.7756E-17 -5.1348E-16  1.1102E-16
 -4.4409E-16 -1.2490E-16  1.3878E-17  0.0000E0
```

Using this same matrix (13.1) and a right-hand side equal to (0.3, 0.5, 0.7, 0.9)′ algorithm (10.2) generated the following output.

```
C
1  0.42   0.54   0.66   -0.3
    -0.42              -0.54              -0.66              0.3
     0                  0                  0                 0
     0                  0                  0                 0
     0                  0                  0                 0
C    0                  0                  0                 0
1  0.8236   0.0932   0.1628   -0.374
    -0.42              -0.4924720738      -0.5769791161      0.1092763477
     0                 -0.113161729       -0.1976687712      0.4541039339
     0                  0                  0                 0
     0                  0                  0                 0
     0                  0                  0                 0
C
1  0.8236   0.6978533269   -0.1548227295   -0.4956775134
    -0.42              -0.4924720738      -0.6862368465      -0.2405211291
     0                 -0.113161729       -0.2227743444      0.3737264072
     0                  0                  0.2218556873      0.7102889594
     0                  0                  0                 0
     0                  0                  0                 0
C
1  0.8236   0.6978533260   0.497871221   -0.738040755
    -0.42              -0.4924720738      -0.6862368465      -1.257793747
     0                 -0.113161729       -0.2227743444      0.04348730439
     0                  0                  0.2218556873      1.039166252
     0                  0                  0                 1.482392884
     0                  0                  0                 0
```

In the kth segment of output, i.e., below the kth C, the first line printed is $c^{(1)}, \cdots, c_k^{(k)}, c_{k+1}^{(k)}, \cdots, c_{n+1}^{(k)}$ followed by the last n columns of the matrix $\tilde{B}^{(k)} - I$, where $\tilde{B}^{(k)}$ is the matrix (10.3). The solution is found in the last column of the last block of output.

A more stringent test of algorithm (10.2) was provided by the ill-conditioned set of equations used by Morris to test his escalator method [24]. The matrix $\tilde{A} = [A \mid -b]$ for this system is

```
539999   523286   435785   362242   276472   184691   123679
523286   787190   362242   525651   184691   280269    48448
435785   362242   388141   297304   263974   167936   124950
362242   525651   297304   437677   167936   263246    47304
276472   184691   263974   167936   201578   114921   106470
184691   280269   167936   263246   114921   194065    37831
```

The last block of computer generated output follows.

```
C
539999  280099.7317  23581.4958  9541.117411  7654.575691  635.3333588  ¯2670.657675
¯0.9690499427  ¯1.014782143  ¯1.758366662  ¯1.486706606  ¯1.386346478  ¯5.386252422
 0             0.2144073074 ¯1.009028197  ¯0.6559707279  0.7331152348  2.813346900
 0             0            ¯1.79847381   ¯2.3017601    2.835645507  11.59232355
 0             0             0             0.7371887929 ¯1.600666275 ¯6.364825112
 0             0             0             0            1.720237317  ¯7.992872117
 0             0             0             0            0             4.20355336
 0             0             0             0            0             0
```

The vector or residuals, rounded to three significant figures, corresponding to the above computed solution was (2.33E-10, 0, 6.98E-10, 4.66E-10, 0).

The ill-conditioned 5×5 matrix A with elements

$$a_{ij} = \frac{10!}{2 \times 10^6} \frac{1}{i + j}, \qquad 1 \leqq i, j \leqq n,$$

related to the 6×6 Hilbert matrix, and used by Wilkinson [36] in a numerical example, was also used to test algorithm (2.8). The fifth and last iteration of algorithm (2.8) yielded the following approximate inverse and residual matrix.

```
THE NEW APPROXIMATION TO A INVERSE IS:
     248.015873     -2314.814815      6944.444444     -8333.333333      3472.222222
   -2314.818415      24305.55556     -77777.77778      97222.22222     -41666.66667
    6944.444444     -77777.77778      259259.2593     -333333.3333      145833.3333
   -8333.333333      97222.22222     -333333.3333      437500          -194444.4444
    3472.222222     -41666.66667      145833.3333     -194444.4444       87500
THE RESIDUAL AB-I IS:
 -8.881784197E-13 -6.146194664E-13 -3.872457910E-13  -3.232969448E-13 -1.669775429E-13
 -2.216893336E-12 -2.955857781E-12 -6.082245818E-12  -3.240074875E-12 -3.751665645E-12
  2.000888344E-11 -1.364242053E-11 -8.526512829E-13  -1.182343112E-11  8.469669410E-12
  7.275957614E-12 -1.000444172E-11 -1.875832822E-11  -5.684341886E-12  1.818989404E-12
 -1.136868377E-11 -1.733724275E-11 -9.322320693E-12  -9.492850950E-12 -6.878053682E-12
```

The APL/360 matrix division function, ⌹, was also used to invert A. The final result was identical to the output above, rounded to 10 decimal digits. The residual matrix for this computed inverse is shown below. The implementation of the operator ⌹ in APL/360 is described by Jenkins [21]. Jenkins' code makes use of Householder transformations to decompose A and incorporates pivoting and scaling procedures. Algorithm (2.8) and the operator ⌹ worked equally well on this example. The residual of maximum modulus was approximately 2.0×10^{-11} for the former method and 6.9×10^{-11} for the latter.

```
 6.288303211E-13 -1.011812856E-11  4.001776688E-11 -6.912159733E-11  3.905142876E-11
-1.527666882E-13  2.273736754E-13  1.546140993E-11 -2.000888344E-11  9.663381206E-12
-1.136868377E-13 -5.115907697E-13 -6.423306331E-12 -1.404032446E-11  7.219114195E-12
-3.552713679E-14 -5.684341886E-13  1.534772309E-12 -5.172751116E-12  6.480149750E-12
-1.350031198E-13  5.684341886E-13  1.421085472E-12  7.958078641E-12  3.922195901E-12
```

Residual Matrix

As an extreme test of algorithm (2.6), the 10×10 matrix A with elements

$$a_{ij} = \frac{20!}{10^{18}} \left(\frac{1}{i+j} \right), \qquad 1 \leqq i, j \leqq n,$$

was inverted. One full pass of the algorithm resulted in the following inverse matrix, listed columnwise.

2.4883E3	-8.9595E4	1.1649E6	-7.6116E6	2.8546E7
-8.9598E4	3.6292E6	-5.0330E7	3.4255E8	-1.3214E9
1.1650E6	-5.0332E7	7.2706E8	-5.0897E9	2.0042E10
-7.6123E6	3.4257E8	-5.0898E9	3.6372E10	-1.4549E11
2.8550E7	-1.3215E9	2.0043E10	-1.4550E11	5.8926E11
-6.5263E7	3.0837E9	-4.7511E10	3.4920E11	-1.4286E12
9.2465E7	-4.4382E9	6.9234E10	-5.1400E11	2.1202E12
-7.9262E7	3.8519E9	-6.0694E10	4.5435E11	-1.8878E12
3.7652E7	-1.8482E9	2.9365E10	-2.2136E11	9.2492E11
-7.6069E6	3.7651E8	-6.0238E9	4.5678E10	-1.9184E11

-6.5254E7	9.2449E7	-7.9247E7	3.7644E7	-7.6052E6
3.0834E9	-4.4376E9	3.8514E9	-1.8480E9	3.7645E8
-4.7508E10	6.9228E10	-6.0688E10	2.9361E10	-6.0230E9
3.4919E11	-5.1397E11	4.5432E11	-2.2134E11	4.5673E10
-1.4285E12	2.1201E12	-1.8872E12	9.2486E11	-1.9182E11
3.4920E12	-5.2188E12	4.6730E12	-2.3019E12	4.7955E11
-5.2189E12	7.8460E12	-7.0614E12	3.4939E12	-7.3074E11
4.6731E12	-7.0615E12	6.3836E12	-3.1709E12	6.6549E11
-2.3020E12	3.4940E12	-3.1709E12	1.5805E12	-3.3274E11
4.7958E11	-7.3077E11	6.6550E11	-3.3274E11	7.0245E10

The element of largest absolute value in the residual matrix, corresponding to this computed inverse, was approximately equal to 1.37×10^{-2}. This was only slightly larger than the residual of maximum modulus, 1.04×10^{-3}, obtained by using the operator $\boxed{\div}$.

A second pass of algorithm (2.6) was performed, starting with the above approximate inverse as the initial guess, $B^{(0)}$. This produced a new approximate inverse, none of whose elements differed from the corresponding elements in the above matrix by more than 0.2 percent. The residual of maximum modulus was reduced to approximately 8.2×10^{-4}.

Department of Computer Science
The City College of CUNY
139th Street & Convent Avenue
New York, New York 10031

1. A. C. AITKEN, "Studies in practical mathematics. I. The evaluation with application of a certain triple product matrix," *Proc. Roy Soc. Edinburgh Sect. A*, v. 57, 1936/37, pp. 172–181.

→ M. S. BARTLETT, "An inverse matrix adjustment arising in discriminant analysis," *Ann. Math. Statist.*, v. 22, 1951, pp. 107–111. MR **12,** 639.

3. E. BODEWIG, *Matrix Calculus*, North-Holland, Amsterdam, 1956. MR **18,** 235.

4. C. G. BROYDEN, "A class of methods for solving nonlinear simultaneous equations," *Math. Comp.*, v. 19, 1965, pp. 577–593. MR **33** #6825.

5. P. D. CROUT, "A short method for evaluating determinants and solving systems of linear equations with real or complex coefficients," *Trans. Amer. Inst. Elect. Engrs.*, v. 60, 1941, pp. 1235–1240.

6. J. E. DENNIS, JR., *On the Convergence of Broyden's Method for Nonlinear Systems of Equations*, Technical Report #69-48, Dept. of Comput. Sci., Cornell University, Ithaca, N.Y., 1969.

7. M. H. DOOLITTLE, *Method Employed in the Solution of Normal Equations and the Adjustment of Triangulation*, U.S. Coast Guard and Geodetic Survey Report, 1878, pp. 115–120.

8. W. J. DUNCAN, "Some devices for the solution of large sets of simultaneous linear equations," *Philos. Mag.*, (7), v. 35, 1944, pp. 660–670. MR **7,** 84.

9. P. S. DWYER, *Linear Computations*, Wiley, New York, 1951. MR **13,** 283.

10. A. P. ERŠOV(ERSHOV), "On a method for inverting matrices," *Dokl. Akad. Nauk SSSR*, v. 100, 1955, pp. 209–211. (Russian) MR **16,** 1082.

11. D. K. FADDEEV & V. N. FADDEEVA, *Computational Methods of Linear Algebra,* Fizmatgiz, Moscow, 1960; English transl., Freeman, San Francisco, Calif., 1963. MR **28** #1742.

12. L. FOX, *An Introduction to Numerical Linear Algebra*, Monographs on Numerical Analysis, Clarendon Press, Oxford, 1964. MR **29** #1733.

13. L. FOX, H. D. HUSKEY & J. H. WILKINSON, "Notes on the solution of algebraic linear simultaneous equations," *Quart. J. Mech. Appl. Math.*, v. 1, 1948, pp. 149–173. MR **10,** 152.

14. C. F. GAUSS, "Supplementum theoriae combinationis observationum erroribus minimis obnoxiae," *Werke*. Band IV, Gottingen, 1873, pp. 55–93.

15. D. GOLDFARB, *Modification Methods for Inverting Matrices and Solving Systems of Linear Algebraic Equations*, IBM-Philadelphia Scientific Center, Technical Report #320-2998, 1971.

16. J. GREENSTADT, "Variations on variable-metric methods (with discussion)," *Math. Comp.*, v. 24, 1970, pp. 1–22. MR **41** #2895.

17. M. R. HESTENES, "Iterative computational methods," *Comm. Pure Appl. Math.*, v. 8, 1955, pp. 85–95. MR **16,** 863.

→ M. R. HESTENES, "Inversion of matrices by biorthogonalization and related results," *J. Soc. Indust. Appl. Math.*, v. 6, 1958, pp. 51–90. MR **19,** 1080.

→ H. HOTELLING, "Some new methods in matrix calculation," *Ann. Math. Statist.*, v. 14, 1943, pp. 1–34. MR **4,** 202.

20. A. S. HOUSEHOLDER, *The Theory of Matrices in Numerical Analysis*, Blaisdell, Waltham, Mass., 1964. MR **30** #5475.

21. M. A. JENKINS, *The Solution of Linear Systems of Equations and Linear Least Squares Problems in APL*, IBM-New York Scientific Center, Technical Report #320-2989, 1970.

22. V. V. KLJUEV & N. I. KOKOVKIN-ŠČERBAK, "On the minimization of the number of arithmetic operations for solving linear algebraic systems of equations," *Ž. Vyčisl. Mat. i Mat. Fiz.*, v. 5, 1965, pp. 22–33 = *U.S.S.R. Comput. Math. and Math. Phys.*, v. 5, 1965, pp. 25–43. MR **30** #2672.

23. G. KRON, *Diakoptics,* Macdonald, London, 1963.

24. J. MORRIS, "An escalator process for the solution of linear simultaneous equations," *Philos. Mag.*, (7), v. 37, 1946, pp. 106–120. MR **8,** 287.

25. J. MORRIS, *The Escalator Method in Engineering Vibration Problems*, Wiley, New York, 1947. MR **9,** 382.

26. T. PIETRZYKOWSKI, *Projection Method*, Zakladu Aparatów Matematycznych Polskiej Akad. Nauk. Praca A8 (as reported in Householder [20]).

27. M. J. D. POWELL, "A theorem on rank one modifications to a matrix and its inverse," *Comput. J.*, v. 12, 1969/70, pp. 288–290. MR **39** #6502.

28. M. J. D. POWELL, "A new algorithm for unconstrained optimization," in *Nonlinear Programming*, J. B. Rosen, O. L. Mangasarian & K. Ritter (Editors), Academic Press, New York, 1970, pp. 31–65.

29. E. W. PURCELL, "The vector method of solving simultaneous linear equations," *J. Mathematical Phys.*, v. 32, 1953, pp. 180–183. MR **15,** 471.

30. J. SHERMAN, *Computations Relating to Inverse Matrices. Simultaneous Linear Equations and the Determination of Eigenvalues*, Nat. Bur. Standards Appl. Math. Ser., no. 29, U. S. Government Printing Office, Washington, D.C., 1953, pp. 123–124. MR **15,** 164.

31. J. SHERMAN & W. J. MORRISON, "Adjustment of an inverse matrix corresponding to changes in a given column or row of the original matrix," *Ann. Math. Statist.*, v. 20, 1949, p. 621.

→ J. SHERMAN & W. J. MORRISON, "Adjustment of an inverse matrix corresponding to a change in one element of a given matrix," *Ann. Math. Statist.*, v. 21, 1950, pp. 124–127. MR **11,** 693.

33. V. STRASSEN, "Gaussian elimination is not optimal," *Numer. Math.*, v. 13, 1969, pp. 354–356. MR **40** #2223.

34. H. S. WILF, "Matrix inversion by the annihilation of rank," *J. Soc. Indust. Appl. Math.*, v. 7, 1959, pp. 149–151. MR **21** #6689.

35. H. S. WILF, "Matrix inversion by the method of rank annihilation," in *Mathematical Methods for Digital Computers*, Wiley, New York, 1960, pp. 73–77. MR **22** #8685.

36. J. H. WILKINSON, *Rounding Errors in Algebraic Processes*, Prentice-Hall, Englewood Cliffs, N.J., 1963. MR **28** #4661.

37. M. A. WOODBURY, *Inverting Modified Matrices*, Statistical Research Group Memo. Report, no. 42, Princeton University, Princeton, N.J., 1950. MR **12,** 361; see also Householder [20].

38. G. ZIELKE, "Inversion of modified symmetric matrices," *J. Assoc. Comput. Mach.*, v. 15, 1968, pp. 402–408.