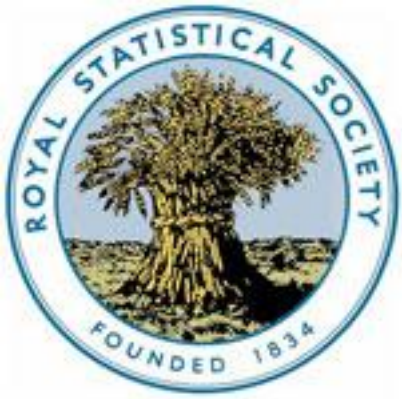

Algorithm AS 140: Clustering the Nodes of a Directed Graph
Author(s): Gary W. Oehler
Source: *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, Vol. 28, No. 2 (1979), pp. 206-214
Published by: Wiley for the Royal Statistical Society
Stable URL: http://www.jstor.org/stable/2346750
Accessed: 26-06-2016 02:03 UTC

```
C        UNPACK THE MATRIX
C
      CALL UNPACK(VCOV, MPLONE, LENWRK)
      RETURN
      END
C
      SUBROUTINE UNPACK(X, N, LENX)
C
C        ALGORITHM AS 139.1  APPL. STATIST. (1979) VOL.28, NO.2
C
C        THIS SUBROUTINE EXPANDS A SYMMETRIC MATRIX STORED IN LOWER
C        TRIANGULAR FORM IN THE FIRST N*(N+1)/2 POSITIONS OF X
C        INTO A MATRIX USING THE FIRST N*N POSITIONS
C
C        LENX = THE LENGTH OF VECTOR X - MUST BE NOT LESS THAN N*N
C
      DIMENSION X(LENX)
      NSQ = N * N
      II = NSQ
      JJ = N * (N + 1) / 2
C
C        STORE LAST ROW
C
      DO 10 I = 1, N
      X(II) = X(JJ)
      II = II - 1
      JJ = JJ - 1
   10 CONTINUE
      DO 80 I = 2, N
C
C        OBTAIN UPPER PART OF MATRIX FROM PART ALREADY SHIFTED
C
      IJ = I - 1
      KK = NSQ + 1 - I
      DO 50 J = 1, IJ
      X(II) = X(KK)
      II = II - 1
      KK = KK - N
   50 CONTINUE
C
C        OBTAIN LOWER PART OF MATRIX FROM
C        ORIGINAL TRIANGULAR STORAGE
C
      IJ = N - IJ
      DO 70 J = 1, IJ
      X(II) = X(JJ)
      II = II - 1
      JJ = JJ - 1
   70 CONTINUE
   80 CONTINUE
      RETURN
      END
```

Algorithm AS 140

Clustering the Nodes of a Directed Graph

By GARY W. OEHLERT†

Yale University

Keywords: CLUSTERING; DIRECTED GRAPH; MAXIMUM LIKELIHOOD; TRANSFER ALGORITHM

LANGUAGE

ISO Fortran

† Development of this algorithm was partially supported by National Science Foundation Grant DCR75–08374.

Description and Purpose

The nodes of a directed graph are to be partitioned into clusters so that for any ordered pair of clusters, the nodes of the first cluster are nearly all linked to the nodes of the second cluster, or nearly all not linked. In this way, a summary is made of the linkages of the graph. Busacker and Saaty (1965) has a discussion of directed graphs and a variety of their applications.

The criterion chosen to evaluate the partition is the likelihood function. Consider the $N \times N$ association matrix B of the N nodes of the graph. If there are $NCLUS$ clusters, then B may be thought of as being partitioned into $NCLUS \times NCLUS$ cells corresponding to the partition of the nodes; cell (a, b), for clusters a and b, has $S(a, b)$ links and $Z(a, b)$ entries. Assume that each binary entry of B is an independent Bernoulli trial with $\Pr\{B(I, J) = 1\}$ equal to $P(a, b)$, where node I is in cluster a and node J is in cluster b. Then the maximum likelihood estimate of $P(a, b)$ is

$$\hat{P}(a, b) = S(a, b)/Z(a, b),$$

and the overall likelihood is equal to

$$\prod_{1 \leq a, b \leq NCLUS} \hat{P}(a, b)^{S(a,b)}(1 - \hat{P}(a, b))^{(Z(a,b)-S(a,b))}$$

(The diagonal elements of B are not used in the likelihood calculations.) Hartigan (1975) suggests the use of maximum likelihood in non-hierarchical clustering; the related Shannon–Wiener information measure (or approximations to it) has been used for quite some time in hierarchical clustering. See Williams and Lambert (1959) and MacNaughton-Smith (1965).

The algorithm is a "transfer" algorithm; an initial partition is produced by subroutine *INIT*, and subroutine *ALLOC* transfers each node in turn to the cluster for which the overall likelihood is maximized. Swaps of pairs of nodes between clusters, as in Banfield and Bassill (1977), are not considered. This algorithm further differs from standard transfer algorithms (for example, Banfield and Bassill) by selecting more judiciously the transfers to be tested, by handling the simultaneous clustering of the rows and columns of the association matrix, and by handling large sparse linkage matrices in list form.

Method

The association matrix B is not used in its $N \times N$ form, but in the form of a vector **X**. The first N elements of **X** are a directory, and the remaining elements are the links in the graph. The numbers of the nodes to which node I is linked are stored in **X** beginning at location **X**(I). Entries in **X** must be packed to the left so that the last node to which node I is linked is immediately followed in **X** by the list of nodes for node $I+1$.

Subroutine *INIT* constructs an initial partition of the nodes. If $ITYPE = 0$, the partition is constructed by approximating the first eigenvector of B (using eight iterations of the power method) and sorting the nodes so that their components in the eigenvector are in ascending order. The sorted nodes are then split into $NCLUS$ initial clusters. If $ITYPE = 1$, a user supplied initial partition, specified in *CLUS*, is used.

Subroutine *ALLOC* transfers a node from one cluster to another if the move increases the log likelihood by more than *TH*, and if the second cluster is smaller than *MXSIZE*, a given maximum size for clusters. *ALLOC* makes global passes, in which all transfers are checked, and local passes, in which transfers between clusters that did not change in the previous pass are not checked. A local pass follows any pass in which a transfer was executed; a global pass follows a local pass which had no transfers. *ALLOC* begins with a global pass and continues until there is a global pass with no transfers, at which time a local optimum has been reached. This technique reduces computation by ignoring unlikely transfers except when testing for a local optimum. On return, the output structures of *ALLOC* contain the final partition in

CLUS, the size of each cluster in *SIZE*, the matrix of success counts in *P* and the overall log likelihood in *R*1(1).

Structure

SUBROUTINE INIT(X, N, XLEN, NCLUS, MXCLUS, MXSIZE, ITYPE, CLUS, Y, SIZE, P, IOLD, INEW, IFAULT)

Formal parameters

X	Integer array (*XLEN*)	input: data vector
N	Integer	input: number of nodes
XLEN	Integer	input: length of *X*
NCLUS	Integer	input: number of clusters requested
MXCLUS	Integer	input: maximum number of clusters
MXSIZE	Integer	input: maximum size of a cluster
ITYPE	Integer	input: type of initialization
CLUS	Integer array (*N*)	input/ output: initial cluster of each node
Y	Integer array (*XLEN*)	output: data vector (by columns of *B*)
SIZE	Integer array (*MXCLUS*)	output: initial size of each cluster
P	Real array (*MXCLUS, MXCLUS*)	output: initial matrix of success counts
IOLD	Integer array (*N*)	workspace:
INEW	Integer array (*N*)	workspace:
IFAULT	Integer	output: fault indicator

Fault indicator

IFAULT = 0 no fault
1 *NCLUS* outside its range
2 $MXSIZE * NCLUS \leqslant N$
3 error in *X* directory: $X(i) \leqslant X(i-1)$, some $i \leqslant N$; $X(1) \neq N+1$; or $X(N) > XLEN$
4 reference in *X* to node number < 1 or $> N$
5 same link listed twice in *X*
6 all-zero column of the association matrix *B*
7 illegal cluster specified in initialization
8 initial cluster violates size restrictions

SUBROUTINE ALLOC(X, Y, N, XLEN, NCLUS, MXCLUS, MXSIZE, TH, MXS2, CLUS, SIZE, P, R1, R2, TLOG, IOLD, INEW)

Formal parameters

These are the same as for *INIT* except

Y	Integer array (*XLEN*)	input: data vector (by columns of *B*)
TH	Real	input: threshold of incremental likelihood
MXS2	Integer	input: equal to *MXSIZE* squared
CLUS	Integer array (*N*)	input/ output: current cluster of each node
SIZE	Integer array (*MXCLUS*)	input/ output: current size of each cluster
P	Real array (*MXCLUS, MXCLUS*)	input/ output: current matrix of success counts

*R*1	Real array (*MXCLUS*)	workspace/ output: *R*1(1) returns the overall log likelihood
*R*2	Real array (*MXCLUS*)	workspace:
TLOG	Real array (*MXS*2)	workspace:

Auxiliary Algorithms

ALLOC calls the real function *XLIKE*(*P*1, *R*1, *S*1, *S*2, *S*3, *S*4, *Y*1, *TLOG*, *MXS*2), which is included. *XLIKE* calculates the change in log likelihood between *P*1 successes in $S1 * S2$ trials and $P1 - Y1 * R1$ successes in $S3 * S4$ trials.

Restrictions

1. $1 < NCLUS \leqslant MXCLUS$; $NCLUS < N$.
2. $NCLUS * MXSIZE > N$.
3. The *X* vector must be such that every row and column of the association matrix *B* has a non-zero element. The easiest way to assure this is to set the diagonal elements of *B* equal to 1; this will not affect the clusters. Faults of types 3 and 6 can indicate an all-zero row or column of *B*.
4. No link may be listed twice in *X*.
5. The algorithm does not accept missing values; that is, the information in *X* is taken to be a complete description of the linkages of the graph.
6. Cluster sizes must be greater than zero and less than *MXSIZE*.

Accuracy

The final partition produced is a local optimum and may not be a global optimum. Also, there may be other partitions which result in the same log likelihood. Use of several initial partitions will reduce the risk of selecting a poor local optimum.

The accuracy of the change in log likelihood and overall log likelihood calculations depends on the accuracy of the real arithmetic on the particular computer in use. Each change in log likelihood calculation requires $20 * NCLUS$ real additions, and the overall log likelihood calculation requires $3 * NCLUS^2$ real additions.

Storage and Time

The variable storage requirement for this algorithm is less than $2XLEN + 3N + 3MXCLUS + MXCLUS^2 + MXSIZE^2 + 100$ words.

The amount of time required for convergence depends on *N*, *XLEN*, *NCLUS* and the configuration of the graph itself, but computation is roughly proportional to $N * NCLUS^2$. A sample graph of 71 nodes and 243 links executed in 20·6 sec for 15 clusters on an IBM 370/158.

Acknowledgement

The author is indebted to J. A. Hartigan and the referee for their helpful suggestions.

References

Banfield, C. F. and Bassill, L. C. (1977). Algorithm AS 113. A transfer algorithm for non-hierarchical classification. *Appl. Statist.*, **26**, 206–210.

Busacker, R. G. and Saaty, T. L. (1965). *Finite Graphs and Networks: An Introduction with Applications.* New York: McGraw-Hill.

Hartigan, J. A. (1975). *Clustering Algorithms.* New York: Wiley.

MacNaughton-Smith, P. (1965). *Some Statistical and Other Numerical Techniques for Classifying Individuals.* Home Office Res. Unit Rep., Publ. No. 6. London: HMSO.

Williams, W. T. and Lambert, J. M. (1959). Multivariate methods in plant ecology. I. Association analysis in plant communities. *J. Ecol.*, **47**, 83–101.

```
      SUBROUTINE INIT(X, N, XLEN, NCLUS, MXCLUS, MXSIZE, ITYPE,
     *  CLUS, Y, SIZE, P, IOLD, INEW, IFAULT)
C
C       ALGORITHM AS 140.1  APPL. STATIST. (1979) VOL.28, NO.2
C
C       CONSTRUCT AN INITIAL PARTITION OF THE NODES
C
      INTEGER XLEN, FIRST, X(XLEN), Y(XLEN), CLUS(N), SIZE(MXCLUS),
     *  INEW(N), IOLD(N)
      REAL P(MXCLUS, MXCLUS)
      LOGICAL FLAG
C
C       CHECK INPUTS
C
      IFAULT = 1
      IF (NCLUS .LE. 1 .OR. NCLUS .GE. N .OR. NCLUS .GT. MXCLUS) RETURN
      IFAULT = 2
      IF (NCLUS * MXSIZE .LE. N) RETURN
      IFAULT = 3
      DO 1 I = 2, N
      L = I - 1
      IF (X(I) .LE. X(L)) RETURN
    1 CONTINUE
      M = N + 1
      IF (X(1) .NE. M .OR. X(N) .GT. XLEN) RETURN
      IFAULT = 4
      DO 2 I = M, XLEN
      IF (X(I) .LE. 0 .OR. X(I) .GT. N) RETURN
    2 CONTINUE
      IFAULT = 5
      DO 5 I = 1, N
      FIRST = X(I)
      L = I + 1
      LAST = X(L) - 1
      IF (I .EQ. N) LAST = XLEN
      IF (FIRST .EQ. LAST) GOTO 5
      JLAST = LAST - 1
      DO 4 J = FIRST, JLAST
      JTEST = X(J)
      KFIRST = J + 1
      DO 3 K = KFIRST, LAST
      IF (JTEST .EQ. X(K)) RETURN
    3 CONTINUE
    4 CONTINUE
    5 CONTINUE
C
C       CONSTRUCT Y VECTOR.  THE Y VECTOR HOLDS THE ASSOCIATION
C       MATRIX BY COLUMNS RATHER THAN BY ROWS AS X DOES.
C
      DO 6 I = 1, N
      INEW(I) = 0
      Y(I) = N + 1
    6 CONTINUE
      Y(M) = 0
      DO 7 I = 1, N
      FIRST = X(I)
      LAST = X(I + 1) - 1
      IF (I .EQ. N) LAST = XLEN
      DO 7 J = FIRST, LAST
      KFIRST = X(J) + 1
      DO 7 K = KFIRST, N
      Y(K) = Y(K) + 1
    7 CONTINUE
      DO 8 I = 1, N
      FIRST = X(I)
      LAST = X(I + 1) - 1
      IF (I .EQ. N) LAST = XLEN
      DO 8 J = FIRST, LAST
      ITEMP = X(J)
      L = Y(ITEMP) + INEW(ITEMP)
      Y(L) = I
      INEW(ITEMP) = INEW(ITEMP) + 1
```

```
    8 CONTINUE
      IFAULT = 6
      DO 9 I = 2, N
      IF (Y(I) .LE. Y(I - 1)) RETURN
    9 CONTINUE
      IF (Y(1) .NE. M .OR. Y(N) .GT. XLEN) RETURN
      IF (ITYPE .EQ. 1) GOTO 33
      IFAULT = 0
C
C        APPROXIMATE FIRST EIGENVECTOR
C
      DO 11 I = 1, N
   11 IOLD(I) = I
      DO 13 ITER = 1, 8
      DO 12 I = 1, N
      LAST = X(I + 1) - 1
      IF (I .EQ. N) LAST = XLEN
      DO 12 J = FIRST, LAST
      ITEMP = X(J)
      INEW(I) = INEW(I) + IOLD(ITEMP)
   12 CONTINUE
      DO 13 I = 1, N
      IOLD(I) = INEW(I)
   13 CONTINUE
C
C        SORT BY FIRST EIGENVECTOR
C
      DO 14 I = 1, N
   14 INEW(I) = I
      DO 16 I = 1, N
      FLAG = .TRUE.
      DO 15 J = 2, N
      L = J - 1
      ITEMP = INEW(L)
      M = INEW(J)
      IF (IOLD(M) .GE. IOLD(ITEMP)) GOTO 15
      FLAG = .FALSE.
      INEW(J) = INEW(L)
      INEW(L) = M
   15 CONTINUE
      IF (FLAG) GOTO 17
   16 CONTINUE
C
C        PARTITION INTO INITIAL CLUSTERS
C
   17 KSTART = N / NCLUS
      DO 21 I = 1, NCLUS
   21 SIZE(I) = KSTART
      ILAST = MOD(N, NCLUS)
      IF (ILAST .EQ. 0) GOTO 31
      DO 22 I = 1, ILAST
   22 SIZE(I) = SIZE(I) + 1
   31 J = 1
      DO 32 I = 1, NCLUS
      KLAST = SIZE(I)
      DO 32 K = 1, KLAST
      ITEMP = INEW(J)
      CLUS(ITEMP) = I
      J = J + 1
   32 CONTINUE
      GOTO 40
   33 DO 34 I = 1, NCLUS
   34 SIZE(I) = 0
      IFAULT = 7
      DO 35 I = 1, N
      J = CLUS(I)
      IF (J .LE. 0 .OR. J .GT. NCLUS) RETURN
      SIZE(J) = SIZE(J) + 1
   35 CONTINUE
      IFAULT = 8
      DO 36 I = 1, NCLUS
      IF (SIZE(I) .LE. 0 .OR. SIZE(I) .GT. MXSIZE) RETURN
```

```
   36 CONTINUE
      IFAULT = 0
C
C        SET UP P MATRIX, SUCCESS COUNTS
C
   40 DO 41 I = 1, NCLUS
      DO 41 J = 1, NCLUS
   41 P(I, J) = 0.0
      DO 42 I = 1, N
      FIRST = X(I)
      LAST = X(I + 1) - 1
      IF (I .EQ. N) LAST = XLEN
      DO 42 J = FIRST, LAST
      IF (X(J) .EQ. I) GOTO 42
      ITEMP = X(J)
      ITEMP = CLUS(ITEMP)
      ITEMP2 = CLUS(I)
      P(ITEMP2, ITEMP) = P(ITEMP2, ITEMP) + 1.0
   42 CONTINUE
      RETURN
      END
C
      SUBROUTINE ALLOC(X, Y, N, XLEN, NCLUS, MXCLUS, MXSIZE,
     *  TH, MXS2, CLUS, SIZE, P, R1, R2, TLOG, IOLD, INEW)
C
C        ALGORITHM AS 140.2  APPL. STATIST. (1979) VOL.28, NO.2
C
C        FROM AN INITIAL PARTITION OF THE NODES OF A GRAPH,
C        REALLOCATE NODES TO CLUSTERS TO FIND A LOCALLY
C        MAXIMUM LIKELIHOOD PARTITION
C
      INTEGER XLEN, FIRST, X(XLEN), Y(XLEN), CLUS(N),
     *  SIZE(MXCLUS), IOLD(N), INEW(N)
      REAL P(MXCLUS, MXCLUS), TLOG(MXS2), R1(MXCLUS), R2(MXCLUS)
      LOGICAL GLOBAL
C
C        INITIALIZE TLOG, INEW, IOLD, PASS.
C
      DO 1 I = 1, MXS2
    1 TLOG(I) = FLOAT(I) * ALOG(FLOAT(I))
    2 DO 3 I = 1, NCLUS
      INEW(I) = 0
      IOLD(I) = 1
    3 CONTINUE
      GLOBAL = .TRUE.
C
C        MOVE A NODE TO A NEW CLUSTER IF THE MOVE INCREASES
C        THE LIKELIHOOD.
C        ONLY CHECK MOVES IF ONE OF THE CLUSTERS HAS IOLD = 1.
C
    4 DO 59 ITER = 1, N
      NBEST = CLUS(ITER)
      BTEST = TH
C
C        SET UP ARRAY OF ASSOCIATIONS (R) FOR THIS NODE
C
      DO 10 I = 1, NCLUS
      R1(I) = 0.0
      R2(I) = 0.0
   10 CONTINUE
      FIRST = X(ITER)
      L = ITER + 1
      LAST = X(L) - 1
      IF (ITER .EQ. N) LAST = XLEN
      DO 11 I = FIRST, LAST
      IF (X(I) .EQ. ITER) GOTO 11
      ITEMP = X(I)
      ITEMP = CLUS(ITEMP)
      R1(ITEMP) = R1(ITEMP) + 1.0
   11 CONTINUE
      FIRST = Y(ITER)
      LAST = Y(L) - 1
```

```
      IF (ITER .EQ. N) LAST = XLEN
      DO 12 I = FIRST, LAST
      IF (Y(I) .EQ. ITER) GOTO 12
      ITEMP = Y(I)
      ITEMP = CLUS(ITEMP)
      R2(ITEMP) = R2(ITEMP) + 1.0
   12 CONTINUE
C
C        CHECK EACH CLUSTER FOR AN INCREASE IN LIKELIHOOD
C
      L = CLUS(ITER)
      DO 49 M = 1, NCLUS
      IF (L .EQ. M .OR. SIZE(M) .GE. MXSIZE .OR.
     *  IOLD(L) + IOLD(M) .EQ. 0) GOTO 49
      TEST = 0.0
      DO 20 J = 1, NCLUS
      IF (J .EQ. L .OR. J .EQ. M) GOTO 20
      IF (P(L, J) .GT. 0.0) TEST = TEST + XLIKE(P(L, J), R1(J), SIZE(L),
     *  SIZE(J), SIZE(L) - 1, SIZE(J), 1.0, TLOG, MXS2)
      IF (P(M, J) .GT. 0.0 .OR. R1(J) .GT. 0.0) TEST = TEST +
     *  XLIKE(P(M, J), R1(J), SIZE(M), SIZE(J), SIZE(M) + 1, SIZE(J),
     *  -1.0, TLOG, MXS2)
      IF (P(J, L) .GT. 0.0) TEST = TEST + XLIKE(P(J, L), R2(J), SIZE(L),
     *  SIZE(J), SIZE(L) - 1, SIZE(J), 1.0, TLOG, MXS2)
      IF (P(J, M) .GT. 0.0 .OR. R2(J) .GT. 0.0) TEST = TEST +
     *  XLIKE(P(J, M), R2(J), SIZE(M), SIZE(J), SIZE(M) + 1, SIZE(J),
     *  -1.0, TLOG, MXS2)
   20 CONTINUE
      TEST = TEST + XLIKE(P(L, L), R1(L) + R2(L) , SIZE(L) - 1, SIZE(L),
     *  SIZE(L) - 1, SIZE(L) - 2, 1.0, TLOG, MXS2) +
     *  XLIKE(P(L, M), R1(M) + R2(L) , SIZE(L), SIZE(M), SIZE(L) - 1,
     *  SIZE(M) + 1, 1.0, TLOG, MXS2) +
     *  XLIKE(P(M, L), R2(M) + R1(L) , SIZE(L), SIZE(M), SIZE(L) - 1,
     *  SIZE(M) + 1, -1.0, TLOG, MXS2) +
     *  XLIKE(P(M, M), R1(M) + R2(M) , SIZE(M) - 1, SIZE(M),
     *  SIZE(M) + 1, SIZE(M), -1.0, TLOG, MXS2)
      IF (TEST .LE. BTEST) GOTO 49
      BTEST = TEST
      NBEST = M
   49 CONTINUE
C
C        MOVE TO BEST CLUSTER
C
      IF (NBEST .EQ. L) GOTO 59
      M = NBEST
      DO 50 II = 1, NCLUS
      P(L, II) = P(L, II) - R1(II)
      P(M, II) = P(M, II) + R1(II)
      P(II, L) = P(II, L) - R2(II)
      P(II, M) = P(II, M) + R2(II)
   50 CONTINUE
      SIZE(L) = SIZE(L) - 1
      SIZE(M) = SIZE(M) + 1
      CLUS(ITER) = NBEST
      INEW(L) = 1
      INEW(M) = 1
   59 CONTINUE
C
C        CHECK FOR OPTIMUM, WERE THERE ANY MOVES THIS PASS.
C
      DO 60 I = 1, NCLUS
      IF (INEW(I) .GT. 0) GOTO 62
   60 CONTINUE
C
C        NO MOVES, IF A GLOBAL CHECK, FINISH.
C        IF A LOCAL CHECK, MAKE A GLOBAL CHECK.
C
      IF (GLOBAL) GOTO 70
      GOTO 2
C
C        SOME MOVES, RESET IOLD, INEW.  MAKE A LOCAL CHECK.
```

```
   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