X-643-66-533

NASA TM X-55757

COMPUTING THE PSEUDO-INVERSE

BY

CHRISTOPHER R. HERRON

NOVEMBER 1966

GODDARD SPACE FLIGHT CENTER

GREENBELT, MARYLAND

X-643-66-533

COMPUTING THE PSEUDO-INVERSE

By

Christopher R. Herron

November 1966

GODDARD SPACE FLIGHT CENTER
Greenbelt, Maryland

COMPUTING THE PSEUDO-INVERSE

By
Christopher R. Herron

ABSTRACT

An orthogonalization algorithm for producing the pseudo-inverse of a matrix is described, and a FORTRAN program which realizes the algorithm is given in detail.

ACKNOWLEDGMENT

E. R. Lancaster, under whose supervision this paper was written, was particularly helpful in the development of certain theoretical aspects and supplied many perceptive suggestions on overall organization. G. H. Wyatt's programming skill was instrumental in the debugging phase of the programming effort.

To every matrix A there corresponds a unique matrix A^+ with the following properties:

$$AA^+A = A \tag{1}$$

$$A^+AA^+ = A^+ \tag{2}$$

$$(A^+A)^T = A^+A \tag{3}$$

$$(AA^+)^T = AA^+ \tag{4}$$

Penrose [1], one of the originators of this concept, called A^+ the generalized inverse of A, and equations (1) through (4) are often called Penrose's Lemmas. Recent usage applies <u>generalized inverse</u> to any matrix satisfying (1), (1) and (2), or (1), (2), and (3), referring to the unique A^+ as the pseudo-inverse of A. Other definitions of A^+ have been given (e.g. Albert [2], Ben-Israel [3]) but the most common is that given above.

For simplicity's sake, the rest of this paper considers only real matrices, although most results hold for complex matrices as well. The pseudo-inverse provides a way to handle the ubiquitous matrix-vector equation

$$Ax = y . \tag{5}$$

If A is square and non-singular, A^+ is A^{-1} and the vector A^+y solves the equation. The particular advantage of the pseudo-inverse appears when A is singular or non-square, since A^+y then is the minimal vector for this equation; that is, if M is the set of all vectors x_0 such that

$$\|Ax_0 - y\| \leq \|Ax - y\| \tag{6}$$

for all x, then $A^+y \in M$ and

$$\|A^+y\| = \min_{x_0 \in M} \left\{\|Ax_0\|\right\} . \tag{7}$$

Here we use the standard Euclidean norm.

A theorem which dates back to the time of Gauss (Newhouse [4]) states, in effect, that if $x_0 \in M$, then x_0 is a solution of

$$A^T A x = A^T y .$$

This type of system, often called a set of normal equations, is found repeatedly in least squares problems. (See, e.g., Rao [5]). Since $A^+ y \in M$, the application of A^+ in these circumstances is evident.

The same theorem also states that if $x_0 \in M$, then x_0 is a projection of y onto the column space of A. Newhouse later gives a theorem which proves condition (7), that $A^+ y$ is the "shortest" of these projections, giving rise to Greville's assertion [6] that $A^+ y$ is the best solution to equation (5) in the least squares sense.

Naturally, the theoretical existence of such a useful mathematical object makes a method for its computation very desirable. Most of the methods suggested, however, require that the product $A^T A$ be formed and that Gaussian elimination (or one of its variants such as pivotal condensation or sweep out) be performed on it. Should we be faced with an ill-conditioned matrix, it is entirely possible that numerical difficulties will prevent any significant computation using such methods. For example, consider the matrix

$$H = \begin{bmatrix} \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \\ \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & \frac{1}{10} \\ \frac{1}{8} & \frac{1}{9} & \frac{1}{10} & \frac{1}{11} \end{bmatrix}$$

The Hilbert matrix is notoriously ill-conditioned with respect to Gaussian elimination. The upper left-hand 4×4 corner of it has a condition number $\lambda_{max}/\lambda_{min}$ given by Marcus (Ref. [7]) as 15,514, so our 4×4 segment of it would certainly be suspect. Fox (Ref. [8]) shows that our suspicions are justified, giving the Gauss elimination process for H, in which the steady decrease in magnitude of the pivots leads to very unreliable quantities. More to our point, he demonstrates that Gauss elimination fails completely when applied to $H^T H$. We should realize that a bad but workable problem can become pathologically unmanageable if such a product is formed, and, as a general rule, avoid such approaches.

The method of Rust, Burrus, and Schneeberger (Ref. [9]) was used to compute the pseudo-inverse because it does conform to this general rule. Briefly, it can be characterized as follows: if the $m \times n$ matrix A is in the form $[R|S]$, where the k linearly independent columns form the submatrix R and the linearly dependent columns form the submatrix S, make up the $n \times n$ identity matrix and write, symbolically,

$$\left[\begin{array}{c|c} R & S \\ \hline I_k & 0 \\ 0 & I_{n-k} \end{array}\right]$$

Then perform the Gram-Schmidt (G.S.) orthogonalization process on $[R|S]$, and apply these elementary column operations to the lower submatrix to get

$$\left[\begin{array}{c|c} Q & 0 \\ \hline Z & -U \\ 0 & I_{n-k} \end{array}\right]$$

Next, perform the G.S. process on the submatrix $\begin{bmatrix} -U \\ I_{n-k} \end{bmatrix}$ to produce

$$\left[\begin{array}{c|c} Q & 0 \\ \hline Z & -UP \\ 0 & P \end{array}\right] ;$$

form the matrix

$$\left[\begin{array}{c} Q^T \\ \hline (UP)^T Z Q^T \end{array}\right]$$

and, finally,

$$A^+ = [R|S]^+ = \begin{bmatrix} Z & -UP \\ 0 & P \end{bmatrix} \begin{bmatrix} Q^T \\ (UP)^T Z Q^T \end{bmatrix}$$

A complete derivation is given in Ref. [9] and a few auxiliary notes are given in Appendix A.

Of course, not every matrix will be in the convenient $[R|S]$ form, but if we can determine which columns of A are dependent we can certainly permute columns to produce it; then $[R|S]^+$ is found and by the authority of Theorems I and II, Appendix A, the rows of $[R|S]^+$ are likewise permuted to get A^+. Since the G.S. process not only orthogonalizes the independent columns of A but also makes the dependent ones zero, we can use it to find the dependent columns.

Now we have a straightforward way to proceed:

(1) Use G.S. to find the dependent columns.

(2) Permute to get $[R|S]$.

(3) Use G.S. to find $[R|S]^+$.

(4) Permute to get A^+.

The reader will have noticed that the G.S. process is used in step (1) and again in step (3). We could save some computation time if we combined the two steps and performed the G.S. process only once. A closer examination of the process reveals that we can, under certain conditions, make this combination.

Our program uses a modified Gram-Schmidt process which is more accurate than the classic textbook version. A recursive algorithm describing our version is:

(1) Orthogonalize c_j, the next column of A:

$$b_j = c_j - \sum_{i=1}^{j-1} \left(\frac{c_j \cdot b_i'}{b_i' \cdot b_i'} \right) b_i'$$

(2) Is $b_j \approx 0$? If so, zero it out and go to step (1). If not, do step (3).

(3) Re-orthogonalize b_j:

$$b_j' = b_j - \sum_{i=1}^{j-1} \left(\frac{b_j \cdot b_i'}{b_i' \cdot b_i'} \right) b_i' ,$$

and go to step (1).

The initial condition is $b_1' = c_1$. After we run out of columns, we normalize each one and we have an orthonormal matrix $A_\perp$.

If we want to duplicate these elementary column operations on another matrix D, we could save the numbers

$$\left(\frac{c_j \cdot b_i'}{b_i' \cdot b_i'} \right) , \qquad \left(\frac{b_j \cdot b_i'}{b_i' \cdot b_i'} \right) , \qquad \text{and} \qquad \left(b_j' \cdot b_j' \right)^{1/2}$$

and then go through the algorithm again, this time letting c_j be the columns of D. More precisely, we might save these numbers in an $n \times n$ matrix S, defined as

$$S_{ji} = \frac{c_j \cdot b_i'}{b_i' \cdot b_i'} \qquad (j > i) ,$$

$$S_{ij} = \frac{b_j \cdot b_i'}{b_i' \cdot b_i'} \qquad (j > i) ,$$

$$S_{jj} = \left(b_j' \cdot b_j' \right)^{1/2} \qquad (1 \le j \le n) .$$

As an example, let A be

$$\begin{bmatrix} 1 & 1 & 3 & 6 \\ 2 & 2 & 6 & 7 \\ 3 & 3 & 9 & 8 \end{bmatrix}$$

Using eight-digit arithmetic and rounding the final answers to three digits, we have

$$A_\perp = \begin{bmatrix} .267 & 0 & 0 & .873 \\ .535 & 0 & 0 & .218 \\ .802 & 0 & 0 & -.436 \end{bmatrix}$$

$$S = \begin{bmatrix} 3.74 & 0 & 0 & .213 \times 10^{-7} \\ 1.00 & 0 & 0 & 0 \\ 3.00 & 0 & 0 & 0 \\ 3.14 & 0 & 0 & 3.27 \end{bmatrix}$$

Once we have done the G.S. process on A, we have done it for all column permutations of A which do not disturb the relative order of the independent columns. If P is a permutation matrix such that $AP = [R|S]$, where R is the matrix of independent columns of A in their original relative order, the orthonormal matrix $[Q|0]$ produced by the G.S. process on $[R|S]$ will be AP. In our example, suppose

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

Then

$$[R|S] = AP = \begin{bmatrix} 1 & 6 & 1 & 3 \\ 2 & 7 & 2 & 6 \\ 3 & 8 & 3 & 9 \end{bmatrix},$$

and

$$[Q|0] = A_\perp P = \begin{bmatrix} .267 & .873 & 0 & 0 \\ .535 & .218 & 0 & 0 \\ .802 & -.436 & 0 & 0 \end{bmatrix}.$$

We can also produce a new S matrix (F) by permutations. Referring back to the definition of S, one can see that a particular column c_j has its initial orthogonalization coefficients $(c_j \cdot b_i')/(b_i' \cdot b_i')$ on the jth row and below the diagonal, and its secondary coefficients $(b_j \cdot b_i')/(b_i' \cdot b_i')$ fall on the jth column and above the diagonal. Once c_j is converted into b_j', all initial coefficients having b_j' as a factor fall on the jth column below the diagonal, and all such secondary coefficients fall on the jth row above the diagonal. When b_j' is normalized, its "length" falls on the jth diagonal element. Moving c_j to a new position therefore means that we must move the jth row and column of S to corresponding positions, or

$$F = P^T S P \quad .$$

In our example,

$$F = \begin{bmatrix} 3.74 & .213 \times 10^{-7} & 0 & 0 \\ 3.14 & 3.27 & 0 & 0 \\ 1.00 & 0 & 0 & 0 \\ 3.00 & 0 & 0 & 0 \end{bmatrix}$$

If we go through the algorithm with c_j taken as the columns of a matrix D and the numbers $(c_j \cdot b_i')/(b_i' \cdot b_i')$ and $(b_j \cdot b_i')/(b_i' \cdot b_i')$ taken as f_{ji} and f_{ij} respectively, we have applied the elementary column operations of the G.S. process on $[R|S]$ to D.

Now we have the desired result: once the G.S. process on A is complete, it is not necessary to do it again on $[R|S]$ to derive its effects; merely execute the indicated permutations on $A_\perp$ and S and we have all the necessary matrices. Using this result, the procedure (1) through (4) on page 4 can be rewritten:

(1) Use G.S. on A; save the G.S. coefficients in S and save $A_\perp$. Note which columns are dependent.

(2) Permute $A_\perp$ to get $[Q|0]$; permute S to get F.

(3) Use the entries of F to operate on

$$\begin{bmatrix} I_k & 0 \\ 0 & I_{n-k} \end{bmatrix} ,$$

producing

$$\begin{bmatrix} Z & -U \\ 0 & I_{n-k} \end{bmatrix}$$

(4) Proceed as usual to find $[R|S]^+$.

(5) Permute to get A^+.

The program whose flow chart and FORTRAN listing appear in Appendices B and C has been checked with a variety of matrices on the IBM 7094 and appears to run properly. Two particular cautions might be extended, however: first, one will note that a decision on the dependency of any column is made by comparing the "length" of the generated orthogonal column with the "length" of the original column. If the check number $(b_j \cdot b_j)/(c_j \cdot c_j)$ is smaller than a certain tolerance, the column b_j is made zero. When the check number is very close to the tolerance, any decision made will not be a good one and the resulting perturbations can become serious; for example, the Hilbert matrix gives poor results for this very reason. One might vary the tolerance to suit special cases.

Second, although this program finds the inverse if it exists, there are routines in general use which get better inverses. For example, the SHARE routine MATINV was tested against this program on a sequence of Pei matrices (Ref. [10], Ref. [11]) and consistently got one more accurate digit in the worst cases. The difference is not great but the prospective user should realize that it exists.

Finally, an experienced programmer will see that the FORTRAN realization in Appendix C is not in optimal form. A more streamlined, double-precision version is being prepared for the IBM 360 as of this writing. The author would appreciate hearing of mistakes in, or improvements upon, the original.

APPENDIX A

(Supplementary notes for Ref. [9])

Theorem I

If P is a permutation matrix (possibly a product of elementary permutation matrices) and A^+ is the pseudo-inverse of A, then

$$(AP)^+ = P^T A^+ .$$

Proof: We need only verify that Penrose's Lemmas hold. Noting that $PP^T = P^T P = I$, we have

(a) $(AP)\left(P^T A^+\right)(AP) = AP$

(b) $P^T A^+ (AP) P^T A^+ = P^T A^+$

(c) $\left[(AP)\left(P^T A^+\right)\right]^T = \left(AA^+\right)^T = AA^+ = (AP)\left(P^T A^+\right)$

(d) $\left[\left(P^T A^+\right)(AP)\right]^T = \left[P^T (A^+ A) P\right]^T$

$= P^T (A^+ A)^T P = P^T (A^+ A) P$

$= \left(P^T A^+\right)(AP) .$

Theorem II

If P is a permutation matrix and the operation AP effects a column permutation of A, then $P^T A$ effects that same permutation on the rows of A.

Proof: Suppose one of the effects of AP is to change column i to the j th place. Then $P_{ij} = 1$, $P^T_{ji} = 1$, and $P^T A$ changes row i to the j th place.

We use this result to get A^+ from a row permutation of $[R|S]^+$— that same permutation of columns which transformed A into $[R|S]$.

The paper states (p. 383, right column) that the G.S. process turns a dependent vector into the zero vector. One might check this statement by referring to Hoffmann and Kunze, p. 230, Theorem III (Ref. [12]). If a_{k+1} is a linear combination of $a_1, \cdots, a_k$ then it is a linear combination of $q_1, \cdots, q_k$ since the vectors q_i span the space of the vectors a_i. Furthermore, by the above-mentioned theorem,

$$a_{k+1} = \sum_{i=1}^{k} \left(a_{k+1}^H q_i\right) q_i$$

and $c_{k+1} = 0$.

On p. 384, left column we are to note that I_{n-k} remains unchanged. Suppose we are operating on column $k+p$ $(p > 0)$ of the matrix $[R|S]$. We have

$$c_{k+p} = a_{k+p} - \sum_{i=1}^{k+p-1} \left(a_{k+p}^H q_i\right) q_i$$

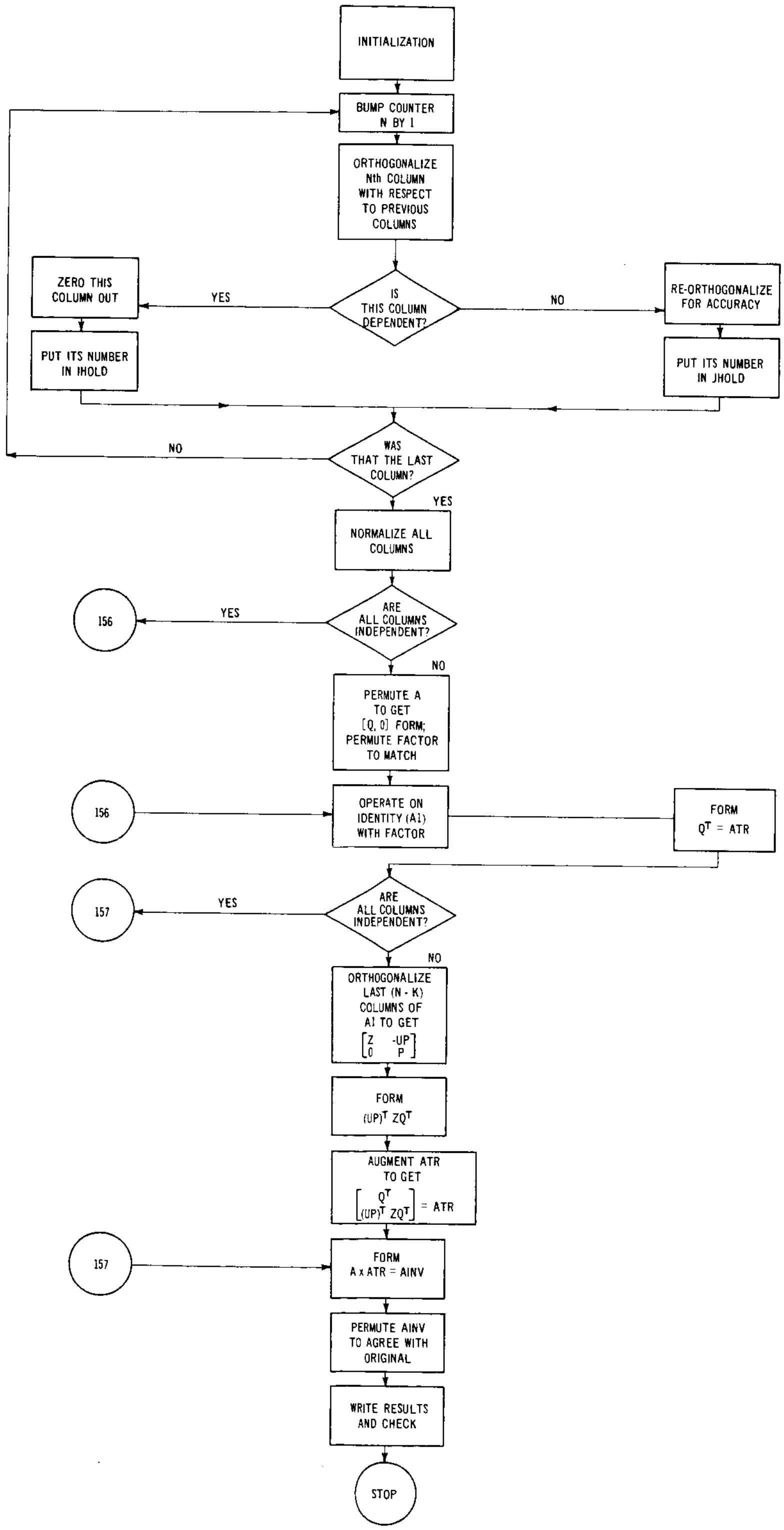
$$= a_{k+p} - \sum_{i=1}^{k} \left(a_{k+p}^H q_i\right) q_i - \sum_{i=k+1}^{k+p-1} \left(a_{k+p}^H q_i\right) q_i \cdot$$

But each q_i, $k+1 \le i \le k+p-1$, has been zeroed out already, since they came from vectors dependent upon $a_1, \cdots, a_k$, so the above is

$$c_{k+p} = a_{k+p} - \sum_{i=1}^{k} \left(a_{k+p}^H q_i\right) q_i \quad .$$

Similar column operations on the identity matrix then use only the first k columns, whose lower $n-k$ entries are all zero and cannot contribute to any modification of I_{n-k}.

APPENDIX B


INITIALIZATION
BUMP COUNTER N BY 1
ORTHOGONALIZE Nth COLUMN WITH RESPECT TO PREVIOUS COLUMNS
IS THIS COLUMN DEPENDENT?
YES
ZERO THIS COLUMN OUT
PUT ITS NUMBER IN IHOLD
NO
RE-ORTHOGONALIZE FOR ACCURACY
PUT ITS NUMBER IN JHOLD
WAS THAT THE LAST COLUMN?
NO
YES
NORMALIZE ALL COLUMNS
ARE ALL COLUMNS INDEPENDENT?
YES
156
NO
PERMUTE A TO GET [Q, 0] FORM; PERMUTE FACTOR TO MATCH
156
OPERATE ON IDENTITY (AI) WITH FACTOR
FORM Q^T = ATR
ARE ALL COLUMNS INDEPENDENT?
YES
157
NO
ORTHOGONALIZE LAST (N - K) COLUMNS OF AI TO GET $\begin{bmatrix} Z & -UP \\ 0 & P \end{bmatrix}$
FORM $(UP)^T ZQ^T$
AUGMENT ATR TO GET $\begin{bmatrix} Q^T \\ (UP)^T ZQ^T \end{bmatrix}$ = ATR
157
FORM A x ATR = AINV
PERMUTE AINV TO AGREE WITH ORIGINAL
WRITE RESULTS AND CHECK
STOP

APPENDIX C

```
      DIMENSION A(10,10),FACTOR(10,10),S2(10,10),IHOLD(10),JHOLD(10),
     1A1(10,10),ORIG(10,10),PROD(10,10),PROD1(10,10),
     2PROD2(10,10),ATR(10,10),AINV(10,10)
      DIMENSION UPTR(10,10)
      DIMENSION AINV1(10,10)
      DATA PRAIN1/5HAINV1/
      DATA PRA,PRFACT,PRS2/1HA,4HFACT,2HS2/
      DATA PRA1/2HA1/
      DATA PRAINV/4HAINV/
      DATA PRORIG,PRPR,PRPR1,PRPS2,PRATR/4HORIG,4HPROD,5HPROD1,5HPROD2,
     13HATR/
      DATA PRUPTR/4HUPTR/
154   READ(5,1) NROWS,NCOLS
1     FORMAT(2I5)
       DO 112 I = 1,NROWS
112   READ(5,2) (A(I,J),J = 1,NCOLS)
2     FORMAT(6E12.8)
      TOL = (10.*0.5**27)**2
      DO 110 I = 1,NROWS
      DO 110 J = 1,NCOLS
110   ORIG(I,J) = A(I,J)
      DO 100 I=1,NCOLS
      IHOLD(I) = 0
      JHOLD(I) = 0
      DO 102 J = 1,NCOLS
      S2(I,J) = 0.
      PROD1(I,J) = 0.
      PROD2(I,J) = 0.
      ATR(I,J) = 0.
      FACTOR(I,J) = 0.
102   A1(I,J) = 0.
100   A1(I,I) = 1.
      JHOLD(1) = 1
      KK = 1
      JJ = 0
      II = 0
      N = 1
152   NLESS1 = N
      N = N + 1
      CHECK = DOT(A,N,N,NROWS)
      DO 101 I = 1,NLESS1
      FACTOR(N,I) = DOT(A,N,I,NROWS)/DOT(A,I,I,NROWS)
      DO 101 J = 1, NROWS
101   A(J,N) = A(J,N) - FACTOR(N,I)*A(J,I)
      CHECK = DOT(A,N,N,NROWS)/CHECK
      IF(CHECK - TOL) 150,150,151
150   DO 103 J = 1,NROWS
```

```
103   A(J,N) = 0.
      JJ = JJ + 1
      IHOLD(JJ) = N
      GO TO 155
151   DO 104 I = 1,NLESS1
      FACTOR(I,N) = DOT(A,N,I,NROWS)/DOT(A,I,I,NROWS)
      DO 104 J = 1,NROWS
104   A(J,N) = A(J,N) - FACTOR(I,N)*A(J,I)
      KK = KK + 1
      JHOLD(KK) = N
155   IF(N - NCOLS) 152,153,153
153   DO 105 J = 1,NCOLS
      FACTOR(J,J) = SQRT(DOT(A,J,J,NROWS))
      IF(FACTOR(J,J) .EQ. 0.) GO TO 105

      DO 106 K = 1,NROWS
106   A(K,J) = A(K,J)/FACTOR(J,J)
105   CONTINUE
      CALL WRITE(FACTOR,NCOLS,NCOLS,PRFACT)
      CALL WRITE(A,NROWS,NCOLS,PRA)
      IF(KK.EQ.NCOLS) GO TO 156
      DO 120 I = 1,KK
      ISUB = JHOLD(I)
      DO 120 J = 1,NCOLS
      A(J,I) = A(J,ISUB)
120   S2(J,I) = FACTOR(J,ISUB)
      DO 121 I = 1,KK
      ISUB = JHOLD(I)
      DO 121 J = 1,NCOLS
121   FACTOR(I,J) = S2(ISUB,J)
      KK = KK + 1
      DO 125 I = KK,NCOLS
      II = II + 1
      ISUB2 = IHOLD(II)
      DO 125 J = 1,NCOLS
125   FACTOR(I,J) = S2(ISUB2,J)
      DO 162 I = KK,NCOLS
      DO 162 J = 1,NROWS
162   A(J,I) = 0.
      KK = KK - 1
156   CALL WRITE(A,NROWS,NCOLS,PRA)
      CALL WRITE(FACTOR,NCOLS,NCOLS,PRFACT)
      CALL WRITE(A1,NCOLS,NCOLS,PRA1)
      DO 502 I = 2,NCOLS
      ILESS1 = I -1
      DO 500 J = 1,ILESS1
```

PRECEDING PAGE BLANK NOT FILMED.

```
      DO 500 K = 1,NROWS
500   A1(K,I) = A1(K,I) - FACTOR(I,J) * A1(K,J)
      DO 502 J = 1,ILESS1
      DO 502 K = 1,NROWS
502   A1(K,I) = A1(K,I) - FACTOR(J,I) * A1(K,J)
      DO 501 I = 1,KK
      DO 501 J = 1,NROWS
501   A1(J,I) = A1(J,I)/FACTOR(I,I)
      CALL WRITE(A1,NCOLS,NCOLS,PRA1)
      CALL TRANSP(A,NROWS,KK,ATR)
      CALL WRITE(ATR,NCOLS,NROWS, PRATR)
      IF(KK.EQ.NCOLS) GO TO 157
      IST = KK + 1
      IF(KK.EQ.(NCOLS-1)) GO TO 158
      N = KK + 1
159   NLESS1 = N
      N = N +1
      DO 107 I = IST,NLESS1
      FACTOR(N,I) = DOT(A1,N,I,NCOLS)/DOT(A1,I,I,NCOLS)
      DO 107 J = 1,NCOLS
107   A1(J,N) = A1(J,N) - FACTOR(N,I) * A1(J,I)
      DO 108 I = IST,NLESS1
      FACTOR(I,N) = DOT(A1,N,I,NCOLS)/DOT(A1,I,I,NCOLS)
      DO 108 J = 1,NCOLS
108   A1(J,N) = A1(J,N) - FACTOR(I,N) * A1(J,I)
      IF(N.LT.NCOLS) GO TO 159
      CALL WRITE(A1,NCOLS,NCOLS,PRA1)
158   DO 128 I = IST,NCOLS
      FACTOR(I,I) = SQRT(DOT(A1,I,I,NCOLS))
      DO 128 J = 1,NCOLS


128   A1(J,I) = A1(J,I)/FACTOR(I,I)
      CALL WRITE(A1,NCOLS,NCOLS,PRA1)
      CALL MATMPY(A1,KK,KK,ATR,NROWS,PROD1)
      CALL WRITE(PROD1,KK,NROWS,PRPR1)
      LIM = NCOLS - KK
      DO 122 I = 1,LIM
      ISUB1 = KK + I
      DO 122 J = 1,NCOLS
122   UPTR(I,J) = -A1(J,ISUB1)
      CALL WRITE(UPTR,LIM,NCOLS,PRUPTR)
      CALL MATMPY(UPTR,LIM,KK,PROD1,NROWS,PROD2)
      CALL WRITE(PROD2,LIM,NROWS,PRPR2)
      DO 123 I = 1,LIM
      ISUB3 = KK + I
      DO 123 J = 1,NROWS
```

```
123   ATR(ISUB3,J) = PROD2(I,J)
      CALL WRITE(ATR,NCOLS,NROWS, PRATR)
157   CALL MATMPY(A1,NCOLS,NCOLS,ATR,NROWS,AINV)
      CALL WRITE(AINV,NCOLS,NROWS,PRAINV)
      CALL WRITE(ORIG,NROWS,NCOLS,PRORIG)
      DO 126 I = 1,KK
      ISUB4 = JHOLD(I)
      DO 126 J = 1,NROWS
126   AINV1(ISUB4,J) = AINV(I,J)
      IF (KK .EQ.NCOLS) GO TO 160
      KK = KK + 1
      II = 0
      DO 127 I = KK ,NCOLS
      II = II + 1
      ISUB5 = IHOLD(II)
      DO 127 J = 1,NROWS
127   AINV1(ISUB5,J) = AINV(I,J)
      KK = KK - 1
160   CALL WRITE(AINV1,NCOLS,NROWS,PRAIN1)
      CALL MATMPY(ORIG,NROWS,NCOLS,AINV1,NROWS,PROD)
      CALL WRITE(PROD,NROWS,NROWS,PRPR)
      CALL MATMPY(PROD,NROWS,NROWS,ORIG,NCOLS,PROD2)
      CALL WRITE(PROD2,NROWS,NCOLS,PRPR2)
      CALL MATMPY(AINV1,NCOLS,NROWS,PROD,NROWS,PROD2)
      CALL WRITE(PROD2,NCOLS,NROWS,PRPR2)
      CALL MATMPY(AINV1,NCOLS,NROWS,ORIG,NCOLS,PROD)
      CALL WRITE(PROD,NCOLS,NCOLS,PRPR)
      GO TO 154
      END

      FUNCTION DOT (A,J,K,NROWS)
      DIMENSION A(10,10)
      DOT = 0.
      DO 10 I = 1,NROWS
10    DOT = DOT + A(I,J) * A(I,K)
      RETURN
      END

      SUBROUTINE WRITE(X,NROWS,NCOLS,NAME)
      DIMENSION X(10,10)
      WRITE(6,3) NAME
3     FORMAT(1H0////7H MATRIX,1A7)
      DO 21 I = 1,NROWS
21    WRITE(6,4) (X(I,J),J = 1,NCOLS)
```

```
4       FORMAT(1H0,8E16.8)
        RETURN
        END

        SUBROUTINE MATMPY(A,NRA,NCA,B,NCB,PROD)
        DIMENSION A(10,10),B(10,10),PROD(10,10)
        DO 600 I = 1,NRA
        DO 600 J = 1,NCB
        PROD(I,J) = 0.
        DO 600 K = 1,NCA
600     PROD(I,J) = PROD(I,J) + A(I,K) * B(K,J)
        RETURN
        END

        SUBROUTINE TRANSP(A,NRA,NCA,ATR)
        DIMENSION A(10,10),ATR(10,10)
        DO 601 I = 1,NRA
        DO 601 J = 1,NCA
601     ATR(J,I) = A(I,J)
        RETURN
        END
```

BIBLIOGRAPHY

1. Penrose, R.: A generalized inverse for matrices, Proc. of the Cambridge Philosophical Society, Vol. 51, 1955, pp. 406-413.

2. Albert, A.: An Introduction and Beginner's Guide to Matrix Pseudo-Inverses, ARCON, 803 Mass. Ave., Lexington 73, Mass., 1964, particularly pp. III-8 to III-16.

3. Ben-Israel, A.: An iterative method for computing the generalized inverse of an arbitrary matrix, Math. of Comp., Vol. 19, 1965, pp. 452-455.

4. Newhouse, S.: Introduction to Matrix Generalized Inverses and Their Applications, GSFC X-643-66-34, 1966.

5. Rao, C. R.: A note on a generalized inverse of a matrix with applications to problems in mathematical statistics, J. of the Royal Stat. Soc., Series B. (Methodological), Vol. 24, No. 1, 1962, pp. 152-158.

6. Greville, T.: The pseudoinverse of a rectangular or singular matrix and its application to the solution of systems of linear equations, SIAM Review, Vol. 1, No. 1, 1959, pp. 38-43.

7. Marcus, M.: Basic Theorems in Matrix Theory, National Bureau of Standards, Applied Mathematics Series, No. 57, p. 23.

8. Fox, L.: An Introduction to Numerical Linear Algebra, Clarendon Press, Oxford, 1964, pp. 136-142.

9. Rust, B., Burrus, W., and Schneeberger, C.: A simple algorithm for computing the generalized inverse of a matrix, Comm. of the ACM, Vol. 9, No. 5, 1966, pp. 381-387.

10. Pei, M. L.: A test matrix for inversion procedures, Comm. of the ACM, Vol. 5, No. 10, 1962, p. 508.

11. La Sor, W.: Test matrix for inversion, Comm. of the ACM, Vol. 6, No. 3, 1963, p. 102.

12. Hoffmann, K. and Kunze, R.: Linear Algebra, Prentice-Hall, Englewood Cliffs, N. J., 1961.