Algorithm AS 141: Inversion of a Symmetric Matrix in Regression Models
Author(s): Philippe Kent
Source: *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, Vol. 28, No. 2 (1979), pp. 214-217
Published by: Wiley for the Royal Statistical Society
Stable URL: http://www.jstor.org/stable/2346751
Accessed: 25-06-2016 00:04 UTC

```
   62 GLOBAL = .FALSE.
      DO 63 I = 1, NCLUS
      IOLD(I) = INEW(I)
      INEW(I) = 0
   63 CONTINUE
      GOTO 4
C
C        COMPUTE OVERALL LOG LIKELIHOOD
C
   70 R1(1) = 0.0
      DO 72 I = 1, NCLUS
      R1(1) = R1(1) + XLIKE(0.0, P(I, I), I, I, SIZE(I) - 1,
     *  SIZE(I), -1.0, TLOG, MXS2)
      DO 71 J = 1, NCLUS
      IF (I .NE. J) R1(1) = R1(1) + XLIKE(0.0, P(I, J),
     *  I, I, SIZE(I), SIZE(J), -1.0, TLOG, MXS2)
   71 CONTINUE
   72 CONTINUE
      RETURN
      END
C
      REAL FUNCTION XLIKE(P1, R1, S1, S2, S3, S4, Y1, TLOG, MXS2)
C
C        ALGORITHM AS 140.3  APPL. STATIST. (1979) VOL.28, NO.2
C
C        EVALUATE THE CHANGE IN LOG LIKELIHOOD BETWEEN P SUCCESSES IN
C        S1 * S2 TRIALS AND P1 - Y1 * R1 SUCCESSES IN S3 * S4 TRIALS.
C
      INTEGER S1, S2, S3, S4, P, R, X, Z
      REAL TLOG(MXS2)
      XLIKE = 0.0
      P = P1
      Z = S1 * S2
      R = Z - P
      IF (R .NE. 0 .AND. P .NE. 0) XLIKE = TLOG(Z) - TLOG(P) - TLOG(R)
      X = P1 - Y1 * R1
      Z = S3 * S4
      R = Z - X
      IF (R .NE. 0 .AND. X .NE. 0)
     *  XLIKE = XLIKE + TLOG(X) + TLOG(R) - TLOG(Z)
      RETURN
      END
```

Algorithm AS 141

Inversion of a Symmetric Matrix in Regression Models

By PHILIPPE KENT

Department of Mathematics, Ecole Polytechnique Fédérale, Lausanne, Switzerland

LANGUAGE

ISO Fortran

INTRODUCTION

In a regression model $\mathbf{Y} = \mathbf{Xb}$, $\mathbf{b}$ is estimated by $(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$. To obtain the regression without a particular variable and thence a partial F value for that variable, $(\mathbf{W}'\mathbf{W})^{-1}\mathbf{W}'\mathbf{Y}$ may be used where $\mathbf{W}$ is obtained from $\mathbf{X}$ by deleting the column in $\mathbf{X}$ corresponding to the variable.

The Fortran subroutine *SINV* computes $(\mathbf{W}'\mathbf{W})^{-1}$ directly from $(\mathbf{X}'\mathbf{X})^{-1}$, achieving a significant gain in time compared to the inversion of $(\mathbf{W}'\mathbf{W})$. It is largely based on Algorithm

AS 37 (Garside, 1971) and Remark AS R9 (Knight, 1974) which permit a pivot-by-pivot inversion of a symmetric matrix. Conventional inversion of a symmetric matrix can also be performed.

It is also possible to call *SINV* with the "working" dimension of the X′X matrix smaller than that defined in a *DIMENSION* statement in the calling program. This allows the user to eliminate physically a variable, for example, by shifting up and left the appropriate rows and columns of a previous X′X or $(X'X)^{-1}$ matrix and subsequently disregarding the last row(s)/column(s). (This must be performed in the user's program.) The resulting matrix, of smaller working dimension, would then be considered a new X′X or $(X'X)^{-1}$ for future calls to *SINV*. Concurrent adjustments should be made to X′Y.

Purpose

Subroutine *SINV* performs one of three possible operations on a symmetric matrix depending on the value of *LO*:

1. Inversion of the matrix, if $LO = 0$ {(X′X) to $(X'X)^{-1}$}.
2. Inversion of the matrix ignoring a specified row/column, if $LO > 0$ {(X′X) to $(W'W)^{-1}$}.
3. Inversion of the matrix ignoring a specified row/column when the input matrix in *A* is the (previously calculated) inverse of the original matrix, if $LO < 0$ {$(X'X)^{-1}$ to $(W'W)^{-1}$}.

Structure

SUBROUTINE SINV(A, K, L, LO, PVT, IFAULT)

Formal parameters

A	Real array (*L*,*L*)	input:	X′X matrix if *LO.GE.*0 $(X'X)^{-1}$ matrix if *LO.LT.*0
		output:	$(X'X)^{-1}$ matrix (ignoring row/column *IABS*(*LO*) if *LO.NE.*0)
K	Integer	input:	working dimension of the input matrix; the matrix occupies the first *K* rows/columns of array *A*
L	Integer	input:	dimension of *A* as defined in a *DIMENSION* statement in the calling program
LO	Integer	input:	row/column number to be left out, if any: ($\|LO\|$ must be $\leqslant K$) *LO.EQ.*0: invert the input matrix *LO.GT.*0: invert the input matrix ignoring row/column number *LO* *LO.LT.*0: "re-invert" the input matrix for row/column number $-LO$ only (point 3 under Purpose)
PVT	Real	output:	value of the smallest pivot encountered
IFAULT	Integer	output:	fault indicator: *IFAULT* = 0 normal return *IFAULT* = 1 small pivot encountered *IFAULT* = 2 nil pivot encountered *IFAULT* = 3 *K* or *LO* out of range

DATA constants

BIG	Real	data:	used to determine the minimum pivot, usually set to the largest convenient number acceptable to the computer
SMALL	Real	data:	pivot value under which rounding errors will significantly affect the accuracy of the results; this will depend on the numbers involved and on the arithmetical precision of the machine

Calculations use and affect only the upper triangle of $A(I,J)$, i.e. when $J.GE.I$. Note that inserting $A(J,I) = A(I,J)$ between the lines labelled 13 and 14 will fill the lower triangle correctly only when $LO.GE.0$.

Note also that, when $LO \neq 0$, row and column $|LO|$ are not physically eliminated from A but simply ignored in the same manner as the $L-K$ excess rows and columns of A when $K<L$.

Time

If K is the order of the original matrix, the direct computation of $(\mathbf{W}'\mathbf{W})^{-1}$ from $(\mathbf{X}'\mathbf{X})^{-1}$ for a particular row/column will proceed approximately K times faster than inversion of the original matrix ignoring the same row/column when both operations are performed by *SINV*. The gain may approach K^2 when compared to a general inversion routine.

Storage

Operation on the upper triangle allows for storage of all but the diagonal elements of the complete inverse (or of the original matrix) in the lower triangle if memory must be husbanded.

Precision

Users of 32-bit-word computers should specify double-precision arithmetic. The *REAL* declarations should be changed to *DOUBLE PRECISION*; the *DATA* statement modified; and *ABS* replaced by *DABS* in two statements.

Accuracy

Tests on a CDC Cyber 7326 (60-bit words, ≈ 14 significant digits in single precision) calculating the product of the inverse by the original matrix showed that rounding errors are of the same order as when proceeding conventionally (deviations from identity matrix elements less than 10^{-13} with seventh order, $]0, 1[$ pseudo-random matrices).

Acknowledgement

The author wishes to thank the algorithms editor and a referee for useful suggestions.

References

Garside, M. J. (1971). Algorithm AS 37. Inversion of a symmetric matrix. *Appl. Statist.*, **20**, 111–112.
Knight, W. (1974). Remark AS R9. A remark on Algorithm AS 37. *Appl. Statist.*, **23**, 100–101.

```
      SUBROUTINE SINV(A, K, L, LO, PVT, IFAULT)
C
C        ALGORITHM AS 141  APPL. STATIST. (1979) VOL.28, NO.2
C
C        CALCULATE THE INVERSE OF A SYMMETRIC MATRIX
C        IGNORING A SPECIFIED ROW/COLUMN IF LO .NE. 0,
C        USING EITHER THE ORIGINAL MATRIX OR A COMPLETE INVERSE OF IT.
C
      DIMENSION A(L, L)
      REAL A, AA, AIP, BIG, EP, PVT, SMALL, T
      INTEGER P, PM, PP
C
      DATA BIG, SMALL /1.0E70, 1.0E-7/
C
C        PARAMETER CHECKS
C
      IFAULT = 3
```

```
      IF (IABS(LO) .GT. K .OR. K .LT. 1 .OR. K .GT.
C
C        INITIAL VALUES
C
      IFAULT = 0
      IF (LO .GE. 0) GOTO 1
      EP = 1.0
      P = -LO
      PVT = ABS(A(P, P))
      T = PVT
      GOTO 3
    1 EP = -1.0
      P = 1
      PVT = BIG
C
C        PIVOT BY PIVOT INVERSION
C
    2 IF (P .EQ. LO) GOTO 12
      T = ABS(A(P, P))
      IF (T .LT. PVT) PVT = T
    3 IF (T .LT. SMALL) IFAULT = 1
      IF (T .EQ. 0.0) GOTO 15
      PM = P - 1
      PP = P + 1
      AA = 1.0 / A(P, P)
      A(P, P) = -AA
      IF (P .EQ. 1) GOTO 8
      DO 7 I = 1, PM
      AIP = A(I, P) * AA
      DO 4 J = I, PM
    4 A(I, J) = A(I, J) - AIP * A(J, P)
      IF (P .EQ. K) GOTO 6
      DO 5 J = PP, K
    5 A(I, J) = A(I, J) - AIP * A(P, J)
    6 A(I, P) = AIP * EP
    7 CONTINUE
    8 IF (P .EQ. K) GOTO 11
      DO 10 I = PP, K
      AIP = A(P, I) * AA
      DO 9 J = I, K
    9 A(I, J) = A(I, J) - AIP * A(P, J)
      A(P, I) = AIP * EP
   10 CONTINUE
   11 IF (EP .GT. 0.0) RETURN
   12 P = P + 1
      IF (P .LE. K) GOTO 2
C
C        SIGN CORRECTION
C
      DO 14 I = 1, K
      DO 13 J = I, K
   13 A(I, J) = -A(I, J)
   14 CONTINUE
      RETURN
C
C        NIL PIVOT EXIT
C
   15 IFAULT = 2
      RETURN
      END
```