Algorithm AS 135: Min-Max Estimates for a Linear Multiple Regression Problem

Author(s): Ronald D. Armstrong and David S. Kung

Source: *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, Vol. 28, No. 1 (1979), pp. 93-100
Published by: Wiley for the Royal Statistical Society
Stable URL: http://www.jstor.org/stable/2346829
Accessed: 27-02-2015 13:20 UTC

```
    5 XB = XA
      GB = GA
      XA = X
      GA = GX
      GOTO 4
C
C        CALCULATE REMAINING CONSTANTS
C
    6 T = X
      R = BT / (BT + ALPHA * AA ** BETA)
      RETURN
      END
C
      SUBROUTINE FNE(REX)
C
C        ALGORITHM AS 134.3  APPL. STATIST. (1979) VOL.28, NO.1
C
C        GENERATES EXPONENTIAL RANDOM VARIABLES
C        BY THE METHOD OF VON NEUMANN
C
      A = 0.0
    1 U = RANF(0.0)
      UO = U
    2 USTAR = RANF(1.0)
      IF (U .LT. USTAR) GOTO 3
      U = RANF(2.0)
      IF (U .LT. USTAR) GOTO 2
      A = A + 1.0
      GOTO 1
    3 REX = A + UO
      RETURN
      END
```

Algorithm AS 135

Min-Max Estimates for a Linear Multiple Regression Problem

By Ronald D. Armstrong and David S. Kung

University of Texas at Austin, Austin, Texas

Keywords: LINEAR PROGRAMMING; REGRESSION; CHEBYCHEV NORM; MIN–MAX

Language

ISO Fortran

Description and Purpose

Let $(x_{i1}, x_{i2}, \ldots, x_{im}, y_i)$, $i = 1, 2, \ldots, n$, be given. The min–max curve fitting problem is to find $\beta = (\beta_1, \beta_2, \ldots, \beta_m)$ to

$$\text{minimize}\left(\text{maximum}\left|y_i - \sum_{j=1}^{m} x_{ij}\beta_j\right|, i = 1, 2, \ldots, n\right). \tag{1}$$

Problem (1) is often termed a Chebychev or L_∞ norm curve-fitting problem. It provides an alternative to the classical least squares analysis and may be particularly attractive if the error distribution is uniform. The reader is referred to Appa and Smith (1973) and Harter (1975) for a further disucssion of min–max properties.

It has been known for some time (see Stiefel, 1960) that (1) is equivalent to the following linear programming (LP) problem.

$$\text{Minimize } z, \text{ subject to } \quad y_i - z \leqslant \sum_{j=1}^{m} x_{ij}\beta_j \leqslant y_i + z, \quad i = 1, 2, \ldots, n. \tag{2}$$

The computer code presented here is based on the algorithm of Armstrong and Kung (1977) which utilizes an LP dual method to solve (2). This algorithm differs from a dual method presented by Stiefel (1959) in certain important aspects. It is a revised simplex algorithm which maintains a basis of size m by m rather than $(m+1)$ by $(m+1)$. It employs an LU decomposition as described by Bartels and Golub (1969) to obtain the solutions to square linear systems. The method guarantees that an observation $(x_{i1}, ..., x_{im})$ removed from the basis at an iteration will not violate its associated constraint immediately after removal. Due to the special structure of the problem, the total number of iterations required by the standard simplex algorithm can be reduced significantly; there are times when two or more iterations may be combined into one. Also, in deciding the observation to leave the basis, the amount of computation is reduced to finding the minimum of m ratios. These lead to a significant saving in overall computational time.

Computational Result

The algorithm was tested together with the Barrodale and Phillips (1975) computer code for the Chebychev problem. The two codes were placed in a program as independent (i.e. no common blocks were present) subroutines. Several runs were made with randomly generated problems of various dimensions and the results are summarized in Table 1. The number of iterations refers to basis updates required. In terms of numerical accuracy, for the problems we solved, all objective values corresponded to ten digits. All runs were performed on a CDC 6600 with a 60-bit word.

Table 1

A summary of computational testing with two algorithms for Chebychev curve fitting. Five problems were solved at each level and all figures are the means of the results. All times are in milliseconds on a CDC 6600

n [*In each pair of rows* 1*st row: Time* 2*nd row: Iterations*]	m							
	5		10		15		20	
	B–P†	*A–K*†	*B–P*	*A–K*	*B–P*	*A–K*	*B–P*	*A–K*
50	134	42	337	216	701	585	1098	1442
	13	7	22	14	34	18	42	25
100	255	105	778	400	1639	1141	2571	2316
	13	11	25	20	40	30	50	35
200	637	174	1928	634	4009	1818	6839	3743
	16	11	32	21	49	35	67	46
200	689	165	1877	660	3434	1538	6035	3927
	17	10	31	23	42	30	59	48
300	906	257	2779	891	5977	2563	10831	5369
	15	11	30	23	49	40	70	54
350	1198	287	3806	1050	7896	2826	12661	5702
	17	11	36	24	55	40	70	53

† *B–P*: Barrodale and Phillips (1975). *A–K*: Algorithm from this paper.

Structure

SUBROUTINE LFNORM (*N, M, NDIM, MDIM, X, Y, BETA, Z, KY, IFAULT*)

Formal parameters

N	Integer	input:	number of observations
M	Integer	input:	number of independent variables

NDIM	Integer	input:	first dimension of *X* and dimension of *Y*
MDIM	Integer	input:	second dimension of *X*, and dimension of *BETA*
X	Real array (*NDIM*, *MDIM*)	input:	values of the independent variables such that each row corresponds to an observation
Y	Real array (*NDIM*)	input:	values of the dependent variable
BETA	Real array (*MDIM*)	output:	final estimates of the coefficients of the problem
Z	Real	output:	the least maximum absolute deviation
KY	Integer	output:	the iteration counter
IFAULT	Integer	output:	the failure indicator = 0 normal termination = 1 observation matrix of less than full rank

Restrictions

The local constants are *ACU* and *BIG* which have the values 10^{-8} and 10^{15} respectively. *ACU* is used to test for optimality. Also, if the absolute value of a number is smaller than *ACU*, it will be treated as zero. *BIG* is used as an initial value when determining the minimum ratio.

Acknowledgement

This research was supported in part by a George Kozmetsky Fellowship Grant and NSF Grant MCS77–00100.

References

Appa, G. and Smith, C. (1973). On L_1 and Chebyshev estimation. *J. Math. Programming*, **5**, 73–87.

Armstrong, R. D. and Kung, D. S. (1977). A dual method for discrete Chebychev curve fitting. Working Paper 78–9, The University of Texas at Austin, 23 pp.

Barrodale, L. and Phillips, C. (1975). Solution of an overdetermined system of linear equations in the Chebychev norm. *ACM Transactions on Mathematical Software*, **1**, 264–270.

Bartels, R. G. and Golub, G. H. (1969). The simplex method of linear programming using LU decomposition. *Commun. Ass. Comp. Mach.*, **12**, 266–268.

Harter, H. L. (1975). The method of least squares and some alternatives—Part III. *Int. Statist. Rev.*, **43**, 1–44.

Stiefel, E. (1960). Note on Jordan elimination, linear programming and Tschebyscheff approximation. *Numer. Math.*, **2**, 1–17.

—— (1959). Über diskrete und lineare Tschebyscheff Approximationen. *Numer. Math.*, **1**, 1–28.

```
      SUBROUTINE LFNORM(N, M, NDIM, MDIM, X, Y, BETA, Z, KY, IFAULT)
C
C        ALGORITHM AS 135  APPL. STATIST. (1979) VOL.28, NO.1
C
C        MIN-MAX ESTIMATES FOR A LINEAR MULTIPLE REGRESSION PROBLEM
C
      DIMENSION X(NDIM, MDIM), Y(NDIM), LU(20, 20), BETA(MDIM)
      DIMENSION HILO(20), XRXF(20), XSXF(20), IBASE(20), INDEX(20)
      REAL LU
      INTEGER SSS, RRR
      LOGICAL INTL
C
      DATA ACU /1.0E-8/, BIG /1.0E15/
C
      IFAULT = 0
      KY = 0
      Z = 0.0
      M1 = M - 1
C
C        SET UP INITIAL LU DECOMPOSITION
```

```
C
      DO 10 I = 1, M
   10 INDEX(I) = I
      INTL = .TRUE.
      KKK = 1
      CALL UPDATE(KKK, X, LU, IBASE, INDEX, INTL,
     *  N, M, NDIM, MDIM, IFAULT)
      IF (IFAULT .NE. 0) RETURN
      INTL = .FALSE.
      IROW = KKK
C
C        CALCULATE BETA VALUE
C
      K = INDEX(1)
      K1 = IBASE(1)
      BETA(K) = Y(K1) / LU(K, 1)
      DO 30 II = 2, M
      K = INDEX(II)
      K1 = IBASE(II)
      BETA(K) = Y(K1)
      II1 = II - 1
      DO 20 I = 1, II1
      KK = INDEX(I)
      BETA(K) = BETA(K) - LU(KK, II) * BETA(KK)
   20 CONTINUE
      BETA(K) = BETA(K) / LU(K, II)
   30 CONTINUE
      DO 40 II = 1, M1
      K1 = M - II
      K = INDEX(K1)
      DO 40 I = 1, II
      KK = M - I + 1
      K2 = INDEX(KK)
      BETA(K) = BETA(K) - LU(K2, K1) * BETA(K2)
   40 CONTINUE
C
C        SEARCH FOR AND SET FIRST VIOLATED
C        CONSTRAINT AS RTH CONSTRAINT
C
   50 IROW = IROW + 1
      IF (IROW .GT. N) RETURN
      DEV1 = 0.0
      DO 60 I = 1, M
   60 DEV1 = DEV1 + X(IROW, I) * BETA(I)
      DEV1 = DEV1 - Y(IROW)
      IF (ABS(DEV1) .LT. ACU) GOTO 50
      SIGR = SIGN(1.0, DEV1)
      RRR = IROW
C
C        ADJUST FOR THE RTH CONSTRAINT
C
      K = INDEX(1)
      XRXF(1) = X(RRR, K)
      DO 80 II = 2, M
      K = INDEX(II)
      XRXF(II) = X(RRR, K)
      II1 = II - 1
      DO 70 I = 1, II1
   70 XRXF(II) = XRXF(II) - LU(K, I) * XRXF(I)
   80 CONTINUE
      K = INDEX(M)
      XRXF(M) = XRXF(M) / LU(K, M)
      HILO(M) = SIGN(1.0, -SIGR * XRXF(M))
      SUMXR = SIGR - HILO(M) * XRXF(M)
      DO 100 II = 1, M1
      K1 = M - II
      K = INDEX(K1)
      DO 90 I = 1, II
      K2 = M - I + 1
      XRXF(K1) = XRXF(K1) - LU(K, K2) * XRXF(K2)
   90 CONTINUE
      XRXF(K1) = XRXF(K1) / LU(K, K1)
      HILO(K1) = SIGN(1.0, -SIGR * XRXF(K1))
      SUMXR = SUMXR - HILO(K1) * XRXF(K1)
```

```
  100 CONTINUE
      Z = ABS(DEV1 / SUMXR)
C
C        START OF MAIN ITERATIVE LOOP.
C        SEARCH FOR THE MOST VIOLATED STH CONSTRAINT
C
  110 SSS = 0
      DEVIAT = ACU
C
C        CALCULATE BETA VALUE
C
      K = INDEX(1)
      K1 = IBASE(1)
      BETA(K) = (Y(K1) + Z * HILO(1)) / LU(K, 1)
      DO 130 II = 2, M
      K = INDEX(II)
      K1 = IBASE(II)
      BETA(K) = Y(K1) + Z * HILO(II)
      II1 = II - 1
      DO 120 I = 1, II1
      KK = INDEX(I)
      BETA(K) = BETA(K) - LU(KK, II) * BETA(KK)
  120 CONTINUE
      BETA(K) = BETA(K) / LU(K, II)
  130 CONTINUE
      DO 140 II = 1, M1
      K1 = M - II
      K = INDEX(K1)
      DO 140 I = 1, II
      KK = M - I + 1
      K2 = INDEX(KK)
      BETA(K) = BETA(K) - LU(K2, K1) * BETA(K2)
  140 CONTINUE
C
C        CALCULATE RESIDUALS
C
      DO 160 I = 1, N
      YEST = 0.0
      DO 150 J = 1, M
  150 YEST = YEST + X(I, J) * BETA(J)
      DEV1 = ABS(Y(I) - YEST) - Z
      IF (DEV1 .LE. DEVIAT) GOTO 160
      YDEV = YEST - Y(I)
      DEVIAT = DEV1
      SSS = I
  160 CONTINUE
C
C        CHECK IF AT OPTIMAL
C
      IF (SSS .EQ. 0) RETURN
C
C        SET UP INFORMATION ON THE S-TH CONSTRAINT
C
      SIGS = SIGN(1.0, YDEV)
      K = INDEX(1)
      XSXF(1) = X(SSS, K)
      DO 180 II = 2, M
      K = INDEX(II)
      XSXF(II) = X(SSS, K)
      II1 = II - 1
      DO 170 I = 1, II1
  170 XSXF(II) = XSXF(II) - LU(K, I) * XSXF(I)
  180 CONTINUE
      K = INDEX(M)
      XSXF(M) = XSXF(M) / LU(K, M)
      SUMXS = -SIGS + HILO(M) * XSXF(M)
      DO 200 II = 1, M1
      K1 = M - II
      K = INDEX(K1)
      DO 190 I = 1, II
      K2 = M - I + 1
      XSXF(K1) = XSXF(K1) - LU(K, K2) * XSXF(K2)
```

```
  190 CONTINUE
      XSXF(K1) = XSXF(K1) / LU(K, K1)
      SUMXS = SUMXS + HILO(K1) * XSXF(K1)
  200 CONTINUE
C
C        SEARCH FOR MINIMUM RATIO
C
  210 KKK = 0
      RATIO = BIG
      DO 220 I = 1, M
      IF (SIGS * SIGN(1.0, XSXF(I)) .NE. HILO(I) .OR.
     *  ABS(XSXF(I)) .LT. ACU) GOTO 220
      TEST = ABS(XRXF(I) / XSXF(I))
      IF (TEST .GE. RATIO) GOTO 220
      RATIO = TEST
      KKK = I
  220 CONTINUE
C
C        CHECK IF R-TH CONSTRAINT MOVES INTERIOR
C
      IF (KKK .NE. 0) GOTO 260
C
C        PROCESS THE MOVEMENT OF THE R-TH CONSTRAINT
C
      DELTA = ABS(DEVIAT / SUMXS)
C
C        CALCULATE THE LARGEST TOLERABLE DELTA
C
      DIV = ABS(SUMXR) - 2.0
      IF (DIV .LT. ACU) GOTO 240
      SWING = 2.0 * Z / DIV
      IF (SWING .GE. DELTA) GOTO 240
C
C        SWITCH R AND S CONSTRAINT INDICATORS
C
      SAVE = SUMXS
      SUMXS = -SUMXR + SIGR + SIGR
      SUMXR = -SAVE
      SAVE = SIGR
      SIGR = SIGS
      SIGS = -SAVE
      DEVIAT = ABS(SUMXS * DELTA) - 2.0 * Z
      Z = Z + DELTA
      DO 230 I = 1, M
      SAVE = XSXF(I)
      XSXF(I) = XRXF(I)
      XRXF(I) = SAVE
  230 CONTINUE
      I = RRR
      RRR = SSS
      SSS = I
      GOTO 210
C
C        REPLACE THE R-TH CONSTRAINT WITH THE S-TH CONSTRAINT
C
  240 SIGR = SIGS
      DO 250 I = 1, M
  250 XRXF(I) = XSXF(I)
      SUMXR = -SUMXS
      Z = Z + DELTA
      RRR = SSS
      GOTO 110
C
C        PROCESS THE MOVEMENT OF THE K-TH CONSTRAINT
C
  260 DELTA = ABS(XRXF(KKK) * DEVIAT /
     *  (XRXF(KKK) * SUMXS + XSXF(KKK) * SUMXR))
      TOP = -2.0 * Z * XRXF(KKK)
      DIV = XRXF(KKK) * XRXF(KKK) + HILO(KKK) * SUMXR
      IF (SIGN(1.0, TOP) .NE. SIGN(1.0, DIV)) GOTO 270
      IF (ABS(DIV) .LT. ACU) GOTO 270
      SWING = TOP / DIV
```

```
C         CHECK TO SEE IF THE K-TH CONSTRAINT SWINGS ACROSS
C
      IF (SWING .GE. DELTA) GOTO 270
      Z = Z + SWING
      DEVIAT = DEVIAT - SWING *
     *  ABS(SUMXS + XSXF(KKK) * SUMXR / XRXF(KKK))
      SUMXR = SUMXR + 2.0 * HILO(KKK) * XRXF(KKK)
      SUMXS = SUMXS - 2.0 * HILO(KKK) * XSXF(KKK)
      HILO(KKK) = -HILO(KKK)
      GOTO 210
C
C         UPDATE XRXF AND THE LU OF THE CURRENT BASIS
C
  270 HILO(KKK) = SIGS
      SUMXR = SIGR
      XRXF(KKK) = XRXF(KKK) / XSXF(KKK)
      SUMXR = SUMXR - HILO(KKK) * XRXF(KKK)
      DO 280 I = 1, M
      IF (I .EQ. KKK) GOTO 280
      XRXF(I) = XRXF(I) - XSXF(I) * XRXF(KKK)
      SUMXR = SUMXR - HILO(I) * XRXF(I)
  280 CONTINUE
      IBASE(KKK) = SSS
C
C         UPDATE LU DECOMPOSITION
C
      CALL UPDATE(KKK, X, LU, IBASE, INDEX, INTL,
     *  N, M, NDIM, MDIM, IFAULT)
      IF (IFAULT .NE. 0) RETURN
      Z = Z + DELTA
      KY = KY + 1
      GOTO 110
      RETURN
      END
C
      SUBROUTINE UPDATE(KKK, X, LU, IBASE, INDEX, INTL,
     *  N, M, NDIM, MDIM, IFAULT)
C
C         ALGORITHM AS 135.1  APPL. STATIST. (1979) VOL.28, NO.1
C
C         UPDATE LU DECOMPOSITION MATRIX
C
      DIMENSION X(NDIM, MDIM), LU(20, 20), IBASE(20), INDEX(20)
      REAL LU
      LOGICAL INTL
      DATA ACU /1.0E-8/
C
      IROW = 0
      DO 90 II = KKK, M
      IF (INTL) GOTO 10
      IROW = IBASE(II)
      GOTO 20
   10 IROW = IROW + 1
      IF (IROW .LE. N) GOTO 20
      IFAULT = 1
      RETURN
   20 DO 30 I = 1, M
   30 LU(I, II) = X(IROW, I)
C
C         SET UP REPRESENTATION OF INCOMING ROW
C
      IF (II .EQ. 1) GOTO 60
      II1 = II - 1
      DO 50 ICOL = 1, II1
      K = INDEX(ICOL)
      SUBT = LU(K, II)
      J = ICOL + 1
      DO 40 I = J, M
      K = INDEX(I)
      LU(K, II) = LU(K, II) - SUBT * LU(K, ICOL)
   40 CONTINUE
   50 CONTINUE
```

```
C          FIND MAXIMUM ENTRY
C
   60 PIVOT = ACU
      KK = 0
      DO 70 I = II, M
      K = INDEX(I)
      IF (ABS(LU(K, II)) .LE. PIVOT) GOTO 70
      PIVOT = ABS(LU(K, II))
      KK = I
   70 CONTINUE
      IF (KK .EQ. 0) GOTO 10
C
C          SWITCH ORDER
C
      ISAVE = INDEX(KK)
      INDEX(KK) = INDEX(II)
      INDEX(II) = ISAVE
C
C          PUT IN COLUMNS OF LU ONE AT A TIME
C
      IF (INTL) IBASE(II) = IROW
      IF (II .EQ. M) GOTO 90
      J = II + 1
      DO 80 I = J, M
      K = INDEX(I)
      LU(K, II) = LU(K, II) / LU(ISAVE, II)
   80 CONTINUE
   90 CONTINUE
      KKK = IROW
      RETURN
      END
```

Algorithm AS 136

A *K*-Means Clustering Algorithm

By J. A. Hartigan and M. A. Wong

Yale University, New Haven, Connecticut, U.S.A.

Keywords: *K*-MEANS CLUSTERING ALGORITHM; TRANSFER ALGORITHM

Language

ISO Fortran

Description and Purpose

The *K*-means clustering algorithm is described in detail by Hartigan (1975). An efficient version of the algorithm is presented here.

The aim of the *K*-means algorithm is to divide M points in N dimensions into K clusters so that the within-cluster sum of squares is minimized. It is not practical to require that the solution has minimal sum of squares against all partitions, except when M, N are small and $K = 2$. We seek instead "local" optima, solutions such that no movement of a point from one cluster to another will reduce the within-cluster sum of squares.

Method

The algorithm requires as input a matrix of M points in N dimensions and a matrix of K initial cluster centres in N dimensions. The number of points in cluster L is denoted by $NC(L)$. $D(I, L)$ is the Euclidean distance between point I and cluster L. The general procedure is to search for a K-partition with locally optimal within-cluster sum of squares by moving points from one cluster to another.