```
      IF (ITR .EQ. 0) ITR = 9999
      DO 24 J = 1, IBS
      IF (NCOMP(N, I, J) .NE. -ITR) GOTO 24
      IF (ITR .EQ. 9999) ITR = 0
      NCOMP(N, I, J) = ITR
      GOTO 25
   24 CONTINUE
   25 CONTINUE
C
C        TRY TO REFIT REMOVED TREATMENT
C        OR REMOVE ANOTHER OCCURRENCE OF SAME TREATMENT
C
      IF (KREP .EQ. IR(IL)) GOTO 18
      N = ND
      GOTO 12
      END
```

Algorithm AS 225

Minimizing Linear Inequality Constrained Mahalanobis Distances

By Peter C. Wollan

Michigan Technological University, U.S.A.

and

Richard L. Dykstra†

University of Iowa, U.S.A.

[Received November 1985. Final revision December 1986]

Keywords: Mahalanobis Distance; Linear Inequality Constraints; Kuhn-Tucker Vectors; Iteration; Active Constraints

Language

Fortran 66

Description and Purpose

Least squares problems with affine equality constraints occur frequently in a variety of contexts, and efficient algorithms for their solution exist in several locations (such as Stirling

† *Address for correspondence*: Department of Statistics and Acturial Science University of Iowa, Iowa City, IA 52242, U.S.A.

 0035–9254/87/36234 $2.00

(1981). When the equality constraints are replaced by inequality constraints, the problem takes on a very different character, especially when the number of constraints is large. This is primarily because it is difficult to identify the active constraints. Here we present a Fortran program for obtaining the solution to the problem.

$$\underset{x:Ax\leqslant b}{\text{Minimize}}\ (g-x)'S^{-1}(g-x), \tag{1.1}$$

where $g^{n\times 1}$ and $b^{k\times 1}$ are specified vectors, $S^{n\times n}$ is a given positive definite matrix and

$$A^{k\times n}=\begin{bmatrix} a(1)' \\ a(2)' \\ \vdots \\ a(k)' \end{bmatrix}$$

denotes a matrix corresponding to k simultaneous affine constraints. The program implements an interative procedure proposed in Dykstra (1983) which in this context proceeds by cyclically estimating Kuhn-Tucker vectors. The estimated Kuhn-Tucker vectors are then used to form a simpler, approximating problem which is easily solvable. The Kuhn-Tucker estimates are then updated and the process repeated. The procedure is guaranteed to converge correctly if a solution exists, usually in an efficient manner. The procedure is exceptionally simple and straightforward and easy to program. The constraint matrix need not be full rank, and k may exceed n. Redundant constraints cause no problems, and in some cases may actually speed convergence.

The procedure simultaneously generates a sequence of estimated solutions $x_m^{n\times 1}$ and a sequence of Kuhn-Tucker vectors $u_m^{k\times 1}$ (where $u_m(j)$ denotes the jth coordinate of u_m corresponding to the jth constraint $a(j)'x\leqslant b(j)$). The procedure is initiated by setting $u^0\equiv 0$ and $x_0\equiv g$, and by defining $m'=m \bmod k=i$ if $m=lk+i$ for integers $l\geqslant 0$ and $1\leqslant i\leqslant k$. To move from (x_{m-1}, u_{m-1}) to (x_m, u_m), the algorithm sets

$$\tilde{y}=x_{m-1}+Sa(m')u_{m-1}(m')/2,$$

and then defines (x_m, u_m) by

$$u_m(j)=\begin{cases} u_{m-1}(j), & j\neq m' \\ \max\{0,2(a(j)'\tilde{y}-b(j))/a(j)'Sa(j)\}, & j=m', \end{cases}$$

$$x_m=\tilde{y}-Sa(m')u_m(m')/2.$$

The factor of 2 appears here because of the choice of squared Mahalanobis distance as the objective function. Even if the minimization problems are equivalent, different objective functions yield different values for the Kuhn-Tucker coefficients. In practice, one is usually interested only in whether or not the coefficient is zero. A non-zero value indicates that the corresponding constraint is active, while a zero value indicates that the constraint is inactive. In some sense, larger values indicate that the corresponding constraints are more important. The program terminates when x_{mk} and $x_{(m+1)k}$ are sufficiently close in the usual l_∞ norm, in which case the program returns $x_{(m+1)k}$ as its estimated solution (denoted by *XHAT* in program output).

A closely related problem for which the algorithm may be used is

$$\underset{x;\ x=\sum_1^k \alpha_i d_i,\ \alpha_i\geqslant 0 \text{ for all } i}{\text{Minimize}}\quad (g-x)'S^{-1}(g-x) \tag{1.2}$$

where $d_1^{n\times 1},\ldots, d_k^{n\times 1}$ are a specified collection of vectors. To use the algorithm for this problem, set

$$A = \begin{bmatrix} d_1'S^{-1} \\ d_2'S^{-1} \\ \vdots \\ d_k'S^{-1} \end{bmatrix}$$

in problem (1.1) and apply the algorithm. If x^* denotes the limiting value from the algorithm. $g - x^*$ solves problem (1.2). If the α_i's are also desired, they are given by one-half the Kuhn-Tucker vector (which is *XKT* in the output).

Equality constraints are handled for forcing them to be active: the *I*th constraint is treated as either an equality or inequality constraint, depending on the value of the indicator variable *IFLAG(I)*. If the problem is such that blocks of constraints can be handled simultaneously, greater economies are possible (see Dykstra and Robertson, 1982).

Restrictions

For simplicity, the program examines the input only enough to protect against dividing by zero. In particular, it does not verify the existence of a solution before proceeding. Thus the program will run even though there may not exist any vectors which are feasible (satisfy all constraints). However, since the procedure used must converge to the true solution (if it exists), failure to converge (*IFAULT* = 1 and *SUPDIF* relatively large) is evidence that there may not be any feasible vectors. The evidence for this increases if the program is repeated with a larger inputted value of *ITMAX*, and the outputted value of *SUPDIF* is not decreased.

Structure

SUBROUTINE LSTSQ(X, S, A, B, IFLAG, N, K, NWORK, ITMAX, EPS, EPS2, W, XHAT, XKT, ITER, SUPDIF, IFAULT

Formal parameters

X	Real array(*N*)	input: data vector
S	Real array(*N,N*)	input: covariance matrix
A	Real array(*K,N*)	input: constraint array The *I*th constraint is of the form $\sum_{j=1}^{N} A(J,I)*X(J)[\leqslant, =]B(I)$, $\leqslant$ if $IFLAG(I) = 0$, $=$ if $IFLAG(I) = 1$
B	Real array(*K*)	input: constraint constants
IFLAG	Integer array(*K*)	input: if *IFLAG(I)* = 0, the *I*th constraint is an inequality constraint. If *IFLAG(I)* = 1, it is an equality constraint
N	Integer	input: length of the data vector
K	Integer	input: number of constraints
NWORK	Integer	input: length of workspace array: must be at least $2*N*K + N + K$
ITMAX	Integer	input: upper bound for number of iterations to be attempted
EPS	Real	input: accuracy parameter for the convergence criterion. The program terminates when *SUPDIF* is smaller than *EPS*
EPS2	Real	input: if a real number is less than *EPS2* in absolute value, it is considered to be zero
W	Real array(*NWORK*)	workspace
XHAT	Real array(*N*)	output: estimate vector
XKT	Real array(*K*)	output: vector of Kuhn-Tucker coefficients
ITER	Integer	output: number of iterations carried out.

SUPDIF	Real	output: greatest difference between estimates across a full cycle
IFAULT	Integer	output: error indicator 0: no error. 1: ITMAX exceeded. 2: invalid constants among *N, K, NWORK, ITMAX, EPS, or EPS2* 3: invalid constraint function. For some row $A(I)*S*A(I)'=0$ 4: insufficient workspace

Applications

Many problems can be phrased in the form of (1.1) and/or (1.2). For example, consider the linear model

$$Y = X\beta + \varepsilon$$

where Y^{mx1} is a vector of observations, X^{mxn} is a full rank design matrix, the components of ε are independent $n(0, \sigma^2)$ random variables, and the regression coefficients $\beta_1, \ldots, \beta_n$ have linear inequality restrictions imposed upon them.

If the restrictions are that the β_1 are non-negative, the problem of finding the maximum likelihood estimate of β is exactly of the form of (1.2) where the d_i's are the columns of X, g is the observed vector y, and S is the identity matrix. The minimizing α is of course the desired maximum likelihood estimate of β.

However, since the projection onto a convex set contained within a subspace can be accomplished by an iterated projection, the above problem can also be phrased as problem (1.1) where g is the unrestricted MLE of β, $S^{-1} = X'X$, and $A\beta \leqslant b$ denotes the inequality constraints on the regression coefficients. Since this problem is phrased in a lower dimensional space, this is usually preferable.

As another example, several authors have considered the problem of finding at least squares fit to a set of points subject to the fitted points being concave (or convex), e.g. Hildreth (1954). This problem can be phrased in the form of (1.1) by taking the ith row of A to have 1, -2, and 1 in the ith, $(i+1)$th and $(i+2)$th positions and zero everywhere else, and setting b equal to zero.

Another important application concerns maximum likelihood estimation problems with ordered parameters. Problems of this nature can often be solved by linearly restricted least squares projections for a wide variety of different families of distributions. For example, ordered parameter MLE's for binomial, Poisson, multinomial, gamma, geometric and other families all fall into this category (see Barlow, Bartholomew, Bremner and Brunk (1972)). Since these problems can be solved by least squares projections, this algorithm is applicable for obtaining the MLE's.

Time and Accuracy

Convergence is generally quite fast. For the problem of projecting random data, with a fixed non-diagonal covariance matrix, onto the linear-order isotonic cone in R^{10}, convergence to within $\pm 10^{-5}$ was usually obtained within 75 iterations, using no more than 2 seconds of CPU time on a PRIME 850 computer. However, the convergence rate depends on the eigenvalues of the covariance matrix, the configuration of the data, the covariance eigenvectors and the constraint vectors.

Intuitively, the program adjusts coordinates of XHAT to satisfy each constraint in turn. In some situations, convergence may be speeded up by adding redundant constraints. For example, suppose the constraints are $x_1 - x_2 \leqslant 0$, $x_2 - x_3 \leqslant 0$, and $x_3 - x_4 \leqslant 0$, $S^{-1} = \text{diag}(w_1, w_2, w_3, w_4)$ where w_1 and w_4 are large and w_2 and w_3 are small, and $g_1 \geqslant g_2 \geqslant g_3 \geqslant g_4$. Then the solution will have all coordinates equal to a weighted average of the g_i's. The algorithm will proceed by successive weighted averagings between pairs of coordinates. However, largely because of the unequal weights, it may take a long time until the solution is reached. By adding the redundant constraint $x_1 - x_4 \leqslant 0$, convergence is much faster. It is much easier to identify redundant constraints which may be helpful when dealing with order constraints than when dealing with general linear inequalities. We know of no systematic method for adding additional redundant constraints to improve convergence, however.

Accuracy is determined by the variable *EPS*: the program terminates when the estimate under each constraint changes by less than *EPS*, in the usual l_∞ norm. If greater accuracy is needed than can be obtained with single precision arithmetic, the program can readily be converted to double precision by including a type statement of the form *"IMPLICIT DOUBLE PRECISION A-H, P-Z"*, replacing the function *ABS* with *DABS*, and replacing the constants in the *DATA* statement.

Acknowledgement

We would like to thank the editor and referee for their helpful comments. This work was partially supported by ONR Contract N00014-83-K-0249.

References

Barlow, R. E., Bartholomew, D. J., Bremner, J. M., and Brunk, H. D. (1972) *Statistical Inference Under Order Restrictions*. New York: Wiley.

Dykstra, Richard L. (1983) An algorithm for restricted least squares regression. *J. Amer. Statist. Ass.*, **78**, 837–842.

Dykstra, Richard L. and Robertson, Tim (1982). An algorithm for isotonic regression for two or more independent variables. *Ann. Statist.*, **10**, 708–716.

Hildreth, C. (1954) Point estimates of ordinates of concave functions. *J. Amer. Statist. Ass.*, **49**, 598–619.

Stirling, W. D. (1981) Algorithm AS 164. Least squares subject to linear constraints. *Appl. Statist.*, **30**, 204–212.

```
      SUBROUTINE LSTSQ(X, S, A, B, IFLAG, N, K, NWORK, ITMAX, EPS, EPS2,
     *  W, XHAT, XKT, ITER, SUPDIF, IFAULT)
      DIMENSION X(N), S(N, N), A(K, N), B(K), IFLAG(K), W(NWORK),
     *  XHAT(N), XKT(K)
C
C        ALGORITHM AS225 APPL. STATIST. (1987) VOL. 36, NO. 2
C
C        COMPUTES THE LEAST-SQUARES PROJECTION OF X ONTO
C        THE INTERSECTION OF K SIMULTANEOUS AFFINE CONSTRAINTS,
C        USING THE MAHALANOBIS DISTANCE DETERMINED BY S.
C        THE ITH CONSTRAINT IS OF THE FORM
C        SUM OVER J OF A(I,J)*X(J) (.LE.,.EQ.) B(I)
C        --.LE. IF IFLAG(I) .EQ. 0
C        --.EQ. IF IFLAG(I) .EQ. 1
C
      DATA ZERO, TWO /0.0E0, 2.0E0/
C
      IFAULT = 0
      NK = N * K
      NK2 = NK + NK
      NK2K = NK2 + K
C
C        CHECK INPUTS.
C
      IF (N .LE. 0 .OR. K .LE. 0 .OR. ITMAX .LE. 0 .OR. EPS .LE.
     *  ZERO .OR. NWORK .LE. 0 .OR. EPS2 .LE. ZERO) GOTO 300
      IF (NWORK .LT. NK2K + N) GOTO 320
      DO 10 I = 1, K
      IF (IFLAG(I) .EQ. 0 .OR. IFLAG(I) .EQ. 1) GOTO 10
      GOTO 300
   10 CONTINUE
```

```
C       INITIALIZE ARRAYS, AND COMPUTE
C       MATRIX PRODUCTS A*S AND DIAG(A*S*A').
C       WORKSPACE STRUCTURE IS AS FOLLOWS:
C       W(1)-W(N*K):  ARRAY OF INCREMENT VECTORS.  ITH INC VECTOR
C       IS (XKT(I)/2)*S*A(I,.).
C       W(NK+1)-W(N*K*2):  MATRIX A*S.
C       W(NK2+1)-W(NK2+K):  VECTOR DIAG(A*S*A').
C       W(NK2K+1)-W(NK2K+N):  PREVIOUS XHAT VECTOR, TO CHECK FOR CONV.
C
      DO 40 I = 1, K
      DO 30 J = 1, N
      INDEX = NK + (J - 1) * K + I
      W(INDEX) = ZERO
      DO 20 L = 1, N
   20 W(INDEX) = W(INDEX) + A(I, L) * S(L, J)
   30 CONTINUE
   40 CONTINUE
      DO 60 I = 1, K
      INDEX = NK2 + I
      W(INDEX) = ZERO
      DO 50 J = 1, N
      IND2 = NK + (J - 1) * K + I
      W(INDEX) = W(INDEX) + A(I, J) * W(IND2)
   50 CONTINUE
      IF (ABS(W(INDEX)) .LE. EPS2) GOTO 310
   60 CONTINUE
      DO 80 J = 1, N
      INDEX = NK2K + J
      W(INDEX) = X(J)
      XHAT(J) = X(J)
      DO 70 I = 1, K
      INDEX = (I - 1) * N + J
      W(INDEX) = ZERO
   70 CONTINUE
   80 CONTINUE
      ITER = 0
C
C       THE ITERATION LOOP:  LINES 100-200.
C
  100 ITER = ITER + 1
      SUPDIF = ZERO
      DO 200 I = 1, K
C
C       REMOVE OLD INCREMENT VECTOR:
C
      DO 110 J = 1, N
      INDEX = (I - 1) * N + J
      XHAT(J) = XHAT(J) + W(INDEX)
  110 CONTINUE
C
C       EVALUATE ITH CONSTRAINT:
C
      SUM = ZERO
      DO 120 J = 1, N
  120 SUM = SUM + A(I, J) * XHAT(J)
      SUM = SUM - B(I)
      IF (IFLAG(I) .EQ. 0 .AND. SUM .LE. ZERO) GOTO 140
C
C       COMPUTE NEW INCREMENT VECTOR THAT FORCES EQUALITY IN ITH CONST.
C
      INDEX = NK2 + I
      TEMP = SUM / W(INDEX)
      DO 130 J = 1, N
      IND1 = (I - 1) * N + J
      IND2 = NK + (J - 1) * K + I
      W(IND1) = W(IND2) * TEMP
      XHAT(J) = XHAT(J) - W(IND1)
  130 CONTINUE
      GOTO 160
C
C       IF CONSTRAINT WAS SATISFIED, SET INCREMENT TO ZERO:
```

```
  140 DO 150 J = 1, N
      INDEX = (I - 1) * N + J
      W(INDEX) = ZERO
  150 CONTINUE
C
C        FIND LARGEST CHANGE, AND CHECK FOR CONVERGENCE:
C
  160 DO 170 J = 1, N
      INDEX = NK2K + J
      ABDIF = ABS(XHAT(J) - W(INDEX))
      IF (SUPDIF .LT. ABDIF) SUPDIF = ABDIF
  170 CONTINUE
  200 CONTINUE
      IF (SUPDIF .LE. EPS) GOTO 400
      DO 210 J = 1, N
      INDEX = NK2K + J
      W(INDEX) = XHAT(J)
  210 CONTINUE
      IF (ITER .LT. ITMAX) GOTO 100
      IFAULT = 1
      RETURN
  300 IFAULT = 2
      RETURN
  310 IFAULT = 3
      RETURN
  320 IFAULT = 4
      RETURN
C
C        COMPUTE KUHN-TUCKER COEFFICIENTS AND RETURN.
C
  400 DO 430 I = 1, K
C
C        FIND A NON-ZERO DENOMINATOR:
C
      DO 410 J = 1, N
      INDEX = NK + (J - 1) * K + I
      IF (ABS(W(INDEX)) .GT. EPS2) GOTO 420
  410 CONTINUE
  420 IND2 = (I - 1) * N + J
      XKT(I) = TWO * W(IND2) / W(INDEX)
  430 CONTINUE
      RETURN
      END
```