CWI Syllabi

Managing Editors

J.W. de Bakker (CWI, Amsterdam)
M. Hazewinkel (CWI, Amsterdam)
J.K. Lenstra (CWI, Amsterdam)

Editorial Board

W. Albers (Eindhoven)
P.C. Baayen (Amsterdam)
R.J. Boute (Nijmegen)
E.M. de Jager (Amsterdam)
M.A. Kaashoek (Amsterdam)
M.S. Keane (Delft)
J.P.C. Kleijn (Tilburg)
H. Kwakernaak (Eindhoven)
J. van Leeuwen (Utrecht)
P.W.H. Lenneman (Utrecht)
M. van der Put (Groningen)
M. Rem (Eindhoven)
A.H.G. Rinnooy Kan (Rotterdam)
M.N. Spilker (Leiden)

Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

The CWI is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O).
STATAL: statistical procedures in Algol 60, part 2
R. van der Horst, R.D. Gill (eds.)
2. COMPUTATION OF STATISTICS

This section contains procedures for the computation of statistics. Most procedures also compute the (approximate) tail probability, under the null-hypothesis, of the statistic concerned, and thus these procedures can be used to perform statistical tests. (The approximate tail probabilities are only appropriate for large samples). The samples for which the statistics are computed are presented to the procedure in arrays, which must be declared in the main program. (Except the multivariate techniques COMONPLR, MINICOMA and JORESKOG; they read data using an auxillary procedure). Some of the procedures have a Boolean parameter SORTED to indicate whether the samples are already sorted in non-decreasing order or not. Sorted samples reduce the computing time of the procedures considerably. (Results do not make sense if it is indicated that the sample is sorted in non-decreasing order, while this is actually not the case!).

The correlation procedures PROMOCORRE, KENDALLS TAU and SPEARMANS RHO require at least three pairs of observations. In some cases it is possible that some results of these procedures are not defined or cannot be computed. In some cases no error message is given (in order to prevent loss of other output), but the value 4.444...4 is assigned.
Wilcoxon's W1

TITLE: Wilcoxon's W1

AUTHOR: R. Kaas

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION
For a sample of independent observations $x_1, x_2, \ldots, x_n$ the procedure computes Wilcoxon's one-sample or signed rank test statistic with respect to a value $\mu$ (i.e., the sum of the ranks of the positive differences $(x_i - \mu)$, where the ranking is over the absolute values of these differences), and an approximated two-sided tail probability under the hypothesis that the underlying distribution is symmetric about $\mu$. The test is mainly a location test.

KEYWORDS
Wilcoxon's one-sample test statistic,
Wilcoxon's signed rank test statistic

CALLING SEQUENCE

Heading
"REAL" "PROCEDURE" WILCOXON'S W1 (X, L, U, MU, NONZERO, P2, SORTED);
"VALUE" L, U, MU, SORTED;
"INTEGER" L, U, NONZERO;
"REAL" MU, P2;
"ARRAY" X;
"CODE" 42400;

Formal parameters
X: <array identifier>, vector containing the sample $x_1, x_2, \ldots, x_n$;
L: <integer arithmetic expression>, smallest index of the sample;
U: <integer arithmetic expression>, largest index of the sample;
MU: <arithmetic expression>, the tested centre of symmetry;
NONZERO: <integer variable>, output parameter, which at exit contains the number of non-zero differences $(x_i - \mu)$;
P2: <real variable>, output parameter, which at exit contains the value of the two-sided tail probability;
SORTED: <Boolean expression>, indicating whether the sample is sorted in non-decreasing order or not.

DATA AND RESULTS
The value of the signed rank test statistic is assigned to the procedure identifier WILCOXON W1, the number of non-zero differences to the output parameter NONZERO, and the approximated two-sided tail probability to the output parameter P2.
The following error message may appear:
Error number 2 (if $lx > ux - 1$)

**PROCEDURES USED**

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>VECSORT</td>
<td>STATA 11020</td>
</tr>
<tr>
<td>WILCOXONS W</td>
<td>STATA 40000</td>
</tr>
<tr>
<td>STATAL3 ERROR</td>
<td>STATAL 40100</td>
</tr>
<tr>
<td>PHI</td>
<td>STATAL 41500</td>
</tr>
</tbody>
</table>

**LANGUAGE**

Algol 60

**METHOD AND PERFORMANCE**

The computation of WILCOXONS W1 and NONZERO are straightforward, using midranks in case of ties. The two-sided tail probability is based on a normal approximation with continuity correction.

**REFERENCE**


**EXAMPLE OF USE**

**Program:**

```
"BEGIN"
"REAL" W1, PROB;
"INTEGER" NZ;
"REAL" "ARRAY" SAMPLE[1:10];
INARRAY(60, SAMPLE);
W1 := WILCOXONS W1(SAMPLE, 1, 10, 0, NZ, PROB, "FALSE");
OUTPUT(61, "(""WILCOXON'S W1 = "",+5ZD.6D,,
"NUMBER OF NONZERO'S = ")",+5ZD./,
"TAIL PROBABILITY = "",+5ZD.68")",
W1, NZ, PROB)
"END"
```
Input:
-16 87 -19 51 -77 -80 -49 38 -94 -75

Output:
WILCOXON'S W1 = +17.000000
NUMBER OF NONZERO'S = +10
TAIL PROBABILITY = +0.308063

Source Text

"CODE" 42400:
"REAL" "PROCEDURE" WILCOXONS W1(Z, LOW, UPP, MU,
NONZERO, P2, SORTED);
"VALUE" LOW, UPP, MU, SORTED; "REAL" MU, P2; "ARRAY" Z;
"INTEGER" NONZERO, LOW, UPP; "BOOLEAN" SORTED;
"BEGIN" "INTEGER" MINPOS, MAXNEG, W, M, N, I, MID, MLI;
"REAL" SIGMA, Z1, D;

"IF" UPP < LOW + 2 "THEN"
STATAL3 ERROR("("WILCOXONS W1")", 2, LOW);
"IF" "NOT" SORTED "THEN" VECQSORT(Z, LOW, UPP);
"IF" MU = 0 "THEN"
"FOR" I:= LOW "STEP" 1 "UNTIL" UPP "DO"
Z[I]:= Z[I] - MU;
"IF" Z[LOW] >= 0 "THEN"
"BEGIN" "IF" Z[UPP] = 0 "THEN"
"BEGIN" NONZERO:= 0; WILCOXONS W1:= 0;
               P2:= 1 "END" "ELSE"
               "BEGIN" "FOR" LOW:= LOW + 1 "WHILE" Z[LOW] = 0 "DO"
               NONZERO:= UPP - LOW + 1; P2:= .5 ** NONZERO;
               WILCOXONS W1:= NONZERO * (NONZERO + 1) / 2
               "END"
"END" "ELSE" "IF" Z[UPP] <= 0 "THEN"
"BEGIN"
"FOR" UPP:= UPP + 1, UPP - 1 "WHILE" Z[UPP] = 0 "DO"
NONZERO:= UPP - LOW + 1; P2:= .5 ** NONZERO;
WILCOXONS W1:= 0
"END" "ELSE"
"BEGIN" MAXNEG:= LOW;
"FOR" MAXNEG:= MAXNEG + 1 "WHILE" Z[MAXNEG] < 0 "DO"
MINPOS:= MAXNEG; MAXNEG:= MAXNEG - 1;
"IF" Z[MINPOS] = 0 "THEN"
"FOR" MINPOS:= MINPOS + 1 "WHILE" Z[MINPOS] = 0 "DO"
M:= MAXNEG - LOW + 1; M:= UPP - MINPOS + 1;
NONZERO:= M + N;
MID:= (LOW + MAXNEG) // 2; MLI:= MAXNEG;
"FOR" I:= LOW "STEP" 1 "UNTIL" MID "DO"
"BEGIN" Z1:= Z[I]; Z[I]:= -Z[MLI]; Z[MLI]:= -Z1;
MLI:= MLI - 1;
"END";
W:= WILCOXONS W(Z, MINPOS, UPP, Z, LOW, MAXNEG,
"TRUE", D) + M * (M + 1);
SIGMA := NONZERO * (NONZERO + 1) * 
(2 * NONZERO + 1) / 6 - (D - NONZERO) / 12;
"IF" SIGMA <= 0 "THEN" P2 := 1 "ELSE"
P2 := 2 * PHI((-ABS(W - NONZERO * (NONZERO + 1)) / 
2 + 1) / SQRT(SIGMA));
"WILCOXONS W1 := W / 2"
"END"
"END" WILCOXONS W1;
"EOP"
Kolmogorovs T

TITLE: Kolmogorovs T

AUTHOR: E. Opperdoes

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION
For a sample of independent observations \( x[I:J] \), the procedure computes Kolmogorov's test statistic \( T \) for testing the hypothesis that the sample is from a specified continuous distribution. Furthermore, an approximated right tail probability of the test statistic under the hypothesis is given.

KEYWORDS
Kolmogorov's goodness of fit test statistic \( T \)

CALLING SEQUENCE

Heading
"REAL" "PROCEDURE" KOLMOGOROV5 T (X, LX, UX, Y, CDF Y, PR, SORTED);
"VALUE" LX, UX, SORTED;
"INTEGER" LX, UX;
"REAL" Y, CDF Y, PR;
"ARRAY" X;
"BOOLEAN" SORTED;
"CODE" 42405;

Formal parameters

\( X: \) <array identifier>, vector containing the sample \( x[I:J] \);

\( LX: \) <integer arithmetic expression>, smallest index of the sample;

\( UX: \) <integer arithmetic expression>, largest index of the sample;

\( Y: \) <real variable>, Jensen parameter, argument of the null-hypothesis distribution function;

\( CDF Y: \) <arithmetic expression>, null-hypothesis distribution function with the Jensen parameter \( Y \) as argument;

\( PR: \) <real variable>, output parameter, which at exit contains the approximate value of the right tail probability of the test statistic under the hypothesis;

\( SORTED: \) <Boolean expression>, indicating whether the sample is sorted in non-decreasing order or not.

DATA AND RESULTS
The value of Kolmogorov's test statistic \( T \) is assigned to the procedure identifier KOLMOGOROV5 T, and the approximate value of the right tail probability to the output parameter \( PR \). The following error message may appear: Errornumber 2 (if \( LX > UX - 1 \))
PROCEDURES USED
VECGSORT STATAL 11020
STATAL3 ERROR STATAL 40100

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
Kolmogorov's test statistic for the two-sided test for goodness of fit of a distribution with distribution function \( F_0 \) is the maximum of \( \text{ABS}(F_E(x[i]) - F_0(x[i])) \), for \( i = 1, \ldots, n \), where \( F_E \) is the empirical distribution function of the sample. The approximated right tail probability is computed using a formula in Smirnov (1948).

REFERENCES

EXAMPLE OF USE
Program:

"BEGIN"
"REAL" T, PROB, Y;
"REAL" "ARRAY" SAMPLE[1:10];
INARRAY(60, SAMPLE);
T := KOLMOGOROVS (SAMPLE, 1, 10, Y, NORMAL(Y, 50, 25), PROB, "FALSE");
OUTPUT(61, """"(""KOLMOGOROV'S T = ")",+52D.60,/, ""(""TAIL PROBABILITY = ")",+52D.60", T, PROB"
"END"

Input:

89 48 13 11 75 42 78 5 17 12
\textbf{Output:}

\begin{verbatim}
KOLMOGOROV'S T = +0.406582
TAIL PROBABILITY = +0.073308
\end{verbatim}

\textbf{Source text:}

```
"CODE" 42405;
"REAL" "PROCEDURE" KOLMOGOROV'S T (OBS, L, U, X, CDF X, P2,
   SORTED);
"VALUE" L, U, SORTED; "ARRAY" OBS; "BOOLEAN" SORTED;
"INTEGER" L, U; "REAL" X, CDF X, P2;
"BEGIN" "INTEGER" I, NOBS, SUM; "REAL" D, F, S, NEXT, LAST;
"PROCEDURE" MAX (A, B); "VALUE" B; "REAL" A, B;
"IF" A < B "THEN" A := B;

"REAL" "PROCEDURE" KOLSMYRAS (X); "VALUE" X; "REAL" X;
"BEGIN" "INTEGER" K; "REAL" TERM, SUM, EPS;
   EPS := "-6 / 2; SUM := 0; X := -2 * 1 * X;
   "FOR" K := 1 , K < 2 "WHILE" TERM := EPS "DO"
   "BEGIN" TERM := EXP(X * K * K) * 
      (1 - EXP(X * (K < 2 + 1)));
   SUM := SUM + TERM
   "END";
KOLSMYRAS := 2 * SUM
"END" KOLSMYRAS;

"IF" L > U - 2
"THEN" STATAL3 ERROR ("("KOLMOGOROV'S T")", 2, L);
"IF" "NOT" SORTED "THEN" VEC QSORT(OBS, L, U);
NOBS := U - L + 1; SUM := 0; X := LAST := OBS [L];
D := F := CDF X;
"FOR" I := L + 1 "STEP" 1 "UNTIL" U "DO"
"BEGIN" NEXT := OBS [I]; SUM := SUM + 1;
   "IF" NEXT > LAST "THEN"
   "BEGIN" S := SUM / NOBS; X := LAST := NEXT;
   MAX (D, ABS (F - S)); F := CDF X;
   MAX (D, ABS (F - S))
   "END"

"END";
MAX (D, 1 - F); KOLMOGOROV'S T := D;
P2 := KOLSMYRAS (D * SQRT (NOBS))
"END" KOLMOGOROV'S T;
"EOP"
```
2.1.1.3

Cramer von Mises W1

AUTHOR: E. Opperdoes

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION
For a sample of independent observations \( x_1, x_2, \ldots, x_n \), the procedure computes Cramer von Mises' test statistic \( W_1 \) for testing the hypothesis that the sample is from a specified continuous distribution function. Furthermore, an approximated right tail probability of the test statistic under the hypothesis is given.

KEYWORDS
Cramer von Mises' goodness of fit test statistic \( W_1 \)

CALLING SEQUENCE

Heading
"REAL" "PROCEDURE" CRAMER VON MISES W1 (X, LX, UX, Y, CDF Y, PR, SORTED);
"VALUE" LX, UX, SORTED;
"INTEGER" LX, UX;
"REAL" Y, CDF Y, PR;
"ARRAY" X;
"BOOLEAN" SORTED;
"CODE" 42407;

Formal parameters

\( X: \)  \(<\text{array identifier}>\), vector containing the sample \( x_1, x_2, \ldots, x_n \);
\( LX: \)  \(<\text{integer arithmetic expression}>\), smallest index of the sample;
\( UX: \)  \(<\text{integer arithmetic expression}>\), largest index of the sample;
\( Y: \)  \(<\text{real variable}>\), Jensen parameter, argument of the null-hypothesis distribution function;
\( CDF \ Y: \)  \(<\text{arithmetic expression}>\), null-hypothesis distribution function with the Jensen parameter \( Y \) as argument;
\( PR: \)  \(<\text{real variable}>\), output parameter, which at exit contains the approximate value of the right tail probability of the test statistic under the hypothesis;
\( SORTED: \)  \(<\text{Boolean expression}>\), indicating whether the sample is sorted in non-decreasing order or not.

DATA AND RESULTS
The value of Cramer von Mises' test statistic \( W_1 \) is assigned to the procedure identifier CRAMER VON MISES W1, and the approximate value of the right tail probability to the output parameter \( PR \).
The following error message may appear:
Cramer von Mises W1

Error number 2 (if $LX \geq UX - 1$)

PROCEDURES USED

LIMIT
VECQSORT
STATAL3 ERROR
BESS KA01

STATA LIMIT
STATA 11020
STATA 40100
NUMAL 35191

LANGUAGE
Algol 60

METHOD AND PERFORMANCE

Cramer von Mises' test statistic for the two-sided test for goodness of fit of a distribution with distribution function $f_0$ equals

$$\frac{1}{12N} \sum_{i=1}^{N} (f_0(X(i)) - (1 - \frac{1}{2})/N),$$

where $f_0$ is the null-hypothesis distribution function, $X(1), \ldots, X(N)$ the ordered sample, and $N$ the sample size. The approximated value of the right tail probability is computed using a formula in Anderson and Darling (1952).

REFERENCE


EXAMPLE OF USE

Program:

"BEGIN"
"REAL" W1, PROB, Y;
"REAL" "ARRAY" SAMPLE[1:10];
INARRAY(60, SAMPLE);
W1:= Cramer von Mises W1(SAMPLE, 1, 10, Y, NORMAL(Y, 50, 25),
PROB, "FALSE");
OUTPUT(61, "(""Cramer von Mises' W1 = ")", +520.60,/, 
"(""TAIL PROBABILITY = "")", +520.60")", W1, PROB)
"END"

210
Input:
66 35 15 52 12 81 67 41 47 99

Output:
CRAMER VON MISES' W1 = +0.024683
TAIL PROBABILITY = +0.990231

Source text:
"CODE" 42407;
"REAL" "PROCEDURE" CRAMER VON MISES W1(OBS, L, U, X,
CDF X, P2, SORTED);
"VALUE" L, U, SORTED; "ARRAY" OBS; "INTEGER" L, U;
"BOOLEAN" SORTED; "REAL" X, CDF X, P2;
"BEGIN" "INTEGER" I, N2; "REAL" T;

"IF" L > U-2
"THEN" STATA13 ERROR ("("CRAMER VON MISES W1")", 2, L);
"IF" "NOT" SORTED "THEN" VEC QSORT(OBS, L, U);
N2:= 2 * (U - L + 1); I:= -1; T:= 1 / 6 / N2;
"FOR" L:= L "STEP" 1 "UNTIL" U "DO"
"BEGIN" I:= I + 2; X:= OBS [L];
T:= T + (CDF X - I / N2)**2
"END";
CRAMER VON MISES W1:= T; P2:= 1 - LIMIT (T)
"END" CRAMER VON MISES W1;
"EOP"
TITLE: Anderson Darling

AUTHOR: A. Nonymous

INSTITUTE: Mathematical Centre

RECEIVED: 1981/1982

BRIEF DESCRIPTION
For a sample of independent observations \( x[1], \ldots, x[n] \), the procedure computes Anderson Darling's test statistic for testing the hypothesis that the sample is from a normal distribution (with unknown mean and variance).

KEYWORDS
Anderson Darling's goodness-of-fit test statistic

CALLING SEQUENCE

Heading
"REAL" "PROCEDURE" ANDERSON DARLING (X, LX, UX, SORTED);
"VALUE" LX, UX, SORTED;
"INTEGER" LX, UX;
"ARRAY" X;
"BOOLEAN" SORTED;
"CODE" 49999;

Formal parameters
X: \(<\text{array identifier}>\), vector containing the sample \( x[1], \ldots, x[n] \);
LX: \(<\text{integer arithmetic expression}>\), smallest index of the sample;
UX: \(<\text{integer arithmetic expression}>\), largest index of the sample;
SORTED: \(<\text{Boolean expression}>\), indicating whether the sample is sorted in non-decreasing order or not.

DATA AND RESULTS
The value of Anderson Darling's test statistic is assigned to the procedure identifier ANDERSON DARLING.
The following error messages may appear:
Error number 1 (if all observations in the sample are equal)
Error number 3 (if \( LX > UX \))

PROCEDURES USED
VECSORT
STATAL 11020
STATAL3 ERROR
STATAL 40100
PHI
STATAL 41500
LANGUAGE
Algol 60

METHOD AND PERFORMANCE
Anderson Darling's test statistic for testing normality equals

\[-(1+4/N+25/N^2)(\sum_{i=1}^{N} \frac{(2i-1)(\ln(z_i)+\ln(1-z(N+1-i))/N-N),}\]

where \(N = UX - LX + 1, z_i = \frac{phi((x_i - MU)/SIGMA)}{, for i=1,...,N, and x(1),...,x(N) is the ordered sample, MU and SIGMA are estimations of the mean and the standard deviations of the sample.}

In order to perform a test, some critical values of the asymptotic distribution of the statistic under the null-hypothesis (normality) are given in the table below. (cf. Stephens, 1974).

<table>
<thead>
<tr>
<th>significant level</th>
<th>0.15</th>
<th>0.10</th>
<th>0.05</th>
<th>0.025</th>
<th>0.01</th>
</tr>
</thead>
<tbody>
<tr>
<td>critical value</td>
<td>0.576</td>
<td>0.656</td>
<td>0.787</td>
<td>0.918</td>
<td>1.092</td>
</tr>
</tbody>
</table>

REFERENCE


EXAMPLE OF USE

Program:

"BEGIN"
"ARRAY" X[1:10];
INARRAY(60, X);
OUTPUT(61, "(""(""""ANDERSON-DARLING TEST STATISTIC"")", "4D.60"")", ANDERSON DARLING(X, 1, 10, "TRUE")");
"END"

Input:

- .34  -.12  .01  .09  .16  .24  .30  .41  .54  .79
Output:

ANDERSON-DARLING TEST STATISTIC  0.107389

Source text:

"CODE" 49999;
"REAL" "PROCEDURE" ANDERSON DARLING(X, L, U, SORTED);
"VALUE" L, U, SORTED;
"ARRAY" X; "INTEGER" L, U; "BOOLEAN" SORTED;
"BEGIN" "INTEGER" I, N;
   "REAL" MU, SIGMA, XI, FACTOR, SUM, ESTIMATE;
   N:= U - L + 1;
   "IF" N <= 1 "THEN"
      STATA3 ERROR(""ANDERSON-DARLING"", 3, U);
   "IF" NOT SORTED "THEN" VECSORT(X, L, U);
   "IF" X[L] = X[U] "THEN"
      STATA3 ERROR(""ANDERSON-DARLING"", 1, X[L]);

   "COMMENT" FIRST THE ESTIMATION (IN THE USUAL WAY) OF
   EXPECTATION AND STANDARD DEVIATION OF THE NORMAL
   DISTRIBUTION. ;
   MU:= SIGMA:= 0; ESTIMATE:= (X[L] + X[U]) / 2;
   "FOR" I:= L "STEP" 1 "UNTIL" U "DO"
      "BEGIN" XI:= X[I]; ESTIMATE:= XI - ESTIMATE;
      MU:= MU + XI; SIGMA:= SIGMA + XI * XI;
   "END";
   MU:= MU / N;
   SIGMA:= SQRT((SIGMA - N * MU * MU) / (N - 1));

   "COMMENT" TRANSFORMATION OF THE OBSERVATIONS TO
   UNIFORM(0, 1)-DISTRIBUTED QUANTITIES. ;
   "FOR" I:= L "STEP" 1 "UNTIL" U "DO"
      X[I]:= PHI((X[I] - MU) / SIGMA);

   "COMMENT" ANDERSON-DARLING TEST QUANTITY;
   SUM:= 0; FACTOR:= -1;
   "FOR" I:= L "STEP" 1 "UNTIL" U "DO"
      "BEGIN" FACTOR:= FACTOR + 2;
      SUM:= SUM +
           FACTOR * (LN(X[I]) + LN(1 - X[L] + U - I));
   "END";
   ANDERSON DARLING:=
       (1 + 4 / N - 25 / N / N) * (-SUM / N - N)
   "END" OF ANDERSON DARLING;
   "EOP"
TITLE: Wilcoxon W2

AUTHOR: R. Kaas

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION
For two samples of independent observations \(x_{[LX]}, \ldots, x_{[UX]}\) and \(y_{[LY]}, \ldots, y_{[UY]}\), the procedure computes Wilcoxon's two-sample test statistic after shifting the first sample by a given distance \(SHIFT\) (i.e. the sum of the ranks of the first sample, where the ranking is over the pooled sample \(x_{[LX]-SHIFT}, \ldots, x_{[UX]-SHIFT}, y_{[LY]}, \ldots, y_{[UY]}\)). Furthermore, an approximate two-sided tail probability is computed under the hypothesis that the underlying distribution of the second sample is equal to the underlying distribution of the first, shifted over a given distance \(SHIFT\). The test is mainly a location test.

KEYWORDS
Wilcoxon's two-sample test statistic

CALLING SEQUENCE

Heading
"REAL" "PROCEDURE" WILCOXONS W2 (X, LX, UX, Y, LY, UY, SHIFT, P2, SORTED);
"VALUE" LX, UX, LY, UY, SHIFT, SORTED;
"INTEGER" LX, UX, LY, UY;
"REAL" SHIFT, P2;
"ARRAY" X, Y;
"BOOLEAN" SORTED;
"CODE" 42401;

Formal parameters

X: <array identifier>, vector containing the first sample \(x_{[LX]}, \ldots, x_{[UX]}\);

LX: <integer arithmetic expression>, smallest index of the first sample;

UX: <integer arithmetic expression>, largest index of the first sample;

Y: <array identifier>, vector containing the second sample \(y_{[LY]}, \ldots, y_{[UY]}\);

LY: <integer arithmetic expression>, smallest index of the second sample;

UY: <integer arithmetic expression>, largest index of the second sample;

SHIFT: <arithmetic expression>, shift applied to the first sample;

P2: <real variable>, output parameter, which at exit contains the value of the two-sided tail probability;

SORTED: <Boolean expression>, indicating whether the samples are sorted
in non-decreasing order or not.

DATA AND RESULTS
The value of the test statistic is assigned to the procedure identifier WILCOXON W2, and the value of the two-sided tail probability to the output parameter P2.
The following error messages may appear:
Error number 2  (if LX > UX - 1)
Error number 5  (if LY > UY - 1)

PROCEDURES USED
WILCOXON W                   STATAL 40000
STATAL3 ERROR                STATAL 40100
PHI                           STATAL 41500

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
The computation of Wilcoxon's W2 is straightforward. The two-sided tail probability is computed using a normal approximation with continuity correction.

REFERENCE

EXAMPLE OF USE
Program:
"BEGIN"
"REAL" W2, PROB;
"REAL" "ARRAY" SAMPLE1[1:8], SAMPLE2[1:12];
INARRAY(60, SAMPLE1); INARRAY(60, SAMPLE2);
W2 := WILCOXON W2(SAMPLE1, 1, 8, SAMPLE2, 1, 12, 0,
PROB, "FALSE");
OUTPUT(61, """"(""""WILCOXON'S W2 == ")"", +5ZD, ,
""""(""""TAIL PROBABILITY = ")"", +5ZD.6P")", W2, PROB)
"END"
2.1.2.1

Input:
84 48 14 39 6 38 74 16
69 98 54 55 85 64 87 11 55 80 45 48

Output:
WILCOXON'S W2 = +47
TAIL PROBABILITY = +0.063877

Source text
"CODE" 42401;
"INTEGER"""PROCEDURE" WILCOXONS W2(X, XLOW, XUPP, Y, YLOW,
YUPP, MU, PZ, SORTED);
"VALUE" XLOW, XUPP, YLOW, YUPP, MU, SORTED;
"ARRAY" X, Y;
"INTEGER" XLOW, XUPP, YLOW, YUPP;
"REAL" MU, P2;
"BOOLEAN" SORTED;
"BEGIN" "INTEGER" W, M, N, I, NTOT; "REAL" SIGMA, D;
"IF" XUPP < XLOW + 2 "THEN"
STATAL3 ERROR("("WILCOXONS W2")", 2, XLOW) "ELSE"
"IF" YUPP < YLOW + 2 "THEN"
STATAL3 ERROR("("WILCOXONS W2")", 5, YLOW);
"IF" MU = 0 "THEN"
"FOR" I:= XLOW "STEP" 1 "UNTIL" XUPP "DO"
XII:= XI + MU;
N:= XUPP - XLOW + 1; M:= YUPP - YLOW + 1; NTOT:= M + N;
WILCOXONS W2:= W:=
WILCOXONS W(X, XLOW, XUPP, Y, YLOW, YUPP, SORTED, D);
SIGMA:= M * N * (NTOT ** 3 - D) / 3 / NTOT / (NTOT - 1);
"IF" SIGMA <= 0 "THEN" P2:= 1 "ELSE"
P2:= 2 * PHI(-ABS(W - M * N + 1) / SQRT(SIGMA))
"END" WILCOXONS W2;
"EOP"
ANSARI BRADLEYS W

AUTHOR:  R. Kaas

INSTITUTE:  Mathematical Centre

RECEIVED:  760901

BRIEF DESCRIPTION
For two samples of independent observations \( X_{LX}, \ldots, X_{UX} \) and \( Y_{LY}, \ldots, Y_{UY} \), the procedure computes Ansari-Bradley's test statistic \( W \) after shifting the first sample by a given distance \( \text{SHIFT} \). Furthermore, an approximate two-sided tail probability is computed under the hypothesis that the underlying distribution of the second sample is equal to the underlying distribution of the first, shifted over a given distance \( \text{SHIFT} \). The test is mainly a scale test.

KEYWORDS
Ansari-Bradley's test statistic \( W \)

CALLING SEQUENCE
Heading
"REAL" "PROCEDURE" ANSARI BRADLEYS W (X, LX, UX, Y, LY, UY, SHIFT, P2, SORTED);
"VALUE" LX, UX, LY, UY, SORTED;
"INTEGER" LX, UX, LY, UY;
"REAL" SHIFT, P2;
"ARRAY" X, Y;
"BOOLEAN" SORTED;
"CODE" 42404;

Formal parameters
\( x: \)  <array identifier>, vector containing the first sample \( X_{LX}, \ldots, X_{UX} \);
\( lx: \)  <integer arithmetic expression>, smallest index of the first sample;
\( ux: \)  <integer arithmetic expression>, largest index of the first sample;
\( y: \)  <array identifier>, vector containing the second sample \( Y_{LY}, \ldots, Y_{UY} \);
\( ly: \)  <integer arithmetic expression>, smallest index of the second sample;
\( uy: \)  <integer arithmetic expression>, largest index of the second sample;
\( shift: \)  <arithmetic expression>, shift applied to the first sample;
\( p2: \)  <real variable>, output parameter, which at exit contains the value of the two-sided tail probability;
\( sorted: \)  <Boolean expression>, indicating whether the samples are sorted in non-decreasing order or not.
DATA AND RESULTS
The value of the test statistic is assigned to the procedure identifier
ANSARI BRADLEYS W, and the value of the two-sided tail probability to the output
parameter P2.
The following error messages may appear:
Error number 2 (if LX > UX - 1)
Error number 5 (if LY > UY - 1)

PROCEDURES USED
VEG QSORT STATAL 11020
STATAL3 ERROR STATAL 40100
PHI STATAL 41500

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
Ansari-Bradley's W is computed as the sum over the first sample of the
Ansari-Bradley ranks. These are defined as
\[ A-B-RANK(X[i]) = \min(r(X[i]), N+M+1-r(X[i])) \]
where \( r(X[i]) \) is the rank of \( X[i] \) in the pooled sample
\( X[1:i-\text{shift}], \ldots, X[i]-\text{shift}, Y[i], \ldots, Y[j] \), and \( N \) and \( M \) are the sizes of
the two samples. In case of ties midranks are used. The two-sided tail prob-
bility is computed using a normal approximation with continuity correction.

REFERENCE

EXAMPLE OF USE

Program:

"BEGIN"
"REAL" ABW, PROB;
"REAL" "ARRAY" SAMPLE1[1:8], SAMPLE2[1:12];
INARRAY(60, SAMPLE1); INARRAY(60, SAMPLE2);
ABW := ANSARI BRADLEYS W(SAMPLE1, 1, 8, SAMPLE2, 1, 12,
0, PROB, "FALSE");
OUTPUT(61, ""("ANSARI-BRADLEY'S W = ");" +5ZD.6D, /
"("TAIL PROBABILITY = ");" +5ZD.6D")", ABW, PROB)
"END"

219
Input:
34 84 32 52 15 52 14 22
99 25 80 56 90 84 40 26 16 88 24 56

Output:
ANSARI-BRADLEY'S W = +47.500000
TAIL PROBABILITY = +0.640652

SOURCE TEXT
"CODE" 42404;
"REAL" "PROCEDURE" ANSARI BRADLEY(X, XLOW, XUPP, Y, YLOW,
YUPP, MU, P2, SORTED);
"VALUE" XLOW, XUPP, YLOW, YUPP, MU, SORTED;
"INTEGER" XLOW, XUPP, YLOW, YUPP;
"REAL" MU, P2;
"ARRAY" X, Y; "BOOLEAN" SORTED;
"BEGIN" "INTEGER" I, J, MNU, NNU, TNU, M, N, NT, NT2, TTOT;
"REAL" R, MIN, A, B, WAB, EWAB, SIGAB, D2;

"REAL" "PROCEDURE" RANK(FIRST, LAST);
"VALUE" FIRST, LAST;
"INTEGER" FIRST, LAST;
RANK:= "IF" LAST <= NT2 "THEN" (FIRST + LAST) / 2 "ELSE"
"IF" FIRST > NT2 "THEN" NT + 1 - (FIRST + LAST) / 2 "ELSE"
(NT2 - FIRST + 1) * (NT2 + FIRST) +
(LAST - NT2) * (NT + 1 - LAST + NT - NT2)) /
(LAST - FIRST + 1);
"IF" XLOW > XUPP - 2 "THEN" STATA5ERROR(""ANSARI BRADLEY""), 2, XLOW);
"IF" YLOW > YUPP - 2 "THEN" STATA5ERROR(""ANSARI BRADLEY""), 5, YLOW);
"IF" "NOT" SORTED "THEN"
"BEGIN" VEC QSORT(X, XLOW, XUPP);
VEC QSORT(Y, YLOW, YUPP) "END";
WAB:= D2:= 0; M:= XUPP - XLOW + 1; N:= YUPP - YLOW + 1;
NT:= M + N; I:= XLOW; J:= YLOW;
NT2:= (NT + 1) // 2; A:= X[I] - MU; B:= Y[J]; TTOT:= 0;
"FOR" MIN:= "IF" A < B "THEN" A "ELSE" B "WHILE" I <= XUPP & J <= YUPP "DO"
"BEGIN" MNU:= I; NNU:= J;
"FOR" A:= X[I] - MU "WHILE" A = MIN & I < XUPP,
A "WHILE" A = MIN & I = XUPP "DO"
I:= I + 1;
"FOR" B:= Y[J] "WHILE" B = MIN & J < YUPP,
B "WHILE" B = MIN & J = YUPP "DO"
J:= J + 1;
MNU:= I - MNU; NNU:= J - NNU; TNU:= MNU + NNU;
R:= RANK(TTOT + 1, TTOT + TNU);
D2:= D2 + TNU * R * R;
WAB:= WAB + MNU * R; TTOT:= TTOT + TNU
"END";
"IF" I <= XUPP "THEN"
"BEGIN" MIN:= A; MNU:= 1;
"FOR" I:= I + 1 "STEP" 1 "UNTIL" XUPP "DO"
"IF" X[IJ] - MU = MIN "THEN" MNU:= MNU + 1 "ELSE"
"BEGIN" R:= RANK(TTOT + 1, TTOT + MNU);
D2:= D2 + R * R * MNU; WAB:= WAB + MNU * R;
TTOT:= TTOT + MNU; MNU:= 1; MIN:= X[IJ] - MU
"END";
R:= RANK(TTOT + 1, NT); D2:= D2 + R * R * MNU;
WAB:= WAB + R * MNU
"END" "ELSE" "IF" J <= YUPP "THEN"
"BEGIN" MIN:= B; MNU:= 1;
"FOR" J:= J + 1 "STEP" 1 "UNTIL" YUPP "DO"
"IF" Y[J] = MIN "THEN" MNU:= MNU + 1 "ELSE"
"BEGIN" D2:= D2 + RANK(TTOT + 1, TTOT + MNU) ** 2 * MNU;
TTOT:= TTOT + MNU; MNU:= 1; MIN:= Y[J]
"END";
D2:= D2 + RANK(TTOT + 1, NT) ** 2 * MNU;
"END";

ANSARIBRADLEY:= WAB; "IF" NT // 2 * 2 = NT "THEN"
"BEGIN" EWAB:= M * (NT + 2) / 4;
SIGWAB:= N * M / 16 / NT / (NT - 1) * 
(NT * 16 - NT * (NT + 2) ** 2)
"END" "ELSE"
"BEGIN" EWAB:= M * (NT + 1) ** 2 / NT / 4;
SIGWAB:= N * M / 16 / NT / (NT - 1) * 
(NT * 16 - NT * (NT + 1) ** 4)
"END";
"IF" SIGWAB <= 0 "THEN" P2:= 1 "ELSE"
P2:= 2 + PHI((-ABS(WAB - EWAB) + .5) / SQRT(SIGWAB))
"END" ANSARI BRADLEY;
"EOP"
Kendalls Tau

TITLE: Kendall's Tau

AUTHOR: R. Kaas

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION
For a sample of paired observations \((x_{1,\ldots,\infty}, y_{1,\ldots,\infty})\), the procedure computes Kendall's rank correlation coefficient \(\tau\), Kendall's test statistic \(S\) and an approximation of the two-sided tail probability under the hypothesis that given \(x_{1,\ldots,\infty}, y_{1,\ldots,\infty}\), all permutations of \(y_{1,\ldots,\infty}\) have equal probability. A special case of such a hypothesis is independence.

KEYWORDS
Kendall's rank correlation coefficient

CALLING SEQUENCE

Heading
"REAL" "PROCEDURE" KENDALLS TAU (X, Y, LXY, UXY, KENDALLS S, P2);
"VALUE" LXY, UXY;
"INTEGER" LXY, UXY;
"REAL" KENDALLS S, P2;
"ARRAY" X, Y;
"CODE" 42444;

Formal parameters

\[ X: \quad \langle\text{array identifier}\rangle, \text{vector containing the first components} \]
\[ x_{1,\ldots,\infty}, x_{1,\ldots,\infty}; \]
\[ Y: \quad \langle\text{array identifier}\rangle, \text{vector containing the second components} \]
\[ y_{1,\ldots,\infty}, y_{1,\ldots,\infty}; \]
\[ LXY: \quad \langle\text{integer arithmetic expression}\rangle, \text{smallest index of the sample}; \]
\[ UXY: \quad \langle\text{integer arithmetic expression}\rangle, \text{largest index of the sample}; \]
\[ KENDALLS S: \quad \langle\text{real variable}\rangle, \text{output parameter, which at exit contains the value of Kendall's S}; \]
\[ P2: \quad \langle\text{real variable}\rangle, \text{output parameter, which at exit contains the approximate value of the two-sided tail probability}. \]

DATA AND RESULTS
The value of \(\tau\) is assigned to the procedure identifier KENDALLS TAU, the value of the test statistic to the output parameter KENDALLS S, and the approximate value of the two-sided tail probability to the output parameter P2. In the case that \(\tau\) cannot be computed the values \(40/9 = 4.44\ldots\) and 1 are assigned to KENDALLS\(\tau\) and P2.

The following error message may appear:
Error number 3 (if \( L_X Y > U_X Y - 1 \))

**PROCEDURES USED**
- VECQSORT: STATA 11020
- VEC2QSOR: STATA 11024
- STATA3 ERROR: STATA 40100
- PHI: STATA 41500

**LANGUAGE**
- Algol 60

**METHOD AND PERFORMANCE**
Kendall's \( \tau \) and Kendall's \( s \) are computed according to the usual formulas. In case of ties midranks are used. The approximated two-sided tail probability is based on a normal approximation with continuity correction of Kendall's \( s \).

**EXAMPLE OF USE**

*Program:*

```
"BEGIN"
"REAL" TAU, S, PROB;
"REAL" "ARRAY" SAMPLE1, SAMPLE2[1:10];
INARRAY(60, SAMPLE1); INARRAY(60, SAMPLE2);
TAU:= KENDALLS TAU(SAMPLE1, SAMPLE2, 1, 10, S, PROB);
OUTPUT(61, "(""KENDALL'S TAU = ")", +5ZD.60, /,
     "(""KENDALL'S S = ")", +5ZD.60, /,
     "(""TAIL PROBABILITY = ")", +5ZD.60")",
     TAU, S, PROB)
"END"
```

*Input:*

```
87 9 19 57 19 39 2 68 70 28
56 99 87 23 36 33 60 1 35 12
```

*Output:*

```
KENDALL'S TAU = -0.359573
KENDALL'S S = -16.000000
TAIL PROBABILITY = +0.178399
```
Kendalls Tau

2.1.2.3

SOURCE TEXT

"CODE" 42411;
"REAL" "PROCEDURE" KENDALLS TAU(X, Y, LOW, UPP,
KENDALLS S, P2);
"VALUE" LOW, UPP; "INTEGER" LOW, UPP; "REAL" KENDALLS S, P2;
"ARRAY" X, Y;
"IF" LOW + 1 >= UPP "THEN"
STATAL3 ERROR(""KENDALLS TAU"", 3, LOW)
"ELSE"
"BEGIN" "INTEGER" F, FF, H, J, K, M, LOW1, UPP1;
"REAL" A, B, CS, TX2, TX3, TY2, TY3, DENOM;
"BOOLEAN" ANTI;
"REAL" "ARRAY" Y1[LOW - 1 - (UPP - LOW) // 2 : UPP];

"REAL" "PROCEDURE" MSORT (A, L1, U2);
"VALUE" L1, U2; "INTEGER" L1, U2; "REAL" "ARRAY" A;
"BEGIN" "INTEGER" J, J1, J2, D1, D2, S, N1, N2;
"REAL" X, X1, X2, Y;
"BOOLEAN" B1, B2;

"PROCEDURE" SORT (L1, U2, LEFT);
"VALUE" LEFT; "INTEGER" L1, U2; "BOOLEAN" LEFT;
"IF" L1 = U2 "THEN"
"BEGIN" "IF" LEFT "THEN"
U2 := L1 "END"
"END" "ELSE"
"IF" L1 + 1 = U2 "THEN"
"BEGIN" X1 := A [L1]; X2 := A [U2];
"IF" X1 < X2 "THEN" S := S + 1 "ELSE"
"IF" X1 > X2 "THEN"
"BEGIN" S := S - 1;
"IF" "NOT" LEFT "THEN"
"ELSE"
"BEGIN" X := X1; X1 := X2; X2 := X "END"
"END";
"IF" LEFT "THEN"
"BEGIN" L1 := L1 - 2; U2 := U2 - 2;
"END"

"END" "ELSE"
"BEGIN" "INTEGER" L2, U1;
U1 := L1 + (U2 - L1) // 2; L2 := U1 + 1;
SORT (L1, U1, "TRUE"); SORT (L2, U2, "FALSE");
D1 := U1 - L1 + 1; D2 := U2 - L2 + 1;
J1 := L1; J2 := L2; X1 := A [J1]; X2 := A [J2];
J := L1 := "IF" LEFT "THEN" L1 - (U2 - L2 + 1)
"ELSE" U1 + 1;
START: B1 := X1 <= X2; B2 := X1 >= X2; N1 := N2 := 0;
"IF" B1 "THEN"
"BEGIN" LAB1 : A [J]:= X1; J:= J + 1;
    N1:= N1 + 1; J1:= J1 + 1;
    "IF" J1 <= U1 "THEN"
    "BEGIN" Y:= A [J1];
    "IF" Y = X1 "THEN" "GOTO" LAB1;
    X1:= Y
    "END"
"END";
"IF" B2 "THEN"
"BEGIN" LAB2 : A [J]:= X2; J:= J + 1;
    N2:= N2 + 1; J2:= J2 + 1;
    "IF" J2 <= U2 "THEN"
    "BEGIN" Y:= A [J2];
    "IF" Y = X2 "THEN" "GOTO" LAB2;
    X2:= Y
    "END"
"END";
D1:= D1 - N1; D2:= D2 - N2;
S:= S + N1 * D2 - N2 * D1;

"GOTO"
"IF" J1 > U1 "THEN"
(""IF" "NOT" LEFT "THEN" OUT "ELSE" FINISH2)
"ELSE"
"IF" J2 > U2 "THEN"
(""IF" LEFT "THEN" OUT "ELSE" FINISH1)
"ELSE" START;
FINISH1: A [J]:= X1; J:= J + 1; J1:= J1 + 1;
"IF" J1 > U1 "THEN" "GOTO" OUT;
    X1:= A [J1]; "GOTO" FINISH1;
FINISH2: A [J]:= X2; J:= J + 1; J2:= J2 + 1;
"IF" J2 > U2 "THEN" "GOTO" OUT;
    X2:= A [J2]; "GOTO" FINISH2;
OUT: "IF" LEFT "THEN" U2:= U1
"END" OF SORT;
S:= 0; "IF" U2 > L1 "THEN" SORT (L1, U2, "FALSE");
MSORT:= S
"END" MSORT;

"PROCEDURE" TIES AND SHADOWS (J);
"VALUE" J; "INTEGER" J;
"BEGIN" K:= J - 1; F:= J - H; "IF" "NOT" ANTI "THEN"
"BEGIN" FF:= F * F; TX2:= TX2 + FF;
    TX3:= TX3 + FF * F "END";
"FOR" M:= H "STEP" 1 "UNTIL" K "DO" Y1[M]:= Y[M];
VECSORT (Y1, H, K);
"COMMENT" NOW THE SHADOW IS SORTED;
"IF" ANTI "THEN"
"BEGIN" M:= H - 1;
"FOR" M:= M + 1 "WHILE" M < K "DO"
"BEGIN" A:= Y1[K]; Y1[M]:= Y1[K];
    Y1[K]:= A; K:= K - 1
"END" NOW THE SHADOW IS ANTI-SORTED;
"END"
"END" TIES AND SHADOWS;

"COMMENT" SORTING OF THE X-SAMPLE WITH SIMULTANEOUS
REPLACEMENTS IN THE Y-SAMPLE;
VEC2QSORT(X, Y, LOW, UPP);

"COMMENT" DETERMINATION OF TIES IN X-SAMPLE AND THEIR
SHADOWS IN THE Y-SAMPLE;
TX2:= TX3:= 0; ANTI:= "FALSE";
TS:= H:= LOW; A:= X[HI];
"FOR" J:= LOW + 1 "STEP" 1 "UNTIL" UPP "DO"
"BEGIN" B:= X[J]; 
"IF" "NOT" A = B "THEN"
"BEGIN" TIES AND SHADOWS (J); H:= J; A:= B "END"
"END";
TIES AND SHADOWS (UPP + 1);

"COMMENT" CALCULATION OF KENDALLS S;
LOW1:= LOW; UPP1:= UPP; 
"IF" ANTI "THEN"
KENDALLS S:= (CS + MSORT (Y1, LOW1, UPP1)) / 2 "ELSE"
"BEGIN" CS:= MSORT (Y1, LOW1, UPP1);
ANTI:= "TRUE"; "GOTO" TS
"END";

"COMMENT" DETERMINATION OF TIES IN THE Y-SAMPLE;
TY2:= TY3:= 0; H:= LOW1; A:= Y[HI];
"FOR" J:= LOW1 + 1 "STEP" 1 "UNTIL" UPP1 "DO"
"BEGIN" B:= Y[J]; 
"IF" "NOT" A = B "THEN"
"BEGIN" F:= J - H; FF:= F * F;
TY2:= TY2 + FF; TY3:= TY3 + FF * F;
H:= J; A:= B
"END"
"END";
F:= UPP1 - LOW1 + 1; FF:= F * F;
TY3:= TY3 + FF * F;

F:= UPP - LOW + 1; FF:= F * F;
DENOM:= ((FF - TX2) * (FF - TY2));
"IF" "THEN"
"BEGIN" KENDALLS TAU:= 2 * KENDALLS S / SQRT(DENOM);
P2:= "IF" F > 2
"THEN" 2 * PHI((1 - ABS(KENDALLS S)) /
SQRT(((2 * F + 3) * F + 5) * F
- 3 * (TX2 + TY2) - 2 * (TX3 + TY3)) / 18
+ (3 * TX2 - TX3 + 2 * F) * 
(3 * TY2 - TY3 + 2 * F) /
(9 * F * (F - 1) * (F - 2))
+ (TX2 - F) * (TY2 - F) / (2 * F * (F - 1)))
"ELSE" 1
"END"
"ELSE" "BEGIN" P2:= 1;
KENDALLS TAU:= 40 / 9 "END"

"END" KENDALLS TAU;
"EOP"
TITLE:  Spearman's Rho

AUTHOR:  R. Kaas

INSTITUTE:  Mathematical Centre

RECEIVED:  760901

BRIEF DESCRIPTION
For a sample of paired observations \((x_{1}, y_{1}), \ldots, (x_{n}, y_{n})\), the procedure computes Spearman's rank correlation coefficient \(Rho\), Spearman's test statistic \(T\), and an approximation of the two-sided tail probability under the hypothesis that given \(x_{1}, \ldots, x_{n}\) all permutations of \(y_{1}, \ldots, y_{n}\) have equal probability. A special case of such a hypothesis is independence.

KEYWORDS
Spearman's rank correlation coefficient

CALLING SEQUENCE

```
Heading
"REAL" "PROCEDURE" SPEARMANS RHO (X, Y, LXY, UXY, SPEARMANS T, P2);
"VALUE" LXY, UXY;
"INTEGER" LXY, UXY;
"REAL" SPEARMANS T, P2;
"ARRAY" X, Y;
"CODE" 42412;
```

Formal parameters

- \(X\): <array identifier>, vector containing the first components \(x_{1}, \ldots, x_{n}\);
- \(Y\): <array identifier>, vector containing the second components \(y_{1}, \ldots, y_{n}\);
- \(LXY\): <integer arithmetic expression>, smallest index of the sample;
- \(UXY\): <integer arithmetic expression>, largest index of the sample;
- \(SPEARMANS T\): <real variable>, output parameter, which at exit contains the value of Spearman's \(T\);
- \(P2\): <real variable>, output parameter, which at exit contains the approximate value of the two-sided tail probability.

DATA AND RESULTS
The value of \(Rho\) is assigned to the procedure identifier SPEARMANS RHO, the value of the test statistic to the output parameter SPEARMANS T, and the approximated value of the two-sided tail probability to the output parameter \(P2\). In the case that \(Rho\) cannot be computed or if \(ABS(Rho)=1\), the values \(4 \times 10^{15}\) and 1 are assigned to SPEARMANS T and \(P2\).

The following error message may appear:
Spearman’s Rho

Error number 3 (if LXY ≥ UXY − 1)

PROCEDURES USED
VEC PERM STATAL 11022
VEC RANKIE STATAL 11023
STATAL3 ERROR STATAL 40100
STUDENT STATAL 41530

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
Spearman’s RHO and Spearman’s T are computed according to the usual formulas. In case of ties midranks are used. The approximated two-sided tail probability is based on the fact that \( RHO \times \sqrt{\frac{(UXY - LXY - 1)}{(1 - RHO^2)}} \) has approximately a Student distribution with (UXY - LXY - 1) degrees of freedom.

EXAMPLE OF USE

Program:

"BEGIN"
"REAL" RHO, T, PROB;
"REAL" "ARRAY" SAMPLE1, SAMPLE2[1:10];
INARRAY(60, SAMPLE1); INARRAY(60, SAMPLE2);
RHO:= SPEARMANS RHO(SAMPLE1, SAMPLE2, 1, 10, T, PROB);
OUTPUT(61, """"(""SPEARMAN'S RHO = ")", +S2D.60, ",
""""SPEARMAN'S T = ")", +S2D.60, ",
""TAIL PROBABILITY = ")", +S2D.60")",
RHO, T, PROB)
"END"

Input:
93 44 12 90 73 0 18 32 42 81
82 6 55 66 38 94 74 4 95 91

Output:
SPEARMAN'S RHO = -0.018182
SPEARMAN'S T = -0.051434
TAIL PROBABILITY = +0.960240
"CODE" 42412;
"REAL" "PROCEDURE" SPEARMANS RHO(X, Y, LOW, UPP, SPEARMANS T, P2);
"VALUE" LOW, UPP;
"INTEGER" LOW, UPP;
"REAL" SPEARMANS T, P2;
"ARRAY" X, Y;
"IF" LOW + 1 >= UPP "THEN"
STATA3 ERROR(""SPEARMANS RHO"", 3, LOW)
"ELSE"
"BEGIN" "INTEGER" N, H;
"REAL" N3, TIES2, TIES3, CTX, CTY, RHO, D;
"INTEGER" "ARRAY" PERMLow : UPP];
N:= UPP - LOW + 1; N3:= N * N * N;
VECRANKTIE (Y, LOW, UPP, PERM, Y, TIES2, TIES3);
CTX:= N3 - TIES3;
VECRANKTIE (X, LOW, UPP, PERM, X, TIES2, TIES3);
CTX:= N3 - TIES3;
D:= 0;
"FOR" H:= LOW "STEP" 1 "UNTIL" UPP "DO"
D:= D + (X[H] - Y[H]) ** 2;
RHO:= "IF" CTX = 0 "OR" CTY = 0 "THEN" 2 "ELSE" .5 * (CTX + CTY - 12 * D) / SQRT (CTX * CTY);
SPEARMANS T:= 4.15 / 9; P2:= 1;
"IF" ABS(RHO) > 1 "THEN" RHO:= 40 / 9 "ELSE"
"IF" ABS(RHO) < 1 "THEN"
"BEGIN"
SPEARMANS T:= RHO * SQRT((N - 2) / (1 - RHO * RHO));
"IF" N > 2
"THEN" P2:= 2 * STUDENT(-ABS(SPEARMANS T), N - 2)
"END";
SPEARMANS rho:= RHO;
VECPERM (PERM, LOW, UPP, X); VECPerm (PERM, LOW, UPP, Y)
"END" SPEARMANS rho;
"EOD"
TITLE: Smirnovs D

AUTHOR: E. Opperdoes

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION
For two samples of independent observations \( x_{[Lx]}, \ldots, x_{[Ux]} \) and \( y_{[Ly]}, \ldots, y_{[Uy]} \), the procedure computes Smirnov's test statistic \( D \) after shifting the first sample by a given distance \( \text{SHIFT} \). Furthermore, an approximated right tail probability of the test statistic is computed under the hypothesis that the underlying distribution of the second sample is equal to the underlying distribution of the first, shifted over a given distance \( \text{SHIFT} \) (this test is two-sided). The test is also called the Kolmogorov-Smirnov test.

KEYWORDS
(Kolmogorov-) Smirnov's two-sample test statistic D

CALLING SEQUENCE
Heading
"REAL" "PROCEDURE" SMIRNOVS D (X, Lx, Ux, Y, Ly, Uy, SHIFT, PR, SORTED);
"VALUE" LX, UX, LY, UY, SORTED;
"INTEGER" LX, UX, LY, UY;
"REAL" SHIFT, PR;
"BOOLEAN" SORTED;
"REAL" "ARRAY" X, Y;
"CODE" 42406;

Formal parameters
X: <array identifier>, vector containing the first sample \( x_{[Lx]}, \ldots, x_{[Ux]} \);
LX: <integer arithmetic expression>, smallest index of the first sample;
UX: <integer arithmetic expression>, largest index of the first sample;
Y: <array identifier>, vector containing the second sample \( y_{[Ly]}, \ldots, y_{[Uy]} \);
LY: <integer arithmetic expression>, smallest index of the second sample;
UY: <integer arithmetic expression>, largest index of the second sample;
SHIFT: <arithmetic expression>, shift applied to the first sample;
PR: <real variable>, output parameter, which at exit contains the approximate value of the right tail probability;
SORTED: <boolean expression>, indicating whether the samples are sorted in non-decreasing order or not.
DATA AND RESULTS
The value of Smirnov's test statistic $D$ is assigned to the procedure identifier SMIRNOVS D and the approximate value of the right tail probability to the output parameter PR.
The following error messages may appear:
Error number 2 (if $LX > UX - 1$)
Error number 6 (if $LY > UY - 1$)

PROCEDURES USED
VECSORT
STATAL 11020
STATAL3 ERROR
STATAL 40100

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
Smirnov's test statistic $D$ equals the maximal value of $\max_{z} |F_1(z) - F_2(z)|$, for $z = \max_{x \in X} - \text{SHIFT}, \ldots, \max_{x \in X} - \text{SHIFT}, \min_{y \in Y} y, \ldots, \min_{y \in Y} y$, where $F_1$ and $F_2$ are the empirical distributions of the first and second sample. The approximate value of the tail probability is computed using a formula in Smirnov (1984). In case of continuous underlying distributions this is the exact asymptotic tail probability; in case of discrete underlying distributions this test is conservative.

REFERENCES

EXAMPLE OF USE
Program:
"BEGIN"
"REAL" D, PROB;
"REAL" "ARRAY" SAMPLE1[1:8], SAMPLE2[1:12];
INARRAY(60, SAMPLE1); INARRAY(60, SAMPLE2);
D := SMIRNOVS D(SAMPLE1, 1, 8, SAMPLE2, 1, 12, 0,
PROB "FALSE" );
OUTPUT(61, "(""SMIRNOV'S D = "")", +5ZD.6D,/, 
"(""TAIL PROBABILITY = "")", +5ZD.6D")", D, PROB)
"END"
Input:
94 83 17 6 43 95 93 47
2 62 38 83 27 30 99 83 9 19 53 41

Output:
SMIRNOV'S D = +0.333333
TAIL PROBABILITY = +0.660386

SOURCE TEXT
"CODE" 42406;
"REAL" "PROCEDURE" SMIRNOVS D(OBS1, L1, U1, OBS2, L2, U2,
      MU, P2, SORTED); "REAL" P2, MU;
"VALUE" L1, U1, L2, U2, MU, SORTED; "REAL" P2, MU;
"INTEGER" L1, U1, L2, U2; "BOOLEAN" SORTED;
"REAL" "ARRAY" OBS1, OBS2;
"BEGIN" "INTEGER" I, J, S1, S2, NOBS1, NOBS2;
"BOOLEAN" STOP1, STOP2;
"REAL" X1, X2, Y1, Y2, F1, F2, T;

"PROCEDURE" TIE (ARRAY, UPP, STOP, INDEX, LAST, NEXT,
      SUM, FUNCTION, NOBS);
"VALUE" UPP, NOBS;
"INTEGER" UPP, INDEX, SUM, NOBS;
"BOOLEAN" STOP;
"REAL" LAST, NEXT, FUNCTION;
"REAL" "ARRAY" ARRAY;
"IF" "NOT" STOP "THEN"
"BEGIN" STEP: SUM:= SUM + 1; INDEX:= INDEX + 1;
"IF" INDEX > UPP "THEN"
"BEGIN" FUNCTION:= 1; STOP:= "TRUE" "END" "ELSE"
"BEGIN" NEXT:= ARRAY [INDEX];
"IF" LAST = NEXT "THEN" "GOTO" STEP "ELSE"
"BEGIN" FUNCTION:= SUM / NOBS; LAST:= NEXT "END"
"END"
"END" TIE;

"REAL" "PROCEDURE" KOLSMYRAS (X); "VALUE" X; "REAL" X;
"BEGIN" "INTEGER" K; "REAL" TERM, SUM, EPS;
EPS:= "-6 / 2; SUM:= 0; X:= -2 * X * X;
"FOR" K:= 1 , K + 2 "WHILE" TERM >= EPS "DO"
"BEGIN" TERM:= EXP(X * K * K) *
   (1 - EXP(X * (K * 2 + 1)));
      SUM:= SUM + TERM
"END";
KOLSMYRAS:= 2 * SUM
"END" KOLSMYRAS;

"IF" L1 > U1 - 2 "THEN"
STATA3 ERROR (""SMIRNOVS D""), L1);
"IF" L2 > U2 - 2 "THEN"
STATA3 ERROR (""SMIRNOVS D""), L2);
"IF" M = 0 "THEN"
"FOR" I:= L1 "STEP" 1 "UNTIL" U1 "DO"
   OBS[I] := OBS[I] - M;
"IF" "NOT" SORTED "THEN"
"BEGIN" VEC QSORT(OBS1, L1, U1);
   VEC QSORT (OBS2, L2, U2) "END";
NOS1 := U1 - L1 + 1; NOS2 := U2 - L2 + 1;
I:= L1; J:= L2; S1:= S2:= 0; T:= F1:= F2:= 0;
STOP1:= STOP2:= "FALSE"; X1:= OBS1 [L1]; Y1:= OBS2 [L2];
AGAIN: "IF" X1 = Y1 "THEN"
"BEGIN" TIE (OBS1, U1, STOP1, I, X1, X2, S1, F1, NOS1);
   TIE (OBS2, U2, STOP2, J, Y1, Y2, S2, F2, NOS2)
"END" "ELSE"
"IF" X1 < Y1 "THEN"
"BEGIN" TIE (OBS1, U1, STOP1, I, X1, X2, S1, F1, NOS1);
   "IF" STOP1 "THEN"
   TIE (OBS2, U2, STOP2, J, Y1, Y2, S2, F2, NOS2)
"END" "ELSE"
"BEGIN" TIE (OBS2, U2, STOP2, J, Y1, Y2, S2, F2, NOS2);
   "IF" STOP2 "THEN"
   TIE (OBS1, U1, STOP1, I, X1, X2, S1, F1, NOS1)
"END";
"IF" STOP1 "AND" STOP2 "THEN"
"BEGIN" SMIRNOVS D := T;
   P2 := KOLSMYRAS (T / SQRT (1 / NOS1 + 1 / NOS2))
"END" "ELSE"
"BEGIN" "REAL" H := ABS (F1 - F2);
   "IF" T < H "THEN" T := H;
   "GOTO" AGAIN
"END"
"END" SMIRNOVS D;
"EOP"
Cramer von Mises W2

TITLE: Cramer von Mises W2

AUTHOR: E. Oppendoes

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION
For two samples of independent observations \(x_1, x_2, \ldots, x_n\) and \(y_1, y_2, \ldots, y_m\), the procedure computes Cramer-von Mises test statistic \(W_2\) after shifting the first sample by a given distance \(\text{SHIFT}\). Furthermore, an approximated right tail probability of the test statistic is computed under the hypothesis that the underlying distribution of the second sample is equal to the underlying distribution of the first, shifted over a given distance \(\text{SHIFT}\). (This test is two-sided).

KEYWORDS
Cramer-von Mises' two-sample test statistic \(W_2\)

CALLING SEQUENCE

**Heading**
"REAL" "PROCEDURE" CRAMER VON MISSES W2 (X, LX, UX, Y, LY, UY, SHIFT, PR, SORTED);
"VALUE" LX, UX, LY, UY, SORTED;
"INTEGER" LX, UX, LY, UY;
"REAL" SHIFT, PR;
"BOOLEAN" SORTED;
"REAL" "ARRAY" X, Y;
"CODE" 42408;

**Formal parameters**

- **x**: <array identifier>, vector containing the first sample \(x_1, \ldots, x_n\);
- **LX**: <integer arithmetic expression>, smallest index of the first sample;
- **UX**: <integer arithmetic expression>, largest index of the first sample;
- **y**: <array identifier>, vector containing the second sample; \(y_1, y_2, \ldots, y_m\);
- **LY**: <integer arithmetic expression>, smallest index of the second sample;
- **UY**: <integer arithmetic expression>, largest index of the second sample;
- **SHIFT**: <arithmetic expression>, shift applied to the first sample;
- **PR**: <real variable>, output parameter, which at exit contains the approximate value of the right tail probability;
- **SORTED**: <boolean expression>, indicating whether the samples are
sorted in non-decreasing order or not.

**DATA AND RESULTS**
The value of Cramer-von Mises' test statistic $w_2$ is assigned to the procedure
identifier `Cramer von Mises W2`, and the approximate value of the right tail
probability to the output parameter `PR`.
The following error messages may appear:
Error number 2 (if $LX=UX-1$)
Error number 6 (if $LY=UY-1$)

**PROCEDURES USED**

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIMIT</td>
<td>STATAL LIMIT</td>
</tr>
<tr>
<td>VEC QSORT</td>
<td>STATAL 11020</td>
</tr>
<tr>
<td>STATAL.3 ERROR</td>
<td>STATAL 40100</td>
</tr>
<tr>
<td>BESSKAO1</td>
<td>NUMAL 35191</td>
</tr>
</tbody>
</table>

**LANGUAGE**
Algol 60

**METHOD AND PERFORMANCE**
Cramer-von Mises' test statistic $w_2$ equals

$$\frac{MN}{(M+N)^2} \sum_i \left( F_1(z_i) - F_2(z_i) \right)^2,$$

where the sum is over $i = XLX_1 - SHIFT, ..., XLX_3 - SHIFT$, $YLX_1, ..., YLY_3$,
the functions $f_1$ and $f_2$ are the empirical distribution functions of the samples,
and $M$ and $N$ are the sample sizes. The approximated value of the test statistic
is computed using a formula in Anderson and Darling (1952).

**REFERENCES**


**EXAMPLE OF USE**

*Program:*

"BEGIN"
"REAL" W2, PROB;
"REAL" "ARRAY" SAMPLE1[1:8], SAMPLE2[1:2];
INARRAY(60, SAMPLE1); INARRAY(60, SAMPLE2);
W2:= CRAMER VON MISSES W2(SAMPLE1, 1, 8, SAMPLE2, 1, 12, 0,
PROB, "FALSE");
OUTPUT(61, ""(""CRAMER VON MISSES' W2 = "")", +520.60, ",
"("TAIL PROBABILITY = "")", +520.60"", W2, PROB)
"END"
Input:

83 28 57 14 44 14 27 43
0 17 9 10 32 57 27 41 64 82 94 77

Output:

CRAMER VON MISES W2 = +0.050833
TAIL PROBABILITY = +0.871203

SOURCE TEXT

"CODE" 42408;
"REAL" "PROCEDURE" CRAMER VON MISES W2 (OBS1, L1, U1,
   OBS2, L2, U2, MU, P2, SORTED);
"VALUE" L1, U1, L2, U2, MU, SORTED; "REAL" P2, MU;
"INTEGER" L1, U1, L2, U2; "BOOLEAN" SORTED;
"REAL" "ARRAY" OBS1, OBS2;
"BEGIN" "INTEGER" I, J, S1, S2, NOBS1, NOBS2;
"BOOLEAN" STOP1, STOP2;
"REAL" X1, X2, Y1, Y2, F1, F2, T;

"PROCEDURE" TIE (ARRAY, UPP, STOP, INDEX, LAST, NEXT,
   SUM, FUNCTION, NOBS);
"VALUE" UPP, NOBS;
"INTEGER" UPP, INDEX, SUM, NOBS;
"BOOLEAN" STOP;
"REAL" LAST, NEXT, FUNCTION;
"REAL" "ARRAY" ARRAY;
"IF" "NOT" STOP "THEN"
"BEGIN" STEP: SUM:= SUM + 1; INDEX:= INDEX + 1;
   "IF" INDEX > UPP "THEN"
   "BEGIN" FUNCTION:= 1; STOP:= "TRUE" "END" "ELSE"
   "BEGIN" NEXT:= ARRAY [INDEX];
   "IF" LAST = NEXT "THEN" "GOTO" STEP "ELSE"
   "BEGIN" FUNCTION:= SUM / NOBS;
   LAST:= NEXT "END"
   "END"
"END" TIE;

"IF" L1 > U1 - 2 "THEN"
STATA3 ERROR ("CRAMER VON MISES W2"), 2, L1);
"IF" L2 > U2 - 2 "THEN"
STATA3 ERROR ("CRAMER VON MISES W2"), 6, L2);
"IF" MU = 0 "THEN"
"FOR" I:= L1 "STEP" 1 "UNTIL" U1 "DO"
   OBS1[I]:= OBS1[I] - MU;
"IF" "NOT" SORTED "THEN"
"BEGIN" VEC QSORT(OBS1, L1, U1);
   VEC QSORT (OBS2, L2, U2) "END";
   NOBS1:= U1 - L1 + 1; NOBS2:= U2 - L2 + 1; I:= L1;
   J:= L2; S1:= S2:= 0; T:= F1:= F2:= 0;
   STOP1:= STOP2:= "FALSE"; X1:= OBS1 [L1]; Y1:= OBS2 [L2];
AGAIN: "IF" X1 = Y1 "THEN"
"BEGIN" TIE (OBS1, U1, STOP1, I, X1, X2, S1, F1, NOBS1);
    TIE (OBS2, U2, STOP2, J, Y1, Y2, S2, F2, NOBS2)
"END" "ELSE"
"IF" X1 < Y1 "THEN"
"BEGIN" TIE (OBS1, U1, STOP1, I, X1, X2, S1, F1, NOBS1);
    "IF" STOP1 "THEN"
    TIE (OBS2, U2, STOP2, J, Y1, Y2, S2, F2, NOBS2)
"END" "ELSE"
"BEGIN" TIE (OBS2, U2, STOP2, J, Y1, Y2, S2, F2, NOBS2);
    "IF" STOP2 "THEN"
    TIE (OBS1, U1, STOP1, I, X1, X2, S1, F1, NOBS1)
"END";
"IF" STOP1 "AND" STOP2 "THEN"
"BEGIN" CRAMER VON MISES W2 := T := T * NOBS1 * NOBS2 /
                     (NOBS1 + NOBS2) ** 2;
    P2 := 1 - LIMIT (T)
"END" "ELSE"
"BEGIN" T := T + (F1 - F2) ** 2; "GOTO" AGAIN "END"
"END" CRAMER VON MISES W2;
"EOP"
TITLE: Students T1

AUTHOR: J. Bethlehem

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION
For a sample of independent observations $x_{l1}, \ldots, x_{lU}$ from a normal distribution, with unknown mean and variance, the procedure computes Student's test statistic to test the hypothesis that the mean of distribution is equal to a given value $\mu$. Furthermore, the two-sided tail probability of the test statistic is computed.

KEYWORDS
Student's one-sample test statistic

CALLING SEQUENCE

Heading  
"REAL" "PROCEDURE" STUDENTS T1 (X, LX, UX, MU, P2);
"VALUE" LX, UX, MU;
"INTEGER" LX, UX;
"REAL" MU, P2;
"ARRAY" X;
"CODE" 42402;

Formal parameters

- $x$: \(<array\ indentation\>, vector containing the sample $x_{l1}, \ldots, x_{lU}$;
- $lx$: \(<integer\ arithmetic\ expression\>, smallest index of the sample;
- $ux$: \(<integer\ arithmetic\ expression\>, largest index of the sample;
- $mu$: \(<arithmetic\ expression\>, mean of the normal distribution according to the hypothesis;
- $p2$: \(<real\ variable\>, output parameter, which at exit contains the value of two-sided tail probability.

DATA AND RESULTS
The value of the test statistic is assigned to the procedure identifier STUDENTS T1, and the value of the two-sided tail probability to the output parameter $p2$.
The following error messages may appear:

- Errornumber 0 (if it is impossible to compute Student's T1)
- Errornumber 2 (if $lx > ux - 1$)
PROCEDURES USED
STATAL3 ERROR  STATAL 40100
STUDENT        STATAL 41530

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
The computation of Student's $t_1$ and its two-sided tail probability is straightforward.

EXAMPLE OF USE

Program:
"BEGIN"
"REAL" $t_1$, PROB;
"REAL" "ARRAY" SAMPLE[1:10];
INARRAY(60, SAMPLE);
$t_1$: STUDENTS $t_1$(SAMPLE, 1, 10, 50, PROB);
OUTPUT(61, "(""STUDENT'S $t_1$ = ")", +5ZD.6D, ",
"(""TAIL PROBABILITY = ")", +5ZD.6D"), $t_1$, PROB)
"END"

Input:
7 74 12 94 79 22 6 51 88 29

Output:
STUDENT'S $t_1$ = -0.342054
TAIL PROBABILITY = +0.740162

SOURCE TEXT
"CODE" 42402;
"REAL" "PROCEDURE" STUDENTS $t_1$(A, LOW, UPP, MU, P2);
"VALUE" LOW, UPP, MU; "REAL" MU, P2; "ARRAY" A;
"INTEGER" LOW, UPP;
"BEGIN" "INTEGER" I, N; "REAL" S, SS, NOM, AI, T;

S:= SS:= 0; N:= UPP - LOW + 1;
"IF" N <= 2
"THEN" STATAL3 ERROR(""STUDENT'S $t_1$"), 2, LOW);
"FOR" I:= LOW "STEP" 1 "UNTIL" UPP "DO"
"BEGIN" AI:= A[I]; S:= S + AI; SS:= SS + AI * AI "END";
NOM:= SS - S * S / N;
"IF" NOM = 0 "THEN"
STATAL3ERROR(""STUDENT'S $t_1$"), 0, NOM) "ELSE"
"BEGIN" STUDENTS $t_1$: $t_2$=
(S / N - MU) * SQRT(N * (N - 1) / NOM);
P2:= 2 * STUDENT(-ABS(T), N - 1)
"END"
"END" STUDENTS T1;
"EOP"
TITLE: Students T2

AUTHOR: J. Bethlehem

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION
For two samples of independent observations \( x_1, x_2, \ldots, x_{ux} \) and \( y_1, y_2, \ldots, y_{uy} \) from two normal distributions, with unknown means and unknown (but equal) variances, the procedure computes Student's test statistic to test the hypothesis that the difference of the means of the distributions is equal to a given value \( \text{SHIFT} \). Furthermore, the two-sided tail probability of the test statistic is computed.

KEYWORDS
Student's two-sample test statistic

CALLING SEQUENCE
Heading
"REAL" "PROCEDURE" STUDENTS T2 (X, LX, UX, Y, LY, UY, SHIFT, P2);
"VALUE" LX, UX, LY, UY, SHIFT;
"REAL" SHIFT, P2;
"INTEGER" LX, UX, LY, UY;
"ARRAY" X, Y;
"CODE" 42403;

Formal parameters
X: \(<\text{array identifier}>, \text{vector containing the first sample}\ x_1, x_2, \ldots, x_{ux}\);
LX: \(<\text{integer arithmetic expression}>, \text{smallest index of the first sample}\);
UX: \(<\text{integer arithmetic expression}>, \text{largest index of the first sample}\);
Y: \(<\text{array identifier}>, \text{vector containing the second sample}\ y_1, y_2, \ldots, y_{uy}\);
LY: \(<\text{integer arithmetic expression}>, \text{smallest index of the second sample}\);
UY: \(<\text{integer arithmetic expression}>, \text{largest index of the second sample}\);
SHIFT: \(<\text{arithmetic expression}>, \text{shift applied to the first sample}\);
P2: \(<\text{real variable}>, \text{output parameter, which at exit contains the value of the value of the two-sided tail probability}\).
DATA AND RESULTS
The value of the test statistic is assigned to the procedure identifier
STUDENTS T2, and the value of the two-sided tail probability to the output
parameter p2.
The following error messages may appear:
Error number 0 (if it is impossible to compute Student’s T2)
Error number 2 (if LX=UX-1)
Error number 5 (if LX=UY-1)

PROCEDURES USED
STATA 3 ERROR
STUDENT

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
The computation of Student’s T2 and its two-sided tail probability is straight-
forward.

EXAMPLE OF USE
Program:
"BEGIN"
"REAL" T2, PROB;
"REAL" "ARRAY" SAMPLE1[1:8], SAMPLE2[1:12];
INARRAY(60, SAMPLE1); INARRAY(60, SAMPLE2);
T2:= STUDENTS T2(SAMPLE1, 1, 8, SAMPLE2, 1, 12, 50,
PROB);
OUTPUT(61, ""("STUDENT'S T2 = ")", T2, 520.60, 
"("TAIL PROBABILITY = ")", T2, 520.60)
"END"

Input:
34 84 33 62 47 82 87 17
4 14 22 19 50 58 41 50 64 75 33 27
Order:

\[
\text{STUDENT'S T2} = -2.977801 \\
\text{TAIL PROBABILITY} = +0.008065
\]

Source Text:

```
"CODE" 42403;
"REAL" "PROCEDURE" STUDENTS T2(A1, L1, U1, A2, L2, U2, 
    MU, P2);
"VALUE" L1, L2, U1, U2, MU;
"INTEGER" L1, L2, U1, U2;
"REAL" MU, P2;
"ARRAY" A1, A2;
"BEGIN" "INTEGER" M, N, I;
"REAL" SX, SY, SSX, SSY, NOM, AI, T;

SX:= SY:= SSX:= SSY:= 0;
M:= U1 - L1 + 1; N:= U2 - L2 + 1;
"IF" M <= 2
"THEN" STATA3 ERROR("("STUDENTS T2")", 2, L1);
"IF" N <= 2
"THEN" STATA3 ERROR("("STUDENTS T2")", 5, L2);
"FOR" I:= L1 "STEP" 1 "UNTIL" U1 "DO"
"BEGIN" AI:= AI[I]; SX:= SX + AI;
    SSX:= SSX + AI * AI "END";
"FOR" I:= L2 "STEP" 1 "UNTIL" U2 "DO"
"BEGIN" AI:= A2[I]; SY:= SY + AI;
    SSY:= SSY + AI * AI "END";
NOM:= SSX + SSY - SX * SX / M - SY * SY / N;
"IF" NOM <= 0 "THEN"
STATA3ERROR("("STUDENTS T2")", 0, NOM) "ELSE"
"BEGIN" STUDENTS T2:= T:= 
    (SX/M - SY/N - MU) * SQRT((M + N - 2) * M * N / 
        (M + N) / NOM); 
    P2:= 2 * STUDENT(-ABS(T), M + N - 2)
"END"
"END" STUDENTS T2;
"EOP"
```
TITLE: Promocorco

AUTHOR: R. Kaas

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION
For a sample of paired observations \((X_1, Y_1), \ldots, (X_n, Y_n)\) the procedure computes the product-moment correlation coefficient, the covariance and the two-sided tail probability under the hypothesis of independence and normality of the observations.

KEYWORDS
product-moment correlation coefficient

CALLING SEQUENCE

_Heading_
"REAL" *PROCEDURE* PROMOCORCO (X, Y, LXY, UXY, COV, P2);
"VALUE" LXY, UXY;
"INTEGER" LXY, UXY;
"REAL" COV, P2;
"ARRAY" X, Y;
"CODE" 42410;

_Formal parameters_
X: \(<\text{array indentifier}>, \text{vector containing the first components of the sample } X_1, \ldots, X_n\>;
Y: \(<\text{array indentifier}>, \text{vector containing the second components of the sample } Y_1, \ldots, Y_n\>;
LXY: \(<\text{integer arithmetic expression}>, \text{smallest index of the sample}\>;
UXY: \(<\text{integer arithmetic expression}>, \text{largest index of the sample}\>;
COV: \(<\text{real variable}>, \text{output parameter, which at exit contains the value of the covariance}\>;
P2: \(<\text{real variable}>, \text{output parameter, which at exit contains the value of the two-sided tail probability}\>.

DATA AND RESULTS
The value of the product-moment correlation coefficient is assigned to the procedure indentifier PROMOCORCO, the value of the covariance to the output parameter COV and the values of the two-sided tail probability to the output parameter P2. In the case that the denominator is less than \(10^{-8}\) the values \(40/9 = 4.44\ldots\) and 1 are assigned to PROMOCORCO and P2.

The following error message may appear:
Error number 3 (if LXY > UXY - 1)
PROCEDURES USED
STATAL3 ERROR (STATAL 40100)
STUDENT (STATAL 41530)

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
The correlation coefficient and the covariance are computed according to the usual formulas. The sample covariance is an unbiased estimator of the population covariance. The values of PROMOCORCO and COV are computed exactly, the precision of P2 is $10^{-10}$.

EXAMPLE OF USE

Program:

"BEGIN"
"REAL" COR, COV, PROB;
"REAL" "ARRAY" SAMPLE1, SAMPLE2[1:10];
INARRAY(60, SAMPLE1); INARRAY(60, SAMPLE2);
COR:= PROMOCORCO(SAMPLE1, SAMPLE2, 1, 10, COV, PROB);
OUTPUT(61, "("CORRELATION = ")", +5Z0.6D,,
"("COVARIANCE = ")", +5Z0.6D,,
"("TAIL PROBABILITY = ")", +5Z0.6D")",
COR, COV, PROB)
"END"

Input:

72 68 77 78 3 75 33 61 22 20
71 48 8 97 33 61 4 10 38 79

Output:

CORRELATION = +0.187479
COVARIANCE = +170.655556
TAIL PROBABILITY = +0.604003

SOURCE TEXT

"CODE" 42410;
"REAL" "PROCEDURE" PROMOCORCO(X, Y, LOW, UPP, COV, P2);
"VALUE" LOW, UPP;
"INTEGER" LOW, UPP;
"REAL" COV, P2;
"ARRAY" X, Y;
"BEGIN"
"REAL" SX, SY, SXX, SYY, SXY, XI, YI, DENOMINATOR, CORR;
"INTEGER" I, N;
N:= UPP - LOW + 1;
"IF" N <= 2
"THEN" STATA3 ERROR(""PROMOCORCO""), 3, LOW);
SX:= SY:= SXX:= SYY:= SXY:= 0;
"FOR" I:= LOW "STEP" 1 "UNTIL" UPP "DO"
"BEGIN"
   XI:= X[I]; YI:= Y[I]; SX:= SX + XI; SY:= SY + YI;
   SXX:= SXX + XI * XI; SYY:= SYY + YI * YI;
   SXY:= SXY + XI * YI;
"END";
DENOMINATOR:= (SXX - SX * SX / N) * (SYY - SY * SY / N);
COV:= (SXY - SX * SY / N) / (N - 1);
"IF" DENOMINATOR < "-8 "THEN"
"BEGIN" PROMOCORCO:= 40 / 9; P2:= 1;
   ALCMESS(""CONSTANT SAMPLE IN PROMOCORCO""");
"END" "ELSE"
"BEGIN"
PROMOCORCO:= CORR:= COV * (N - 1) / SQRT(DENOMINATOR);
P2:= "IF" ABS(CORR) < 1
   "THEN" 2 * STUDENT(-ABS(CORR * SQRT((N - 2)/
   (1 - CORR * CORR))), N - 2)
   "ELSE" 0
"END"
"END" PROMOCORCO;
"EOP"
2.2.3.1

TITLE: Cormat

AUTHOR: R. Kaas

INSTITUTE: Mathematical Centre

RECEIVED: 750315

BRIEF DESCRIPTION
For a data matrix consisting of NRESP rows, each row representing the observations of one respondent on NVAR variables, the procedure computes the matrix of product-moment correlation coefficient and prints its lower triangular part. Full control over matrix layout, print format, variable labels and values to be skipped is obtained through the parameters of the procedure. The data matrix is read by a procedure called READRESP, which has to be provided by the user elsewhere in the program.

KEYWORDS
Product-moment correlation matrix

CALLING SEQUENCE

Heading
"PROCEDURE" CORMAT (CHN, NCOL, FORMATV, FORMATC, TEXT, NVAR, NRESP, SKIP, READRESP, LABELS);
"VALUE" CHN, NCOL, NVAR, NRESP, SKIP;
"REAL" SKIP;
"INTEGER" CHN, NCOL, NVAR, NRESP;
"STRING" FORMATV, FORMATC, TEXT;
"PROCEDURE" READRESP, LABELS;
"CODE" 44410;

Formal parameters
CHN: <integer arithmetic expression>, channel number via which the output is written to file;
NCOL: <integer arithmetic expression>, number of columns to be printed on one page;
FORMATV: <format string>, print format of the variable labels;
FORMATC: <format string>, print format of the correlations;
TEXT: <string>, indentifying text, heading of the correlation matrix;
NVAR: <integer arithmetic expression>, number of variables;
NRESP: <integer arithmetic expression>, number of respondents;
SKIP: <arithmetic expression>, value indicating a missing observation;
READRESP: <procedure>, reads the data of one respondent;
LABELS: <procedure>, assigns integer labels to the variables.
CALLING SEQUENCE READRESP

Heading

"PROCEDURE" READRESP (J, VJ, NVAR, SKIP);
"VALUE" NVAR, SKIP; "REAL" VJ, SKIP; "INTEGER" J, NVAR;

Formal parameters
J: <integer variable>, Jensen parameter for VJ;
VJ: arithmetic expression>, function depending on the actual param-
eter for the Jensen variable J, to which the respondent's score on
the J-th variable is assigned;
NVAR: <integer arithmetic expression>, number of variables;
SKIP: <arithmetic expression>, value indicating a missing observation.

CALLING SEQUENCE LABELS

Heading

"PROCEDURE" LABELS (VARNR, NVAR); "VALUE" NVAR;
"INTEGER" NVAR; "INTEGER" "ARRAY" VARNR;

Formal parameters
VARNR: <integer array identifier>, to contain integer labels for the
variables;
NVAR: <integer arithmetic expression>, number of variables.

DATA AND RESULTS

The data are read by the procedure READRESP. If for any respondent a variable
has the value SKIP, this respondent is not used for the the calculation of the
correlations involving this variable. (pairwise deletion)

The following values (a.o.) of formats and number of columns are appropriate
to obtain a neat layout:

NCOL     FORMATV      FORMATC
15       "(""8A")"      "(""-ZIV50")"
21       "(""B4AB")"    "(""-ZV40")"
         "(""52B")"      "(""B=0.0D0")"

Make sure, that (NCOL+1)*WIDTH <=133, where WIDTH is the number of the
positions over which matrix elements and variable labels have to be printed.

The value .4444...... is printed if a correlation coefficient cannot be com-
puted.

The following error message may appear:
Errornumber 1 (if CHN <0 or CHN=60)
PROCEDURES USED
AVAILABLE STATAL 11011
DUPVEV NUMAL 31030
DUPVECCOL NUMAL 31033
STATAL3 ERROR STATAL 40100
OPEN SCRATCH STATAL OSCR
CLOSE SCRATCH STATAL CSCR

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
Let \( x \) and \( y \) denote two variables whose observations are given in two columns of the data matrix. Their correlation is computed as follows:

\[
\begin{align*}
R(x, y) &= (sx - x \cdot sy/n) / \\
& \quad \sqrt{(sx - x \cdot sx/n) \cdot (sy - y \cdot sy/n)}.
\end{align*}
\]

Where \( n \) is the number of cases that scored the value SKIP on neither \( x \) nor \( y \), \( sx \) the sum of the \( x \)-observations, \( sy \) the sum of the \( y \)-observations, \( sx \) the sum of the squares of the \( x \)-observations, \( syy \) the sum of the squares of the \( y \)-observations and \( sxy \) the sum of the products of the \( x \)-observation and the \( y \)-observation.

The correlation matrix is temporarily stored on extended core storage, the data are written to scratch file. The procedure uses a lot of input and output time because of drum-transports.

If the data matrix does not contain missing observations one is advised to use the procedure MINICORMAT (section 2.2.3.2.), which is considerably faster.

EXAMPLE OF USE

Program:

"BEGIN"

"PROCEDURE" LABELS(VAR, M);
"VALUE" M; "INTEGER" M; "INTEGER" "ARRAY" VAR;
"BEGIN" "END" THIS IS A DUMMY PROCEDURE;

"PROCEDURE" READRESP(J, VJ, M, SKIP);
"VALUE" M, SKIP; "INTEGER" J, M; "REAL" VJ, SKIP;
"BEGIN" EOF(6, OUT) J:= 1; INPUT(6, "(""B30")", VJ);
J:= 2; INPUT(6, "(""S30")", VJ);
"FOR" J:=3 "STEP" 1 "UNTIL" M "DO"
INPUT(6, "(""-ZD")", VJ);
J:= 5; "IF" VJ = SKIP "THEN"
"BEGIN"
"FOR" J:=1 "STEP" 1 "UNTIL" M "DO" VJ:= SKIP;
"GOTO" OUT

249
"END" LISTWISE DELETION;
J:= 8; "IF" VJ = 0 "THEN" VJ:= SKIP;
J:= 1; "IF" VJ = 123 ! VJ = 125 "THEN" VJ:= 124;
INPUT(6, "(""/""""));
OUT:
"END";

"COMMENT" THE PROCEDURE READRESP READS THE DATA OF A
RESPONDENT FROM CHANNEL 6. THE FIRST VARIABLE IS GIVEN
IN FORMAT B3D, THE SECOND ONE IN FORMAT B2D AND THE
OTHER ONES IN FORMAT -2D. IF A Respondent SCORES THE
SKIP VALUE ON VARIABLE 5, THE OTHER DATA OF THIS
RESPONDENT ARE NOT USED IN THE CALCULATION OF THE
CORRELATIONS (LISTWISE DELETION).
VARIABLE 8 HAS 4 SKIP VALUES. THE SCORES 123 AND
125 ON VARIABLE 1 ARE CHANGED TO 124;

CHANNEL(6, "(""E""""", 60);
CHANNEL(71, "(""E""""", 61);
CORMAT(71, 8, "(""SZB""""", "(""-ZV4D""""", "(""EXAMPLE""""",
8, 17, -1, READRESP, LABELS);
"END"

Input:

232 23 5 7 4 2 4 6
182 14 5 8 3 2 4 0
190 15 6 -1 3 8 5 6
123 2 5 7 3 7 3 7
184 16 3 8 2 2 3 2
194 17 4 7 2 4 4 0
220 17 5 8 -1 5 5 5
174 7 5 7 3 5 3 4
222 16 6 4 4 1 4 4
154 13 4 5 3 2 -1 2
123 6 5 -1 3 0 3 0
142 3 6 9 3 4 4 4
215 18 6 6 3 3 5 -1
156 5 6 9 3 3 3 2
178 12 4 6 3 4 3 3
125 6 5 8 4 5 4 2
156 6 7 2 4 8 4 2
Output:

Example

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8361</td>
<td>10000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6632</td>
<td>2394</td>
<td>10000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4888</td>
<td>1126</td>
<td>6750</td>
<td>10000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6631</td>
<td>3487</td>
<td>8324</td>
<td>5429</td>
<td>10000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1813</td>
<td>-0800</td>
<td>4068</td>
<td>5545</td>
<td>2612</td>
<td>10000</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8102</td>
<td>6111</td>
<td>8328</td>
<td>6383</td>
<td>7249</td>
<td>4323</td>
<td>10000</td>
</tr>
<tr>
<td>8</td>
<td>2202</td>
<td>0946</td>
<td>3817</td>
<td>-1169</td>
<td>2137</td>
<td>5133</td>
<td>3482</td>
</tr>
</tbody>
</table>

Source text

"CODE" 44410;
"PROCEDURE" CORMAT(CHN, NROFCOL, FORMATVARLABELS, FORMATMATRIEXELEMENTS, TEXT, ORDER, NRESP, SKIP, READRESP, LABELS);
"VALUE" CHN, NROFCOL, ORDER, NRESP, SKIP; "REAL" SKIP;
"INTEGER" CHN, NROFCOL, ORDER, NRESP;
"STRING" FORMATVARLABELS, FORMATMATRIEXELEMENTS, TEXT;
"PROCEDURE" READRESP, LABELS;
"BEGIN" "INTEGER" I, J, K, LOW, UPP, MAX, FILENR,
    RESPCOUNTER, PAGE, NRESP IN CORE,
    NRESP READ, MINADDR, CORSPACE, DATASPACE,
    MEMORY, SUMSPACE, ORDER MIN I, ADDR, OLDDP;

"INTEGER" "ARRAY" VAR LABELS[1:ORDER];
"BOOLEAN" LPP60, COR IN CORE, DATA IN CORE, FEW RESP,
    SUMS IN CORE, SCRATCH USED;

"PROCEDURE" HEADLINE;
"BEGIN" PAGE := PAGE + 1;
    OUTPUT(CHN, "("*,",")", TEXT);
    SYSPARAM(CHN, 2, 71);
    OUTPUT(CHN, "("*,"PAGE:.", 32D/",")", PAGE)
"END;"

"PROCEDURE" HEADING;
"BEGIN" "INTEGER" I; OUTPUT(CHN, "/")
    OUTPUT(CHN, FORMATVARLABELS, 0);
    OUTPUT(CHN, "/")
    "FOR" I := LOW "STEP" 1 "UNTIL" UPP "DO"
    OUTPUT(CHN, FORMATVARLABELS, VAR LABELS[I]);
"END;"

"PROCEDURE" NEWPAGE;
"IF" LPP60 "THEN"
"BEGIN" HEADING; HEADING;
       OUTPUT(CHN, "("/")") "END";
"IF" CHN <= 0 "OR" CHN = 60 "THEN"
       STATAL3 ERROR("CORMAT"); 1, CHN);
       SYSPARAM(CHN, 7, OLDP); LPP60 := OLDP = 66;
"IF" LPP60 & CHLENGTH(TEXT) > 70 "THEN"
       SYSPARAM(CHN, 8, 61 + (CHLENGTH(TEXT) - 71) // 135);
"FOR" J := 1 "STEP" 1 "UNTIL" ORDER
"DO" VAR LABELS[I] := J;
       LABELS(VAR LABELS, ORDER);
       ORDER MIN 1 := ORDER - 1;
       CORSPACE := ORDER * (ORDER + 1) // 2;
       DATASPACE := NRESP * ORDER;
       FEW RESP := NRESP < 2.5 * ORDER MIN 1;
       SUMSPACE := CORSPACE - ORDER;
       DATA IN CORE := SUMS IN CORE := "TRUE";
       MEMORY := AVAILABLE - 400 - CORSPACE - .25 * DATASPACE;
       COR IN CORE := MEMORY > 0;
"IF" FEW RESP "THEN"
"BEGIN" MEMORY := MEMORY - .75 * DATASPACE -
       2 * (NRESP + ORDER);
       DATA IN CORE := MEMORY > 0
"END" "ELSE"
"BEGIN" MEMORY := MEMORY - 5 * SUMSPACE - 8 * ORDER + 5;
       SUMS IN CORE := MEMORY > 0;
"END";

SCRATCH USED := "NOT" (DATA IN CORE "AND" SUMS IN CORE);
"IF" SCRATCH USED
"THEN" FILENR := OPEN SCRATCH("SC44410");

"IF" "NOT" COR IN CORE "THEN"
       MINADDR := "IF" FEW RESP
       "THEN" DATASPACE "ELSE" 5 * SUMSPACE;

"BEGIN" "ARRAY" COR[I] := "IF" COR IN CORE
       "THEN" CORSPACE "ELSE" 1;

"IF" FEW RESP "THEN"
"BEGIN" "ARRAY" DATA[I] := "IF" DATA IN CORE
       "THEN" NRESP "ELSE" 1,
       1 := "IF" DATA IN CORE
       "THEN" ORDER "ELSE" 1;

"IF" DATA IN CORE "THEN"
"BEGIN" "FOR" I := 1 "STEP" 1 "UNTIL" NRESP "DO"
       READRESP(J, DATAII, JJ, ORDER, SKIP);
"END" "ELSE"
"BEGIN" NRESP IN CORE := (AVAILABLE - 400 - NRESP)
       // ORDER;
       NRESP READ := 0;
"FOR" NRESP IN CORE:=
  "IF" NRESP IN CORE > NRESP "THEN" NRESP
  "ELSE" NRESP IN CORE, NRESP - NRESP READ "DO"
  "IF" NRESP IN CORE > 0 "THEN"
  "BEGIN"
  "REAL" "ARRAY" BUFFER[1:NRESP IN CORE, 1:ORDER],
  COLUMN[1:NRESP IN CORE];
  "FOR" NRESP READ:= NRESP READ
  "STEP" NRESP IN CORE
  "UNTIL" NRESP = NRESP IN CORE "DO"
  "BEGIN"
  "FOR" I:= 1 "STEP" 1
  "UNTIL" NRESP IN CORE "DO"
  READRESP(J, BUFFER[I, J], ORDER, SKIP);
  "FOR" J:= 1 "STEP" 1 "UNTIL" ORDER "DO"
  "BEGIN"
  DUPVECCOL(1, NRESP IN CORE, J,
  COLUMN, BUFFER);
  ADDR:= (J-1) * NRESP + NRESP READ
  + 1;
  STORE ARRAY(FILENR, ADDR, COLUMN);
  "END"
  "END"
  "END" READ DATA;
  "BEGIN" "REAL" XI, XJ, SI, SJ, SII, SJJ, SJJ, DENOM;
  "ARRAY" VARI, VARI[1 : NRESP];
  "INTEGER" ADDR1, ADDR2;
  ADDR2:= MINADDR + 1;
  "IF" COR IN CORE "THEN" COR[I]:= 1 "ELSE"
  STORE ITEM(FILENR, ADDR2, 1.0);
  ADDR1:= 1;
  "FOR" J:= 2 "STEP" 1 "UNTIL" ORDER "DO"
  "BEGIN" "INTEGER" PJ; "ARRAY" CORREL[1:J];
  CORREL[J]:= 1; "IF" DATA IN CORE "THEN"
  DUPVECCOL(1, NRESP, J, VARJ, DATA) "ELSE"
  FETCH ARRAY(FILENR, ADDR1, VARJ);
  ADDR1:= 1;
  "FOR" I:= 2 "STEP" 1 "UNTIL" J "DO"
  "BEGIN" SI:= SJJ:= SII:= SJ:= SJJ:= 0;
  REPCOUNTER:= NRESP;
  "IF" DATA IN CORE "THEN"
  DUPVECCOL(1, NRESP, I - 1, VARI, DATA)
  "ELSE" FETCH ARRAY(FILENR, ADDR1, VARI);
  "FOR" K:= 1 "STEP" 1 "UNTIL" NRESP "DO"
  "BEGIN" XI:= VARJ[K]; XJ:= VARJ[K];
  "IF" XJ X= SKIP & XJ X= SKIP "THEN"
  "BEGIN" SI:= SI + XI; SJ:= SJ + XJ;
  SII:= SII + XI * XI;
  SJJ:= SJJ + XI * XJ;
SJ1:= SJ1 + XJ * XJ
"END"
"ELSE" RESPCOUNTER:= RESPCOUNTER - 1
"END"

DENOM:= (RESPCOUNTER * SII - SI * SI) *
       (RESPCOUNTER * SJJ - SJ * SJ);

CORREL[I - 1]:= "IF" DENOM < "-10
                   "THEN" 2 "ELSE"
                   (RESPCOUNTER * SIJ - SI * SJ)
                   / SQRT(DENOM)
                   "END"
PJ:= J * (J - 1) / 2;
"IF" COR IN CORE "THEN"
DUPVEC(PJ + 1, PJ + J, -PJ, COR, CORREL)
"ELSE" STORE ARRAY(FILENR, ADDR2, CORREL)
"END"
"END" BUILD UP CORRELATION-MATRIX;
"END" "ELSE"

"BEGIN" "ARRAY" SOM1Z, SOM21, KSOM1Z, KSOM21, COUNT1 :
"IF"SUMS IN CORE "THEN" SUMSPACE "ELSE" 13;

"INTEGER" CORPOINTER, POINTER1, N NOT MISS, VAR1, VAR2, VAR11, ADDR1,
       ADDR2, AAMT RESP;
"REAL" SCORE1, SCORE2, SI1, S21, DENOM;

"IF" SUMS IN CORE "THEN"
"BEGIN" "FOR" I:= 1 "STEP" 1 "UNTIL" SUMSPACE "DO"
       "BEGIN"
       COR[I]:= SOM1Z[I]:= SOM21[I]:= KSOM1Z[I]:= KSOM21[I]:= 0;
       COUNT[I]:= 0;
       "END"
       "FOR" I:= SUMSPACE + 1 "STEP" 1 "UNTIL" CORSPACE "DO"
       "COR[I]:= 0"
       "END" "ELSE"
       "BEGIN" "ARRAY" ZERO1 : ORDER1; "INTEGER" TIMES1;
       TIMES1:= 3 * ORDER MIN 1; ADDR:= 1;
       "FOR" I:= 1 "STEP" 1 "UNTIL" ORDER "DO"
       "ZERO[I]:= 0"
       "FOR" I:= 0 "STEP" 1 "UNTIL" TIMES1 "DO"
       STORE ARRAY(FILENR, ADDR, ZERO)
       "END" INITIALIZE;

"BEGIN" "ARRAY" SCORES[1 : ORDER];
"INTEGER" "ARRAY" NOT MISSING[1 : ORDER];

"FOR" RESPINDEX:= 1 "STEP" 1 "UNTIL" NRESP "DO"
"BEGIN" READRESP(I, SCORES[I], ORDER, SKIP);
       N NOT MISS:= POINTER:= 0;
       "IF" SCORES[I] NOT SKIP "THEN"

254
"BEGIN" N NOT MISS:= 1;
   NOT MISSING[1]:= 1 "END";
COR POINTER:= ADDR1:= 1;
ADDR2:= MINADDR + 2;
"FOR" VAR1:= 2 "STEP" 1 "UNTIL" ORDER "DO"
"BEGIN" SCORE1:= SCORES[VAR1];
VAR11:= VAR1 - 1;
"IF" SCORE1 = SKIP "THEN"
"BEGIN" "ARRAY" CORREL[1 : VAR11],
   SUM12, SUM21, KSUM12, KSUM21,
   RCOUNT[1 : VAR11];
"IF" COR IN CORE "THEN"
   DUPVEC(1, VAR1, COR POINTER,
      CORREL, COR)
"ELSE"
   FETCH ARRAY(FILENR, ADDR2, CORREL);
"END" "ELSE"
   FETCH ITEM(FILENR, ADDR1, SUM12,
      SUM21, KSUM12, KSUM21, RCOUNT);
"END" "FOR"

"FOR" I:= 1 "STEP" 1 "UNTIL" N NOT MISS "DO"
"BEGIN" VAR2:= NOT MISSING[1];
SCORE2:= SCORES[VAR2];
SUM12[VAR2]:= SUM12[VAR2] +
   SCORE1;
SUM21[VAR2]:= SUM21[VAR2] +
   SCORE2;
KSUM12[VAR2]:= KSUM12[VAR2] +
   SCORE1 * SCORE1;
KSUM21[VAR2]:= KSUM21[VAR2] +
   SCORE2 * SCORE2;
RCOUNT[VAR2]:= RCOUNT[VAR2] + 1;
CORREL[VAR2]:= CORREL[VAR2] +
   SCORE1 * SCORE2;
"END" FOR I;
N NOT MISS:= N NOT MISS + 1;
NOT MISSING[1 NOT MISS]:= VAR1;
"IF" COR IN CORE "THEN"
   DUPVEC(COR POINTER + 1,
COR POINTER + VAR1,
-COR POINTER, COR, CORREL)
"ELSE"
"BEGIN" ADDR2:= ADDR2 - VAR1;
STORE ARRAY(FILENR, ADDR2,
CORREL);
"END";

"IF" SUMS IN CORE "THEN"

"BEGIN"
DUPVEC(PATR + 1,
POINTER + VAR11,
-POINTER, XOM12, SUM12);
DUPVEC(PATR + 1,
POINTER + VAR11,
-POINTER, XOM21, SUM21);
DUPVEC(PATR + 1,
POINTER + VAR11,
-POINTER, XOM12, KSUM12);
DUPVEC(PATR + 1,
POINTER + VAR11,
-POINTER, XOM21, KSUM21);
DUPVEC(PATR + 1,
POINTER + VAR11,
-POINTER, COUNT, RCOUNT);
"END" "ELSE"
"BEGIN" ADDR1:= ADDR1 - 5 * VAR11;
STORE ITEM(FILENR, ADDR1,
SUM12, SUM21, KSUM12,
KSUM21, RCOUNT);
"END";

"END" IF SCORE1 ≈ SKIP;

COR POINTER:= COR:pointer + VAR1;
POINTER:= POINTER + VAR11
"END" FOR VAR1;
"END" FOR RESPONDENT;
"END" BUILD UP BLOCK;

POINTER:= 0; ADDR1:= COR:pointer:= 1;
ADDR2:= MINADDR + 1;
"IF" COR IN CORE "THEN" COR[1]:= 1
"ELSE" STORE ITEM(FILENR, ADDR2, 1.0);
"FOR" VARI:= 2 "STEP" 1 "UNTIL" ORDER "DO"
"BEGIN" "ARRAY" CORREL[1 : VARI],
SUM12, SUM21, KSUM12, KSUM21, RCOUNT[1 : VARI - 1];
VARI1:= VARI - 1;

"IF" COR IN CORE "THEN"
DUPVEC(1, VAR1, COR POINTER, CORREL, COR)
"ELSE"
FETCH ARRAY(FILENR, ADDR2, CORREL);

"IF" SUMS IN CORE "THEN"
"BEGIN"
  DUPVEC(1, VAR11, POINTER, SUM12, SOM12);
  DUPVEC(1, VAR11, POINTER, SUM21, SOM21);
  DUPVEC(1, VAR11, POINTER, KSUM12, KSOM12);
  DUPVEC(1, VAR11, POINTER, KSUM21, KSOM21);
  DUPVEC(1, VAR11, POINTER, RCOUNT, COUNT)
"END" "ELSE"
FETCH ITEM(FILENR, ADDR1, SUM12, SUM21, KSUM12, KSUM21, RCOUNT);

"FOR" VAR2:= 1 "STEP" 1 "UNTIL" VAR11 "DO"
"BEGIN" AANT RESP:= RCOUNT[VAR2];
   S12:= SUM12[VAR2]; S21:= SUM21[VAR2];
   DENOM:= (AANT RESP * KSUM12[VAR2] - S12 * S12) * (AANT RESP *
               KSUM21[VAR2] - S21 * S21);
   CORREL[VAR2]:= "IF" DENOM < "-10"
     "THEN" 2 "ELSE"
       (AANT RESP * CORREL[VAR2] - S12 * S21) / SQRT(DENOM);

"END" FOR VAR2;
CORREL[VAR11]:= 1;

"IF" COR IN CORE "THEN"
DUPVEC(COR POINTER + 1, COR POINTER + VAR1,
   -COR POINTER, COR, CORREL)
"ELSE"
"BEGIN" ADDR2:= ADDR2 - VAR1;
   STORE ARRAY(FILENR, ADDR2, CORREL);
"END";

"IF" SUMS IN CORE "THEN"
"BEGIN" DUPVEC(POINTER + 1, POINTER + VAR11,
   -POINTER, SOM12, SUM12);
   DUPVEC(POINTER + 1, POINTER + VAR11,
   -POINTER, SOM21, SUM21);
   DUPVEC(POINTER + 1, POINTER + VAR11,
   -POINTER, KSOM12, KSOM12);
   DUPVEC(POINTER + 1, POINTER + VAR11,
   -POINTER, KSOM21, KSOM21);
   DUPVEC(POINTER + 1, POINTER + VAR11,
   -POINTER, COUNT, RCOUNT);
"END" "ELSE"
"BEGIN" ADDR1:= ADDR1 - 5 * VAR11;
   STORE ITEM(FILENR, ADDR1, SUM12, SUM21,
               KSUM12, KSUM21, RCOUNT);
"END";
COR POINTER := COR POINTER + VAR1;
POINTER := POINTER + VAR11;
"END" FOR VAR1;
"END" IF FEW RESP;

SYSPARAM(chn, 3, I);
"IF" I > 0 "THEN" OUTPUT(61, "("*")");
OUTPUT(chn, "("*")", TEXT); SYSPARAM(chn, 2, 71);
"IF" ORDER > NR OF COL "THEN"
OUTPUT(chn, "(""PAGE: 1")", "/")"); PAGE := 1;
UPP := 0; "FOR" LOW := UPP + 1 "WHILE" UPP < ORDER "DO"
"BEGIN" K := UPP - UPP // 50 * 50;
UPP := LOW + NROFCOL - 1;
SYSPARAM(chn, 4, K // 10 + K + 1);
"IF" UPP > ORDER "THEN" UPP := ORDER;
HEADING; OUTPUT(chn, "/")");
"FOR" J := LOW "STEP" 1 "UNTIL" ORDER "DO"
"BEGIN" OUTPUT(chn, "("/")");
OUTPUT(chn, FORMATVARLABELS, VAR LABELS(J));
OUTPUT(chn, "(""2B")");
MAX := "IF" J < UPP "THEN" J "ELSE" UPP;
"BEGIN" "REAL" "ARRAY" CORREL(LOW:MAX);
"IF" COR IN CORE "THEN"
DUVEC(LOW, MAX, J * (J-1) / 2, CORREL, COR)
"ELSE"
"BEGIN"
ADDR := (J - 1) * J / 2 + LOW + MINADDR;
FETCH ARRAY(FILNBR, ADDR, CORREL);
"END";
"FOR" I := LOW "STEP" 1 "UNTIL" MAX "DO"
OUTPUT(chn, FORMATMATRIEXELEMENTS,
"IF" ABS(CORREL[I]) < 1.000005
"THEN" CORREL[I]
"ELSE" 40 / 9);

"IF" J // 10 * 10 = J "THEN"
"BEGIN" OUTPUT(chn, "/")");
"IF" J // 50 * 50 = J & J < ORDER
"THEN" NEWPAGE
"END"
"END"
"END";

"IF" ORDER // 10 * 10 = ORDER
"THEN" OUTPUT(chn, "("/")");
HEADING; "IF" UPP < ORDER "THEN" HEADLINE;
"END"
"END" COR-BLOK;

EXIT: "IF" SCRATCH USED "THEN" CLOSE SCRATCH(FILNBR);
SYSPARAM(chn, 8, OLDPP);
"END" CORMAT;
"EOP"
TITLE: Minicormat

AUTHOR: R. Kaas

INSTITUTE: Mathematical Centre

RECEIVED: 750315

BRIEF DESCRIPTION
For a data matrix consisting of NRESP rows, each row representing the observations of one respondent on NVAR variables, the procedure computes the matrix of product-moment correlation coefficient and prints its lower triangular part. Control information about each variable is given: mean, standard deviation, minimum and maximum. Optionally the correlation matrix is written to a file. The data matrix is read by a procedure called READRESP, which has to be provided by the user elsewhere in the program. No observations are supposed to be missing. If the data matrix contains missing observations, the procedure CORMAT (section 2.2.3.1.) can be used.

KEYWORDS
Product-moment correlation matrix

CALLING SEQUENCE
Heading
"PROCEDURE" MINICORMAT (CHN, NVAR, NRESP, READRESP, TOFILE);
"VALUE" CHN, NVAR, NRESP, TOFILE;
"INTEGER" CHN, NVAR, NRESP;
"PROCEDURE" READRESP;
"BOOLEAN" TOFILE;
"CODE" 44411;
Formal parameters

CHN: <integer arithmetic expression>, channel number via which the output is written to file;

NVAR: <integer arithmetic expression>, number of variables;

NRESP: <integer arithmetic expression>, number of respondents;

READRESP: <procedure>, reads the data matrix;

TOFILE: <boolean expression>, to indicate whether the correlation matrix should be written to file or not.

CALLING SEQUENCE READRESP
Heading
"PROCEDURE" READRESP (DATA); "ARRAY" DATA;
Formal parameter

DATA: <array identifier>, output parameter, a two dimensional array DATA[1: NRESP, 1: NVAR], which at exit contains the data.
DATA AND RESULTS
The data are read by a call of the procedure READRESP. MINICORMAT writes
minimum, maximum, mean and standard deviation of each variable and the
lower triangular part of the correlation matrix to the file linked to channel CHN.
The correlations are printed as integers, after being multiplied by 100000. The
value 40/9 is printed if a correlation cannot be computed. If Tofile has the
value "true", the procedure writes the number of variables (in format 3ZD)
and the lower triangular part of the correlation matrix (in format -D.5D2B) via
channel 65 to file.
The following error message may appear:
Error number 1 (if CHN <0 or CHN = 60)

PROCEDURES USED
TAMMAT
NUMAL 34014
STATA3 ERROR
STATAL 40100

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
Let \( x \) and \( y \) denote two variables whose observations are given in two columns
of the data matrix. Their correlation is computed as follows:
\[
R(x, y) = \frac{(sx \cdot sy) - (nresp) \cdot (sxy)}{\sqrt{(sx^2 - nresp \cdot sx) \cdot (sy^2 - nresp \cdot sy)}}
\]
where \( sx \) is the sum of the \( x \)-observations, \( sy \) the sum of the \( y \)-observations,
\( sxx \) the sum of the squares of the \( x \)-observations, \( syy \) the sum of the squares
of the \( y \)-observations and \( sxy \) the sum of the products of the \( x \)-observation
and the \( y \)-observation. The crossproduct sums \( sxx \), \( sxy \) and \( syy \) are com-
puted by calls of the procedure TAMMAT.

The procedure declares an array DATA[1: NRESP, 1: NVAR], an array
CORR[1: NVAR*(NVAR+1)/2], and two arrays of length NVAR.

EXAMPLE OF USE
Program:
"BEGIN" "PROCEDURE" READRESP(DATA); "ARRAY" DATA;
INARRAY(60, DATA);
MINICORMAT(61, 4, 10, READRESP, "FALSE")
"END"
Minicormat

Input:

1295 683 262 190
1330 721 295 200
1348 687 275 195
1371 703 272 175
1389 719 280 185
1349 724 275 192
1327 683 275 176
1323 704 292 194
1290 700 270 190
1381 708 295 198

Output:

CONTROL INFORMATION

======================

NUMBER OF VARIABLES: 4
NUMBER OF CASES: 10

VAR. MEAN S.D. MIN. MAX.
1 1340.300 32.047 1290.000 1389.00C
2 703.200 14.531 683.000 724.000
3 279.100 10.719 262.000 295.000
4 189.500 8.078 175.000 200.000

CORRELATIONMATRIX (CORRELATIONS MULTIPLIED BY 100000)

1 2 3 4
1 100000
2 46435 100000
3 38011 52890 100000
4 -11917 32970 54110 100000

Source text

"CODE" 44411;
"PROCEDURE" MINICORMAT(CHN, ORDER, NRESP, READRESP,
COMAT TO FILE);
"VALUE" CHN, ORDER, NRESP, COMAT TO FILE;
"INTEGER" CHN, ORDER, NRESP; "BOOLEAN" COMAT TO FILE;
"PROCEDURE" READRESP;
"BEGIN" "INTEGER" I, J, K, LOW, UPP, MAX, PJ;
"REAL" SUMJ, DEVJ, DIJ, MINJ, MAXJ;
"ARRAY" DATA[1:NRESP, 1:ORDER],
CORREL[1:ORDER * (ORDER + 1) / 2],
DATASUM, DATADEV[1:ORDER];
"PROCEDURE" HEADING;
"BEGIN" "INTEGER" I; OUTPUT(CHN, "(\text{"/\text{AB}\text{\text{'}}}\text{"})\text{"});
 "FOR" I:= LOW "STEP" 1 "UNTIL" UPP "DO"
 OUTPUT(CHN, "(\text{"5ZDBB\text{"}}\text{")}, I)
 "END" HEADING;

"IF" CHN <= 0 "OR" CHN = 60 "THEN"
 STATAL3 ERROR("("MINICORMAT")", 1, CHN);
 OUTPUT(CHN, "("("CONTROL INFORMATION")\text{"}, /19("("\text{"\text{"})\text{"},
 //("NUMBER OF VARIABLES")\text{"}, ZD3D, /,
 (("NUMBER OF CASES")\text{"}, 3ZD, /,
 "("VAR. MEAN S.D. MIN. MAX.")\text{"})\text{"},
 ORDER, NRESP);
 "IF" CORMAT TO FILE
 "THEN" OUTPUT(65, "("ZD3D, /"), ORDER);
 READRESP(DATA);

 "FOR" J:= 1 "STEP" 1 "UNTIL" ORDER "DO"
 "BEGIN" SUMJ:= MINJ:= MAXJ:= DATA[J, J];
 "FOR" I:= 2 "STEP" 1 "UNTIL" NRESP "DO"
 "BEGIN" DIJ:= DATA[I, J]; SUMJ:= SUMJ + DIJ;
 "IF" DIJ < MINJ "THEN" MINJ:= DIJ "ELSE"
 "IF" DIJ > MAXJ "THEN" MAXJ:= DIJ
 "END";
 DATASUM[J, J]:= SUMJ;
 DEVJ:= TAMMAT1, NRESP, J, J, DATA, DATA) * NRESP -
 SUMJ * SUMJ;
 DATASDEV[J, J]:= DEVJ:= "IF" DEVJ <= 0
 "THEN" 0 "ELSE" SQR(DEVJ);
 PJ:= J * (J - 1) / 2; CORREL[PJ + J]:= 1;
 "IF" DEVJ = 0 "THEN"
 "BEGIN"
 "FOR" I:= J - 1 "STEP" -1 "UNTIL" 1 "DO"
 CORREL[PJ + I]:= 2
 "END" "ELSE"
 "FOR" I:= J - 1 "STEP" -1 "UNTIL" 1 "DO"
 CORREL[PJ + I]:= "IF" DATASDEV[I, I, J, J, DATA, DATA) * NRESP -
 DATASUM[I, J] * SUMJ) / (DATASDEV[I] * DEVJ);
 OUTPUT(CHN, "("ZD3D, 5ZD3D, 3ZD3D")\text{"},
 J, SUMJ / NRESP, DEVJ / NRESP, MINJ, MAXJ);
 "END" FOR J AND OF COMPUTATION PART;
 UPP:= 0; SYSPARAM(CHN, 8, 0);
 "FOR" LOW:= UPP + 1 "WHILE" UPP <= ORDER "DO"
 "BEGIN" K:= UPP - UPP // 50 = 50; UPP:= LOW + 15;
 "IF" UPP > ORDER "THEN" UPP:= ORDER;
 OUTPUT(CHN, "("("1CORRELATIONMATRIX")\text{"}, 10B,
 "("("CORRELATIONS MULTIPLIED BY 10000")\text{"})\text{"},
 SYSPARAM(CHN, 4, K // 10 + K + 1);
 HEADING; OUTPUT(CHN, "("")\text{"},
 "FOR" J:= LOW "STEP" 1 "UNTIL" ORDER "DO"
 "BEGIN" OUTPUT(CHN, "("ZD3DBB", J);
 PJ:= J * (J - 1) / 2;
 MAX:= "IF" J < UPP "THEN" J "ELSE" UPP;
"FOR" I:= LOW "STEP" 1 "UNTIL" MAX "DO"
"IF" ABS(CORREL(PJ + I)) > 1.00005 "THEN"
 OUTPUT(CHN, "(""(" "----- "\")")") "ELSE"
 OUTPUT(CHN, "(""(""-ZV5DB")", CORREL(PJ + I));
"IF" J // 10 * 10 = J
 "THEN" OUTPUT(CHN, "(""/""))";
"END";
"IF" ORDER//10+10 < ORDER
 "THEN" OUTPUT(CHN,"("="/")")";
HEADING;
"END";
OUTPUT(CHN, "("="/,"(""\")")")");
"IF" CORMAT TO FILE
 "THEN" OUTPUT65, "(""-D.5DBB")", CORREL)
"END" MINI CORMAT;
"EOP"
2.2.3.3

**Title:** Joreskog

**Authors:** R. Wiggers, E. Opperdoes, J. Rijvordt

**Institute:** Mathematical Centre

**Received:** 740906

**Brief Description**
The procedure performs a factor analysis with a user specified number of factors under the assumption that the model underlying the data is the model of Joreskog (see Joreskog, 1963). The factors of the canonical solution are varimax rotated. Input may be a raw data matrix, a covariance matrix or a correlation matrix. The data matrix is read by a procedure called **READRESP**, which has to be provided by the user elsewhere in the program. If requested, factorscores and fac torscore coefficients are computed.

**Keywords**
Joreskog factor analysis

**Calling Sequence**

**Calling Sequence**

**Heading**

"PROCEDURE" JORESKOG (EPS, READRESP, LABELS);
"VALUE" EPS;
"REAL" EPS;
"PROCEDURE" READRESP, LABELS;
"CODE" 44400;

**Formal parameters**

EPS: <arithmetic expression>, **ABS (EPS)** is the precision of the rotation, **SIGN (EPS)** indicates whether channel cards must be read or not (see DATA AND RESULTS below);

READRESP: <procedure>, reads the data of one respondent;

LABELS: <procedure>, assigns integer labels to the variables and respondents.

**Calling Sequence** **READRESP**:

**Heading**

"PROCEDURE" READRESP (J, VJ, NVAR, SKIP);
"VALUE" NVAR, SKIP; "REAL" VJ, SKIP; "INTEGER" J, NVAR;

**Formal parameters**

J: <integer variable>, Jensen parameter for VJ;

VJ: <arithmetic expression>, function depending on the actual parameter for the Jensen variable J, to which the respondent's score on the J-th variable is assigned;

NVAR: <integer arithmetic expression>, number of variables;

SKIP: <arithmetic expression>, value indicating a missing observation.
If the input is a correlation or covariance matrix, **READRES** is not used and may be a dummy procedure.

**CALLING SEQUENCE LABELS:**

**Heading**

"PROCEDURE" LABELS (VARNR, NVAR, RESPNR, NRESP);
"VALUE" NVAR, NRESP; "INTEGER" NVAR, NRESP;
"INTEGER" "ARRAY" VARNR, RESPNR;

**Formal parameters**

VARNR:  
<integer array identifier>, to contain integer labels for the variables;

NVAR: 
<integer arithmetic expression>, number of variables;

RESPNR:  
<integer array identifier>, to contain integer labels for the respondents;

NRESP:  
<integer arithmetic expression>, number of respondents.

The procedure body of **LABELS** overwrites **VARNR** and **RESPNR**.

The procedure **JORESKOG** fills these array's with the integers 1, 2, 3, 4,... before a call of **LABELS**, so an empty procedure body is equivalent to:

"BEGIN"  
"INTEGER" I;
"FOR" I:=1 "STEP"1 "UNTIL" NVAR "DO" VARNR[I]:=I;
"FOR" I:=1 "STEP"1 "UNTIL" NRESP "DO" RESPNR [I]:= I
"END"

If no printing of raw data, standardized data or factor scores is requested, an assignment to an element of the array **RESPNR** causes an array bounds error.

**DATA AND RESULTS**

The following channel cards are used:

- **CHANNEL**, 62: input channel for covariance or correlation matrix
- **CHANNEL**, 63: input channel for control information
- **CHANNEL**, 64: output channel
- **CHANNEL**, 1: raw data
- **CHANNEL**, 2: covariance matrix
- **CHANNEL**, 3: correlation matrix
- **CHANNEL**, 4: inverse of the covariance or correlation matrix
- **CHANNEL**, 5: standardized data (optional)
- **CHANNEL**, 6: reduced covariance matrix (if computed)
- **CHANNEL**, 7: reduced correlation matrix (if computed)
- **CHANNEL**, 8: factor scores (optional)

The channels 1, 2,..., 8 are binary sequential io channels, on which array's are dumped by the procedure using the CDC-ALGOL procedure **PUTARRAY** (see p. 8-15 of the CDC, Algol-60, r.m. version 5, 1979) These channels have to be given, but need not to be used.

If **EPS**<0, no channel cards are read and the channels 62, 63 and 64 are defined by
CHANNEL (62, "("C")", "("LFN")", "("CINFILE")", "("P")", 80);
CHANNEL (63, "("C")", "("LFN")", "("INFILE")", "("P")", 80);
CHANNEL (64, "("C")", "("LFN")", "("OUTFILE")", "("P")", 136,
"("PP")", 60);
and the channels 1,..., 8 by CHANNEL (...,"("B")");
if EPS>0, channel cards are read from channel 60 (from the file INPUT) which
makes it possible to change the default parameters of the channel cards or to
define new channels. A new channel must be defined by the user when a raw
data matrix is read. This channel must be unequal to 61, 62, 64, 1, 2, 3, 4, 5,
6, 7, or 8. (For example CHANNEL 65,"("C")", "("LFN")", "("RAWDATA")",
"("P")", 80).
The following input is read from channel 63:
1: Number of respondents. If negative, the raw data are printed.
2: Number of variables. If negative, the standardized data are printed. The
standardized data are computed by subtracting from each score the mean
of the variable and then diving the result by the standard deviation.
3: Integer code for the entered matrix:
   +1 or -1: raw data matrix
   +2 or -2: covariance matrix
   3: correlation matrix.
The covariance or correlation matrix must be offered to the procedure as
a lower triangular matrix, given row by row. If the matrix code is nega-
tive, the analysis of the covariance matrix is suppressed.
4: If the matrix code is +1 or -1:
   4.1:if observations are missing: value indicating a missing observation.
   4.2:if no observations are missing: "-7.
5: Number of latent roots and vectors to be computed. (see Joreskog, 1963,
p. 25).
6: Number of unrotated factors. The first (number of unrotated factors) fac-
tors are not rotated.
7: Number of analyses to be performed. If negative, the standardized data
are computed and written to channel 5.
8: For each analysis: number of factors. This number has to be < number
of latent roots and vectors, and > number of unrotated factors. If for
one of the analyses the number of factors is negative, the standardized
data, factorscore coefficients and factorscores are computed, while the
latter two are printed.
After reading this input the data are read, starting on a new line. A raw data
matrix is read via user selected channel, a covariance or correlation matrix is
read via channel 62, in which case the procedures READRESP and LABELS are
not used and may be dummy procedures.
The following error messages may appear:
"INSUFFICIENT CM FOR READING DATA"
"INSUFFICIENT CM FOR PRINTING <NAME>"
"EOF/EOR ENCOUNTERED ON CHANNEL 63,
<NP> RESPONDENTS HAVE BEEN READ"

In the first and in the third case execution is terminated. In the second case only the printing of <NAME> is omitted, <NAME> being "RAW DATA", "STANDARDIZED" or "FACTORSCORES" in the first and in the second case one has to increase the CM parameter. To avoid the third error one has to count the respondents again or check the procedure READRESP. (The third error may occur if the data are read in fixed field format and the format string ends with a "/".)

The following output is written to channel 64:
1: Control information table.
2: If matrix code is +1 or -1 and the code of the missing observations is unequal to "-7:"
   2.1: number of missing observations for each variable.
   2.2: number of respondents with corresponding number of missing observations.
3: Raw data, if requested. The missing observations of a variable indicated by the value of skip, are replaced by the mean of the non-missing observations of that variable.
4: Standardized data, if requested.
5: If matrix code is unequal to 3: covariance matrix.
6: Correlation matrix.
7: If covariance matrix is singular: reduced non-singular covariance matrix.
8: If correlation matrix is singular: reduced non-singular correlation matrix.
9: If matrix code is unequal to 3: diagonal of the inverse of the covariance matrix.
10: If matrix code = 3: Diagonal of the inverse of the correlation matrix.
11: Multiple correlation coefficient R and R².
13: Latent roots and vectors of $S^*$.
   For each analysis:
14: If matrix code = +1 or +2: analysis of covariance matrix.
   14.1: Matrix of canonical factorloadings.
   14.2: Guttman-criterion. (see Bethlehem et.al., 1977)
   14.3: If rotation is possible (i.e. if number of factors $\geq$ number of unrotated factors +2):
      14.3.1: transformation matrix.
      14.3.2: number of iterations.
      14.3.3: matrix of factorloadings after rotation.
      14.3.4: number of unrotated factors.
      14.3.5: Guttman-criterion after rotation.
14.4: Percentage of variance explained.
14.5: Communalities.
14.6: Residual covariance matrix.
15: Analysis of correlation matrix.
15.1: matrix of canonical factor loadings.
15.2: Guttman-criterion.
15.3: if rotation is possible:
   15.3.1: transformation matrix.
   15.3.2: number of iterations.
   15.3.3: matrix of factor loadings after rotation.
   15.3.4: number of unrotated factors.
   15.3.5: Guttman-criterion after rotation.
15.4: percentage of variance explained.
15.5: communalities.
15.6: residual correlation matrix.
16: If matrix code = +1 or -1 and number of factors < 0:
   16.1: matrix of factorscore coefficients.
17: Test of fit table with:
   17.1: mean of the (number of variables - number of factors) smallest latent roots of $s^*$.
   17.2: $u$-statistic with number of degrees of freedom.

PROCEDURES USED

EXIT NUMAL 11010
AVAILABLE NUMAL 11011
DATE NUMAL 11012
INIVEC NUMAL 31010
INIMAT NUMAL 31011
INIMATD NUMAL 31012
DUPVECCROW NUMAL 31031
DUPROMVEC NUMAL 31032
DUPVECCOL NUMAL 31033
COLCST NUMAL 31131
ROWCST NUMAL 31132
VECVEC NUMAL 34010
TAMVEC NUMAL 34012
MATMAT NUMAL 34013
TAMMAT NUMAL 34014
MATTAM NUMAL 34015
SEQVEC NUMAL 34016
ICHVEC NUMAL 34030
ICHSEQVEC NUMAL 34034
ICHSEQ NUMAL 34035
ROTCOL NUMAL 34040

269
EIGSYM1  NUMAL 34156
TRIANGLE  STATAL 40201
RECTANGLE  STATAL 40202
CHANNELCARDS  STATAL CCARDS

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
The way the covariance or correlation matrix of a raw data matrix is computed is dependent on the amount of central memory available and of the missing of observations. If requested, the standardized data or raw data are printed. After printing the covariance and correlation matrix the inverses of these matrices are computed, as well as the matrix $$S^*$$, its determinant, trace, eigenvalues and eigenvectors. Using this data the canonical factor solution is computed. If possible, this solution is varimax rotated (see Harman, 1970) the residual covariance or correlation matrix is computed. In case input was a raw data matrix the matrices of factorscore coefficients and of factorscores are computed, the latter using standardized data.

The NOS/BE-card LIMIT, 3000 is required, because of the number of files which the procedure JORESKOG uses. The minimum amount for central memory is 110000 (octal) words; 130000 (octal) is advised. Small problems, like the "EXAMPLE OF USE", need only T30. For large problems, often occurring in practice, at least T200 is required.

REFERENCES
[1] K. G. Joreskog,
Statistical estimation in factor analysis,
Almqvist and Wiksell, Uppsala, 1963.
Control Data Corporation, 1979.
[3] H.H. Harman,
Modern factor analysis,
Methoden, Voetangels en klemmen in de factoranalyse,
Mathematical Centre report SN 7/77,
Mathematical Centre, Amsterdam, 1977.
2.2.3.3

Example of Use

Program:

"BEGIN"
"PROCEDURE" READRESP(J, VJ, NVAR, SKIP);
"VALUE" NVAR, SKIP; "INTEGER" J, NVAR; "REAL" VJ, SKIP;
"COMMENT" DUMMY PROCEDURE; ;

"PROCEDURE" LABELS(VARNR, NVAR, RESPNR, NRESP);
"VALUE" NVAR, NRESP; "INTEGER" NVAR, NRESP;
"INTEGER" "ARRAY" VARNR, RESPNR;
"COMMENT" DUMMY PROCEDURE; ;

JOERSEKOG(.05, READRESP, LABELS)
"END"

Input:

CHANNEL.42=40
CHANNEL.43=60
CHANNEL.44=61
CHANNEL.45=A
CHANNEL.46=A
CHANNEL.47=A
CHANNEL.48=A
CHANNEL.49=A
CHANNEL.50=A
CHANNEL.51=A
CHANNEL.52=A
CHANNEL.53=A
CHANNEL.54=A
CHANNEL.55=A
CHANNEL.56=A
CHANNEL.57=A
CHANNEL.58=A
CHANNEL.59=A
CHANNEL.END
100 11 3 0 0 1 2
1.0000
.7948 5.0000
.7432 1.0000
.7387 5.0000
.7512 5.0000
.7674 5.0000
.8536 5.0000
.8516 5.0000
.9510 5.0000
.9445 5.0000
.9314 5.0000
.9163 5.0000
.9067 5.0000

Output:

CHANNEL.42=40
CHANNEL.43=60
CHANNEL.44=61
CHANNEL.45=A
CHANNEL.46=A
CHANNEL.47=A
CHANNEL.48=A
CHANNEL.49=A
CHANNEL.50=A
CHANNEL.51=A
CHANNEL.52=A
CHANNEL.53=A
CHANNEL.54=A
CHANNEL.55=A
CHANNEL.56=A
CHANNEL.57=A
CHANNEL.58=A
CHANNEL.59=A
CHANNEL.END

271
Joreskog

JORESKOG FACTOR ANALYSIS

NUMBER OF RESPONDENTS: 100
NUMBER OF VARIABLES: 11
CODE FOR ENTERED MATRIX: 1
NUMBER OF LATENT ROOTS AND VECTORS TO BE COMPUTED: 3
NUMBER OF UNROTATED FACTORS: 2
NUMBER OF ANALYSES TO BE PERFORMED: 1
NUMBERS OF FACTORS:

CORRELATION MATRIX

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>.794</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>.745</td>
<td>.794</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>.731</td>
<td>.745</td>
<td>.649</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>.751</td>
<td>.731</td>
<td>.683</td>
<td>.595</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>.676</td>
<td>.649</td>
<td>.492</td>
<td>.404</td>
<td>.456</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>.656</td>
<td>.618</td>
<td>.568</td>
<td>.525</td>
<td>.521</td>
<td>.729</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>.645</td>
<td>.605</td>
<td>.549</td>
<td>.522</td>
<td>.529</td>
<td>.756</td>
<td>.716</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>.534</td>
<td>.518</td>
<td>.476</td>
<td>.469</td>
<td>.456</td>
<td>.511</td>
<td>.754</td>
<td>.716</td>
<td>1.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>.415</td>
<td>.371</td>
<td>.366</td>
<td>.363</td>
<td>.359</td>
<td>.349</td>
<td>.384</td>
<td>.383</td>
<td>.387</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>.367</td>
<td>.181</td>
<td>.190</td>
<td>.236</td>
<td>.265</td>
<td>.163</td>
<td>.508</td>
<td>.525</td>
<td>.568</td>
<td>.556</td>
<td>.722</td>
</tr>
</tbody>
</table>

DIAG INV MULT R MULT R+R

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+.1477**+0</td>
<td>+.966**+0</td>
<td>+.0327**+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>+.4776**+0</td>
<td>+.889**+0</td>
<td>+.795**+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>+.3597**+0</td>
<td>+.838**+0</td>
<td>+.763**+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>+.2799**+0</td>
<td>+.764**+0</td>
<td>+.634**+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>+.2688**+0</td>
<td>+.682**+0</td>
<td>+.652**+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>+.4114**+0</td>
<td>+.870**+0</td>
<td>+.756**+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>+.3949**+0</td>
<td>+.759**+0</td>
<td>+.610**+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>+.3665**+0</td>
<td>+.809**+0</td>
<td>+.724**+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>+.2522**+0</td>
<td>+.738**+0</td>
<td>+.563**+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>+.2668**+0</td>
<td>+.773**+0</td>
<td>+.597**+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>+.2608**+0</td>
<td>+.7852**+0</td>
<td>+.616**+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MATRIX S+

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+.1477**+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>+.6677**+0</td>
<td>+.4776**+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>+.3265**+0</td>
<td>+.319**+0</td>
<td>+.337**+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>+.4654**+0</td>
<td>+.2894**+0</td>
<td>+.1972**+0</td>
<td>+.2799**+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>+.4607**+0</td>
<td>+.2469**+0</td>
<td>+.1834**+0</td>
<td>+.1489**+0</td>
<td>+.2884**+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>+.9986**+0</td>
<td>+.1991**+0</td>
<td>+.152**+0</td>
<td>+.1353**+0</td>
<td>+.1761**+0</td>
<td>+.4114**+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>+.7599**+0</td>
<td>+.2862**+0</td>
<td>+.237**+0</td>
<td>+.1984**+0</td>
<td>+.2634**+0</td>
<td>+.3411**+0</td>
<td>+.528**+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>+.3252**+0</td>
<td>+.8134**+0</td>
<td>+.597**+0</td>
<td>+.4618**+0</td>
<td>+.6467**+0</td>
<td>+.2648**+0</td>
<td>+.1731**+0</td>
<td>+.3624**+0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>+.2668**+0</td>
<td>+.6999**+0</td>
<td>+.4784**+0</td>
<td>+.4159**+0</td>
<td>+.6909**+0</td>
<td>+.1244**+0</td>
<td>+.9837**+0</td>
<td>+.2621**+0</td>
<td>+.2202**+0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>+.2523**+0</td>
<td>+.6925**+0</td>
<td>+.4797**+0</td>
<td>+.3645**+0</td>
<td>+.6416**+0</td>
<td>+.1756**+0</td>
<td>+.1393**+0</td>
<td>+.1972**+0</td>
<td>+.1107**+0</td>
<td>+.2486**+0</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>+.2401**+0</td>
<td>+.538**+0</td>
<td>+.5644**+0</td>
<td>+.4353**+0</td>
<td>+.4487**+0</td>
<td>+.1665**+0</td>
<td>+.1208**+0</td>
<td>+.2061**+0</td>
<td>+.1339**+0</td>
<td>+.1841**+0</td>
<td>+.2608**+0</td>
</tr>
</tbody>
</table>

TRACE: 4.886**+0
DETERMINANT: 1.568**+0
### Latent Roots and Vectors of S

<table>
<thead>
<tr>
<th>Root</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.3174*01</td>
</tr>
<tr>
<td>2</td>
<td>0.7763*01</td>
</tr>
<tr>
<td>3</td>
<td>0.8106*01</td>
</tr>
<tr>
<td>4</td>
<td>1.068*01</td>
</tr>
<tr>
<td>5</td>
<td>1.136*01</td>
</tr>
<tr>
<td>6</td>
<td>1.308*01</td>
</tr>
<tr>
<td>7</td>
<td>1.469*01</td>
</tr>
<tr>
<td>8</td>
<td>1.504*01</td>
</tr>
<tr>
<td>9</td>
<td>1.529*01</td>
</tr>
<tr>
<td>10</td>
<td>1.591*01</td>
</tr>
</tbody>
</table>

### Factor Solution

**Based on Correlation Matrix:**

#### Canonical Factor Loadings

<table>
<thead>
<tr>
<th>Factor</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5427*01</td>
</tr>
<tr>
<td>2</td>
<td>0.5697*01</td>
</tr>
<tr>
<td>3</td>
<td>0.6427*01</td>
</tr>
<tr>
<td>4</td>
<td>0.7066*01</td>
</tr>
<tr>
<td>5</td>
<td>0.7757*01</td>
</tr>
<tr>
<td>6</td>
<td>0.8030*01</td>
</tr>
<tr>
<td>7</td>
<td>0.8269*01</td>
</tr>
<tr>
<td>8</td>
<td>0.8965*01</td>
</tr>
<tr>
<td>9</td>
<td>0.9100*01</td>
</tr>
<tr>
<td>10</td>
<td>0.9390*01</td>
</tr>
</tbody>
</table>

#### Guttman-Criterion

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>1.9971*01</td>
</tr>
</tbody>
</table>

#### Transformation Matrix

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.8815*01</td>
</tr>
<tr>
<td>2</td>
<td>0.8340*01</td>
</tr>
</tbody>
</table>

**Number of Iterations:** 2
### Varimax Rotated Factor Loadings

<table>
<thead>
<tr>
<th>Variable</th>
<th>Factor 1</th>
<th>Factor 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.428571</td>
<td>.511364</td>
</tr>
<tr>
<td>2</td>
<td>.889805</td>
<td>-.388600</td>
</tr>
<tr>
<td>3</td>
<td>-.666667</td>
<td>.535601</td>
</tr>
<tr>
<td>4</td>
<td>.803846</td>
<td>.433333</td>
</tr>
<tr>
<td>5</td>
<td>.760930</td>
<td>-.105700</td>
</tr>
<tr>
<td>6</td>
<td>-.547619</td>
<td>-.654600</td>
</tr>
<tr>
<td>7</td>
<td>.748900</td>
<td>-.407000</td>
</tr>
<tr>
<td>8</td>
<td>-.134600</td>
<td>-.846370</td>
</tr>
<tr>
<td>9</td>
<td>.106600</td>
<td>-.682370</td>
</tr>
<tr>
<td>10</td>
<td>-.147500</td>
<td>-.769800</td>
</tr>
<tr>
<td>11</td>
<td>-.110000</td>
<td>-.759100</td>
</tr>
</tbody>
</table>

**Number of Unrotated Factors:** 0

### Guttman-Criterion After Rotation

<table>
<thead>
<tr>
<th>Variable</th>
<th>Factor 1</th>
<th>Factor 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.876700</td>
<td>.744100</td>
</tr>
</tbody>
</table>

### Percentage of Variance Explained: 67.26%

### Communalities

<table>
<thead>
<tr>
<th>Variable</th>
<th>Communalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.942800</td>
</tr>
<tr>
<td>2</td>
<td>.713300</td>
</tr>
<tr>
<td>3</td>
<td>.448800</td>
</tr>
<tr>
<td>4</td>
<td>.663000</td>
</tr>
<tr>
<td>5</td>
<td>.415200</td>
</tr>
<tr>
<td>6</td>
<td>.727400</td>
</tr>
<tr>
<td>7</td>
<td>.706400</td>
</tr>
<tr>
<td>8</td>
<td>.774300</td>
</tr>
<tr>
<td>9</td>
<td>.473700</td>
</tr>
<tr>
<td>10</td>
<td>.585900</td>
</tr>
<tr>
<td>11</td>
<td>.588300</td>
</tr>
</tbody>
</table>

### Residual Correlation Matrix

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-.1787*00</td>
<td>-.2047*00</td>
<td>-.1200*00</td>
<td>-.1000*00</td>
<td>-.2030*00</td>
<td>-.9612*00</td>
<td>-.9724*00</td>
<td>-.8100*00</td>
<td>-.4522*00</td>
<td>-.6926*00</td>
<td>-.5294*00</td>
</tr>
<tr>
<td>2</td>
<td>.2642*02</td>
<td>.2449*01</td>
<td>.2300*01</td>
<td>.2260*01</td>
<td>-.7912*02</td>
<td>-.3362*02</td>
<td>-.2303*01</td>
<td>-.3032*02</td>
<td>-.5698*02</td>
<td>-.7983*01</td>
<td>-.6100*00</td>
</tr>
<tr>
<td>3</td>
<td>.1100*01</td>
<td>-.9700*01</td>
<td>-.2301*01</td>
<td>-.2600*01</td>
<td>-.7912*02</td>
<td>-.3362*02</td>
<td>-.2303*01</td>
<td>-.3032*02</td>
<td>-.5698*02</td>
<td>-.7983*01</td>
<td>-.6100*00</td>
</tr>
<tr>
<td>4</td>
<td>.4600*04</td>
<td>-.7200*01</td>
<td>-.2600*01</td>
<td>-.7912*02</td>
<td>-.3362*02</td>
<td>-.2303*01</td>
<td>-.3032*02</td>
<td>-.5698*02</td>
<td>-.7983*01</td>
<td>-.6100*00</td>
<td>-.4000*00</td>
</tr>
<tr>
<td>5</td>
<td>.2620*02</td>
<td>-.6500*01</td>
<td>-.6522*01</td>
<td>-.3360*01</td>
<td>-.2303*01</td>
<td>-.3032*02</td>
<td>-.5698*02</td>
<td>-.7983*01</td>
<td>-.6100*00</td>
<td>-.4000*00</td>
<td>-.2990*00</td>
</tr>
<tr>
<td>6</td>
<td>.3000*02</td>
<td>-.6100*01</td>
<td>-.2301*01</td>
<td>-.2600*01</td>
<td>-.7912*02</td>
<td>-.3362*02</td>
<td>-.2303*01</td>
<td>-.3032*02</td>
<td>-.5698*02</td>
<td>-.7983*01</td>
<td>-.6100*00</td>
</tr>
<tr>
<td>7</td>
<td>.3100*01</td>
<td>-.5054*00</td>
<td>-.9439*01</td>
<td>-.9724*00</td>
<td>-.8100*01</td>
<td>-.5898*01</td>
<td>-.6371*01</td>
<td>-.5898*01</td>
<td>-.6371*01</td>
<td>-.5898*01</td>
<td>-.5898*01</td>
</tr>
<tr>
<td>8</td>
<td>.4549*01</td>
<td>-.3200*01</td>
<td>-.1533*01</td>
<td>-.1066*01</td>
<td>-.7957*02</td>
<td>-.4167*01</td>
<td>-.4964*01</td>
<td>-.4964*01</td>
<td>-.4964*01</td>
<td>-.4964*01</td>
<td>-.4964*01</td>
</tr>
<tr>
<td>9</td>
<td>.3284*02</td>
<td>-.4200*01</td>
<td>-.6577*01</td>
<td>-.1964*01</td>
<td>-.1578*01</td>
<td>-.8631*01</td>
<td>-.6462*01</td>
<td>-.6462*01</td>
<td>-.6462*01</td>
<td>-.6462*01</td>
<td>-.6462*01</td>
</tr>
<tr>
<td>10</td>
<td>.4293*02</td>
<td>-.3176*01</td>
<td>-.2080*00</td>
<td>-.2556*01</td>
<td>-.3584*02</td>
<td>-.2233*02</td>
<td>-.3128*02</td>
<td>-.2776*02</td>
<td>-.3519*01</td>
<td>-.2999*00</td>
<td>-.1351*00</td>
</tr>
<tr>
<td>11</td>
<td>.6159*02</td>
<td>-.4300*01</td>
<td>-.6887*01</td>
<td>-.7130*01</td>
<td>-.5598*01</td>
<td>-.4642*01</td>
<td>-.6598*01</td>
<td>-.6598*01</td>
<td>-.6598*01</td>
<td>-.6598*01</td>
<td>-.6598*01</td>
</tr>
</tbody>
</table>

### Test of Fit

<table>
<thead>
<tr>
<th>Mean of the 9 lowest latent roots of S-STAR</th>
<th>U-Statistic</th>
<th>Degrees of Freedom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.109*01</td>
<td>1.426*02</td>
<td>44</td>
</tr>
</tbody>
</table>

**End of Joreskog**
"CODE" 44400;
"PROCEDURE" JORESKOG (EPS, READ RESPONSE, OWN MARGLABS);
"VALUE" EPS; "REAL" EPS;
"PROCEDURE" READ RESPONSE, OWN MARGLABS;
"BEGIN"
   "INTEGER" HK, K, P1, P, T, NRESP, MATRIX, NPROB,
           NVAL, J, COVCORR, IN, OC, SCORES, C0WM, CORRM,
           STANDARD, INV, REDCOV, REDCORR, FSCORES,
           PSEUDO, CIN, RESP, CHN;
   "REAL" SKIP;
   "BOOLEAN" PR SCORES, STANDARDIZE, PR STANDARD,
           PR FSCORES, RESPLABS;

   "INTEGER" "PROCEDURE" MEMORY;
   "BEGIN" "INTEGER" A; A := AVAILABLE - 1000;
       MEMORY := "IF" A < 0 "THEN" 0 "ELSE" A
       "END" MEMORY;

   "INTEGER" "PROCEDURE" VARIMAXROTATION (F, NTEST,
           NFACTOR1, NFACTOR, ROTATIONMATRIX, T, VV, EPS);
   "VALUE" NTEST, NFACTOR;
   "INTEGER" NTEST, NFACTOR, NFACTOR1;
   "BOOLEAN" ROTATIONMATRIX; "REAL" VV, EPS; "ARRAY" F, T;
   "BEGIN"
      "INTEGER" I, J, K, COUNTER, NFACTORMIN1;
      "REAL" A, B, C, D, B2, D2, U, V, NUMERATOR,
           DENOMINATOR, HALFCOS2FI, COSFI, SINF;
      "BOOLEAN" OKAY; "ARRAY" H [1 : NTEST];

      "FOR" I := 1 "STEP" 1 "UNTIL" NTEST "DO"
      "BEGIN"
         A := H [I] := SQRT (MATTAM (NFACTOR1, NFACTOR,
                           I, I, F, F));
      "FOR" J := NFACTOR1 "STEP" 1 "UNTIL" NFACTOR
          "DO" F [I, J] := F [I, J] / A
      "END";
      VV := VECVEC (1, NTEST, 0, H, H);
      "IF" ROTATIONMATRIX "THEN"
      "FOR" J := NFACTOR1 "STEP" 1 "UNTIL" NFACTOR "DO"
          "BEGIN" T [J, J] := 1;
              "FOR" I := J - 1 "STEP" -1 "UNTIL" 1
                  "DO" T [I, J] := T [I, J] := 0
          "END";
          COUNTER := 0; NFACTORMIN1 := NFACTOR - 1;
      "END"
      OKAY := "TRUE"; COUNTER := COUNTER + 1;
      "FOR" J := NFACTOR1 "STEP" 1 "UNTIL" NFACTORMIN1 "DO"
      "FOR" K := J + 1 "STEP" 1 "UNTIL" NFACTOR "DO"
          "BEGIN" A := B := C := D := 0;
              "FOR" I := 1 "STEP" 1 "UNTIL" NTEST "DO"
                  "BEGIN" B2 := F [I, J]; D2 := F [I, K];
                      U := (B2 + D2) * (B2 - D2); B2 := B2 + D2;
                      V := B2 + B2; A := A + U; B := B + V;
  "END";
C:= C + (U + V) * (U - V); D2:= U * V;
D:= D + D2 + D2
"END";
U:= A * B; NUMERATOR:= D - (U + U) / NTEST;
DENOMINATOR:= C - (A + B) * (A - B) / NTEST;
"IF" ^ (ABS (NUMERATOR) < "-6" OR
(DENOMINATOR = 0 AND
ABS (NUMERATOR / DENOMINATOR) < EPS))
"THEN"
"BEGIN" OKAY:= "FALSE";
HALFCOS2FI:= SQRT (.125 +
DENOMINATOR / (8 * SQRT (NUMERATOR ** 2
+ DENOMINATOR ** 2)));
COSFI:= SQRT (.5 + HALFCOS2FI);
SINFI:= SQRT (.5 - HALFCOS2FI);
"IF" NUMERATOR < 0 "THEN" SINF:= - SINF;
ROTCOL (I, NTEST, J, K, F, COSFI, SINF);
"IF" ROTATIONMATRIX
"THEN" ROTCOL (NFACTOR1, NFACTOR, J, K,
T, COSFI, SINF)
"END"
"END";
"IF" ^ OKAY "AND" COUNTER < 30
"THEN" "GOTO" ROTATION;
"FOR" I:= 1 "STEP" 1 "UNTIL" NTEST "DO"
"BEGIN" A:= H [I];
"FOR" J:= NFACTOR1 "STEP" 1 "UNTIL" NFACTOR
"DO" F [I, J]:= F [I, J] * A
"END";
VARIAXROTATION:= COUNTER
"END" VARIAX ROTATION;
"PROCEDURE" STANDARD SCORES (COVM, SCORES, STANDARD,
C, K, NRESP);
"PROCEDURE" STANDARD SCORES (COVM, SCORES, STANDARD, K, NRESP);
"INTEGER" COVM, SCORES, STANDARD, K, NRESP; "ARRAY" C;
"BEGIN" "INTEGER" I, J; "REAL" SQRTSJS;
"ARRAY" S, G, BUF [I : K];
"FOR" I:= 1 "STEP" 1 "UNTIL" K "DO" G [I]:= 0;
REWIND (COVM); REWIND (SCORES);
"FOR" I:= 1 "STEP" 1 "UNTIL" NRESP "DO" "BEGIN" GET ARRAY (SCORES, BUF);
"FOR" J:= 1 "STEP" 1 "UNTIL" K "DO" "BEGIN" G [J]:= (I - 1) / I; G [J] + BUF[J] / I
"END";
GET ARRAY (COVM, C); I:= 0;
"FOR" J:= 1 "STEP" 1 "UNTIL" K "DO" "BEGIN" I:= I + J; S [J]:= C [I] "END";
REWIND (SCORES);;
"FOR" I:= 1 "STEP" 1 "UNTIL" NRESP "DO" "BEGIN" GET ARRAY (SCORES, BUF);
"FOR" J:= 1 "STEP" 1 "UNTIL" K "DO" "BEGIN" SQRTSJS:= SQRT (S [JJ]);
PUT ARRAY (STANDARD, BUF)
END" STANDARD SCORES;

"PROCEDURE" FACTOR SCORES (STANDARD, FSM, K, P, NRESP, FSCORES);
"VALUE" STANDARD, K, P, NRESP, FSCORES; "ARRAY" FSM;
"INTEGER" STANDARD, K, P, NRESP, FSCORES;
"BEGIN" "INTEGER" I, J, L;
"ARRAY" BUF1 [1 : K], BUF2 [1 : P];
REWIND (STANDARD); REWIND(FSCORES);
"FOR" I := 1 "STEP" 1 "UNTIL" NRESP "DO"
"BEGIN" GET ARRAY (STANDARD, BUF1);
"FOR" L := 1 "STEP" 1 "UNTIL" P
"DO" BUF2 [L] := TAMVEC (I, K, L, FSM, BUF1);
PUT ARRAY (FSCORES, BUF2)
"END"
"END" FACTOR SCORES;

"PROCEDURE" BIGPRINT (HEADING, FILE, K, NRESP, VARNR, RESPNR);
"VALUE" FILE, K, NRESP; "INTEGER" FILE, K, NRESP;
"STRING" HEADING; "INTEGER" "ARRAY" RESPNR;
"ARRAY" VARNR;

"IF" K * (NRESP + 1) < MEMORY "THEN"
"BEGIN" "INTEGER" I, J;
"ARRAY" X [1 : NRESP, 1 : K], BUF [1 : K];
REWIND (FILE);
"FOR" I := 1 "STEP" 1 "UNTIL" NRESP "DO"
"BEGIN" GET ARRAY (FILE, BUF);
DUP ROW VEC (1, K, I, X, BUF)
"END"
RECTANGLE (OC, HEADING, X, 1, NRESP, 1, K, VARNR, 2, RESPNR, 1)
"END"

"ELSE"
"BEGIN" "INTEGER" I, J, AA, HL, HU, VL, VU, PAGE;
"ARRAY" BUF [1 : K];
REWIND (FILE); AA := MEMORY // (K * 50); PAGE := 1;
"IF" AA = 0 "THEN"
"BEGIN" AA := MEMORY // 50;
AA := HU := (AA // 10) * 10;
VU := 50; "IF" AA = 0 "THEN"
"BEGIN"
OUTPUT (OC, "(" // "INSUFFICIENT ")", "CM FOR PRINTING ")", N,;">"), HEADING);
"GOTO" WRONG
"END";

277
"FOR" VL:= 1, VL + 50 "WHILE" VL <= NRESP "DO"
"BEGIN"
"FOR" HL:= 1, HL + AA "WHILE" HL <= K "DO"
"BEGIN" "ARRAY" X [VL : VU, HL : HU];
"FOR" I:= VL "STEP" 1 "UNTIL" VU "DO"
"BEGIN" GET ARRAY (FILE, BUF);
DUP ROWVEC (HL, HU, I, X, BUF)
"END";
PAGE:= RECTANGLE (OC, HEADING, X, VL,
VU, HL, HU, VARNR, Z, RESPNR, PAGE);
"IF" HU= K "THEN"
"FOR" I:= 1 "STEP" 1 "UNTIL" 50
"DO" BACKSPACE (FILE);
HU:= HU + AA; "IF" HJ > K "THEN" HU:= K
"END";
VU:= VU + 50;
"IF" VU > NRESP "THEN" VU:= NRESP;
HU:= AA
"END"
"END" "ELSE"
"BEGIN" VU:= AA:= AA * 50;
"IF" VU > NRESP "THEN" VU:= NRESP;
"FOR" VL:= 1, VL + AA "WHILE" VL <= NRESP "DO"
"BEGIN" "ARRAY" X [VL : VU, I : K];
"FOR" I:= VL "STEP" 1 "UNTIL" VU "DO"
"BEGIN" GET ARRAY (FILE, BUF);
"FOR" J:= 1 "STEP" 1 "UNTIL" K
"DO" X [I, J]:= BUF [J]
"END";
PAGE:= RECTANGLE (OC, HEADING, X, VL, VU,
1, K, VARNR, Z, RESPNR, PAGE);
VU:= VU + AA;
"IF" VU > NRESP "THEN" VU:= NRESP
"END"
"END"; WRONG:
"END" BIGPRINT;

"REAL" "PROCEDURE" SUM (I, A, B, X); "VALUE" B;
"INTEGER" I, A, B; "REAL" X;
"BEGIN" "REAL" S;
S:= 0;
"FOR" I:= A "STEP" 1 "UNTIL" B "DO" S:= S + X;
SUM:= S
"END" SUM;

"INTEGER" "PROCEDURE" LINENUMBER;
"BEGIN" "INTEGER" L;
SYSPARAM (OC, 3, L); LINENUMBER:= L + 1
"END" LINENUMBER;

"PROCEDURE" INVSYM1 (A, N);
"VALUE" N; "INTEGER" N; "ARRAY" A;
"BEGIN" "INTEGER" I, II, II, J, IJ, JJ; "REAL" R;
"ARRAY" U [1 : N];

II:= (N + 1) * N // 2;
"FOR" I:= N "STEP" - 1 "UNTIL" 1 "DO"
"BEGIN" R:= 1 / A [II]; II:= I + 1; IJ:= II + I;
"FOR" J:= II "STEP" 1 "UNTIL" N "DO"
"BEGIN" U [IJ]:= A [IJ]; IJ:= IJ + J "END";
"FOR" J:= N "STEP" - 1 "UNTIL" I "DO"
"BEGIN" IJ:= IJ - I; IJ:= IJ - J;
SEVEC (J + 1, N, J + J, 0, A, U)) * R
"END";
A [IIJ]:= (R - SEVEC (II, N, II + I, 0, A, U)) * R;
II:= II - I
"END"
"END" INVSYM;

"INTEGER" "PROCEDURE" RNKSYM10 (A, N, P, AUX);
"VALUE" N; "INTEGER" N; "INTEGER" "ARRAY" P;
"ARRAY" A, AUX;
"BEGIN"
"INTEGER" K, PK, KK, KJ, PP, I, J, JJ, T, LOW, UP;
"REAL" NORM, EPSNORM, M, MAX, D, R, W;

D:= 1; NORM:= 0; KK:= 0;
"FOR" K:= 1 "STEP" 1 "UNTIL" N "DO"
"BEGIN" KK:= KK + K;
"END";
EPSNORM:= AUX [0] * NORM; AUX [I]:= NORM;
M:= 0; KK:= 0;
"FOR" K:= 1 "STEP" 1 "UNTIL" N "DO"
"BEGIN" MAX:= EPSNORM; T:= KK;
"FOR" J:= K "STEP" 1 "UNTIL" N "DO"
"BEGIN" T:= T + J;
"IF" A [T] > MAX "THEN"
"BEGIN" MAX:= A [T]; PK:= J; PP:= T "END"
"END";
"IF" MAX <= EPSNORM "THEN"
"BEGIN" "FOR" I:= K "STEP" 1 "UNTIL" N "DO"
"BEGIN" KK:= KK + I; LOW:= KK - I + 1;
UP:= LOW + K - 2; R:= ABS (A [KK]);
"IF" R > M "THEN" M:= R; KJ:= KK + I;
"FOR" J:= I + 1 "STEP" 1 "UNTIL" N "DO"
"BEGIN" R:= A [KJ]; VECVEC (LOW, UP, KJ - KK, A, A);
R:= ABS (R);
"IF" R > M "THEN" M:= R;
KJ:= KK + J
"END"
"END";
"GOTO" END
"END";

KK := KK + K;
LOW := KK - K + 1;
UP := KK - 1;
P [KK] := PK; D := D * MAX;
"IF" PK <= K "THEN"
"BEGIN"
ICHVEC (LOW, UP, PP - PK - KK, K, A);
ICHSEQVEC (K + 1, PK - 1, KK + K, PP - PK, K, A);
ITCHVEC (PK + 1, N, PP + K, PK - K, A);
"END";

A [KK] := R := SQRT (MAX);
KJ := KK + K;
JJ := KK;
"FOR" J := K + 1 "STEP" 1 "UNTIL" N "DO"
"BEGIN"
VECVEC (LOW, UP, KJ - K, K, A)) / R;
JJ := JJ + J;
KJ := KJ + J
"END"

"END";
K := N + 1;
RNSYM10 := "IF" M <= 2 * EPSNORM
"THEN" K - 1 "ELSE" - K
"END" RNSYM10;

"PROCEDURE" INVSYM10 (A, N, P);
"VALUE" N; "INTEGER" N;
"INTEGER" "ARRAY" P; "ARRAY" A;
"BEGIN" "INTEGER" I, II, PI, PP; "REAL" R;

INVSYM1 (A, N); II := (N + 1) * N // 2;
"FOR" I := N "STEP" - 1 "UNTIL" 1 "DO"
"BEGIN" PI := P [I];
"IF" PI <= 1 "THEN"
"BEGIN" PP := (PI + 1) * PI // 2;
ICHVEC (II - I + 1, II - 1, PP - PI - II + I, A);
ICHSEQVEC (II + 1, PI, II + 1, II + I, PP - PI, A);
ICHSEQ (PI + 1, N, PP + I, PI - I, A);
"END";
II := II - I
"END" INVSYM10;

IN := 63; OC := 64; SCORES := 1; COVM := 2; CORRM := 3;
INV := 4; STANDARD := 5; REDCOV := 6; REDCORR := 7;
FSCORES := 8; CIN := 62;
"IF" EPS <= 0 "THEN" EPS := -EPS
"ELSE" CHANNEL CARDS(60);
"IF" "NOT" CHEXIST(CIN) "THEN"
CHANNEL(CIN, "("C")", "("LM")", "("CINFO")", "("P")", 80);
"IF" "NOT" CHEXIST(CIN) "THEN"
CHANNEL(IN, "("C")", "("LFN")", "("INFILE")", "("PP")", 80);
"IF" NOT CEXIST(CC) "THEN"
CHANNEL(CC, "("C")", "("LFN")", "("OUTFILE")", "("PP")", 136, "("PP")", 60);
"FOR" CHN:= SCORES, COVM, CORRM, INV, STANDARD,
REDCOV, REDCORR, FSORES "DO"
"BEGIN" "IF" NOT CEXIST(CH) "THEN"
CHANNEL(CH, "("B")");
OPEN(CH, "("I")");
"END"

INPUT(IN, "("N")", NRESP, K, MATRIX);
"IF" NRESP < 0 "THEN"
"BEGIN" NRESP:= -NRESP; PR SCORES:= "TRUE" "END"
"ELSE" PR SCORES:= "FALSE";
"IF" K < 0 "THEN"
"BEGIN" K:= -K; PR STANDARD:= "TRUE" "END"
"ELSE" PR STANDARD:= "FALSE"; HK:= K;
REWIND (SCORES); REWIND (COVM); REWIND (CORRM);
REWIND (INV); REWIND (REDCOV); REWIND (REDCORR);
"IF" MATRIX = 3 "THEN" MATRIX:= -3;
"IF" ABS (MATRIX) = 1 "THEN" INREAL (IN, SKIP)
"ELSE" SKIP:= "-
INPUT(IN, "("N")", P, P1, NVAL); P1:= P1 + 1;
"IF" NVAL < 0 "THEN"
"BEGIN" NVAL:= -NVAL; STANDARDIZE:= "TRUE" "END"
"ELSE" STANDARDIZE:= PR STANDARD;
T:= (K + 1) * K / 2;
COVCORR:= "IF" ABS (MATRIX) < 3
"THEN" COVM "ELSE" CORRM;
"BEGIN" "INTEGER" I, COUNTER2, COUNT, RANK, HI;
"REAL" RE, REL, TRACE, DI, DIAG, DET, VV;
"INTEGER" "ARRAY" ICC, VARRN, H [I : K],
VALUE [I : NVAL];
"ARRAY" C [I : K], EM [0 : 91], AUX [0 : 3], D, CC,
DIAG [I : K], KOM [I : K, 1 : 2];
"BOOLEAN" "ARRAY" BCC [I : K];

"FOR" I:= 1 "STEP" 1 "UNTIL" K "DO"
"BEGIN" BCC[I]:="TRUE"; ICC[I]:=I "END";
RESPLABS:= PR SCORES "OR" PR STANDARD;
"IF" STANDARDIZE "AND" RESPLABS
"THEN" INITARRAY(IN, VALUE)
"ELSE"
"BEGIN" "FOR" I:= 1 "STEP" 1 "UNTIL" NVAL "DO"
"BEGIN" ININTEGER(IN, J);
"IF" J < 0 "THEN" STANDARDIZE:= RESPLABS:= "TRUE";
VALUE [I]:= J
"END"
"END"
"IF" STANDARDIZE "THEN" REWIND (STANDARD);
RESPLABS:= ABS (MATRIX) = 1 "AND" RESPLABS;
"BEGIN" "REAL" X; EOF (IN, L);
    INPUT (IN, "(""/"")"); L: "END"
EOF (IN, DUMP);
OUTPUT (OC, "(""*"", "(""JORESKOG FACTOR ANALYSIS"")", \\
24(""(""="")", \\
(""NUMBER OF RESPONDENTS:"")",
62D,
(""NUMBER OF VARIABLES:"")",
62D,
(""CODE FOR ENTERED MATRIX:"")",
-52D,
(""NUMBER OF LATENT ROOTS AND VECTORS TO BE COMPUTED:"")",
62D,
(""NUMBER OF UNROTATED FACTORS:"")",
62D,
(""NUMBER OF ANALYSES TO BE PERFORMED:"")",
62D,
(""NUMBERS OF FACTORS:"")",
2D")",
NRESP, K, "IF" MATRIX = -3 "THEN" 3 "ELSE" MATRIX,
P, P1 - 1, NVAL);
"FOR" I:= 1 "STEP" 1 "UNTIL" NVAL
"DO" OUTPUT (OC, "(""B3DDB"")", ABS (VALUE [I]));
OUTPUT (OC, "(""="")");
"IF" MATRIX < 0 "AND" MATRIX = -3
"THEN" OUTPUT (OC, "(""/")",
(""THE ANALYSIS OF THE COVARIANCE MATRIX"")",
(""WILL BE SUPPRESSED"")", \\
"THEN" OUTPUT (OC, "(""/, \\
(""THE MISSING OBSERVATIONS OF A VARIABLE, CODED BY"")",
-0.3D+9D, "(""", HAVE BEEN REPLACED BY THE MEAN"")", \\
(""OF THE NON-MISSING OBSERVATIONS OF THAT VARIABLE"")",
48(""(""="")", \\
"SKIP";
PSEUDO: = "IF" RESPLABS "THEN" NRESP "ELSE" 1;
"BEGIN" "REAL" "ARRAY" L, LR, VECA,
[1 : K, 1 : ("IF" P > 2 "THEN" P "ELSE" 3)]]
HELPVEC [1 : K], ROTMAT [1 : P, 1 : P], VAL,
KOMZ [1 : P];
"INTEGER" "ARRAY" RRR[1 : PSEUDO], FACNR[1 : P];
"ARRAY" DUPFACNR[1 : P];
"IF" RESPLABS "THEN"
"BEGIN"
"FOR" I:= 1 "STEP" 1 "UNTIL" NRESP "DO"
RR [I] := I
"END";
"FOR" I:= 1 "STEP" 1 "UNTIL" P "DO"
"BEGIN" FACNR[I] := 1; DUPFACNR[I] := I "END";
OWN MARGLABS (ICC, K, RR, PSEUDO);
"FOR" I:= 1 "STEP" 1 "UNTIL" < "DO"

CC[I] := ICC[I];
INIMAT (1, P, 1, P, ROTMAT, 0);
INIMAT D (1, P, 0, ROTMAT, 1);
"IF" ABS (MATRIX) = 1 "THEN"
"BEGIN" "INTEGER" A, PJ, P;
"ARRAY" OWRESP [0 : K], OWVAR [1 : K];

INIVEC (1, K, OWRESP, 0);
INIVEC (1, K, OWVAR, 0);
"GOTO" "IF" (NRESP + 2) * K < MEMORY
"THEN" SYSTEM1
"ELSE"
"IF" SKIP = − "7"
"THEN" SYSTEM2 "ELSE" SYSTEM3;

SYSTEM1:
"BEGIN" "REAL" SJ; "INTEGER" R;
"ARRAY" X [1 : NRESP, 1 : K], S,
      BUF [1 : K];

RESP:= 0;
"FOR" R:= 1 "STEP" 1 "UNTIL" NRESP "DO"
"BEGIN"
    READ RESPONSE (J, X [R, J],
            K, SKIP);
    RESP:= RESP + 1;
    "IF" SKIP = "7" "THEN"
    "BEGIN" A:= 0;
      "FOR" J:= 1 "STEP" 1
        "UNTIL" K "DO"
      "IF" X [R, J] = SKIP
      "THEN" A:= A + 1;
      OWRESP [A]:= OWRESP [A] + 1
    "END"
    "END";
  "FOR" J:= 1 "STEP" 1 "UNTIL" K "DO"
  "BEGIN" "IF" SKIP = "7"
    "THEN" SJ:= S [J]:= SUM (I, 1, NRESP, X [I, J])
  "ELSE"
  "BEGIN" "REAL" M;
    SJ:= 0; A:= 0;
    "FOR" I:= 1 "STEP" 1
      "UNTIL" NRESP "DO"
    "IF" X [I, J] = SKIP
    "THEN" A:= A + 1
    "ELSE" SJ:= SJ + X [I, J];
    OWVAR [J]:= A;
  M:= "IF" A = NRESP "THEN" 0
  "ELSE" SJ / (NRESP - A);
  SJ:= S [J]:= NRESP + M;
  "FOR" I:= 1 "STEP" 1
    "UNTIL" NRESP "DO"
  "IF" X [I, J] = SKIP
  "THEN" X [I, J]:= M
  "END"
;PJ:= (J - 1) * J / 2;
"FOR" I:= J "STEP" − 1 "UNTIL" 1
"DO" C [PJ + IJ]:= (TAMMAT (0, NRESP, I, J, X, X) * NRESP − S [IJ] * SJ) / (NRESP − 1) / NRESP
"END";
"FOR" I:= 1 "STEP" 1 "UNTIL" NRESP "DO"
"BEGIN" DUP VEC ROW (I, K, I, BUF, X);
PUT ARRAY (SCORES, BUF)
"END"
"END";
"GOTO" OUTP;

SYSTEM2:
"BEGIN" "REAL" XJ, SJ; "ARRAY" S, X [1 : K];
"INTEGER" R; RESP:= 0;
INIVEC (1, K, S, 0);
INIVEC (1, T, C, 0);
"FOR" R:= 1 "STEP" 1 "UNTIL" NRESP "DO"
"BEGIN"
READ RESPONSE (J, X [IJ], K, SKIP);
PUT ARRAY (SCORES, X);
RESP:= RESP + 1;
"FOR" J:= 1 "STEP" 1 "UNTIL" K "DO"
"BEGIN" PJ:= (J − 1) * J / 2;
XJ:= X [IJ];
S [IJ]:= S [IJ] + XJ;
"FOR" I:= J "STEP" − 1 "UNTIL" 1
"DO" C [PJ + IJ]:= C [PJ + IJ] + X [IJ] * XJ
"END"
"END";
"FOR" J:= 1 "STEP" 1 "UNTIL" K "DO"
"BEGIN" PJ:= (J − 1) * J / 2;
SJ:= S [IJ];
"FOR" I:= J "STEP" − 1 "UNTIL" 1
"DO" C [PJ + IJ]:= C [PJ + IJ] + X [IJ] * XJ
"END"
"END";
"GOTO" OUTP;

SYSTEM3:
"BEGIN" "ARRAY" BUF, S, N [1 : K];
"INTEGER" AA, II; "REAL" PJ, MI, MJ, SJ;
A:= AA:= MEMORY // K;
"IF" A = 0 "THEN"
"BEGIN"
OUTPUT (OC, "("/"),("INSUFFICIENT ")",
"("CM FOR READING DATA")",///")
EXIT
"END";
INIVEC (1, K, S, 0);
INIVEC (1, T, C, 0); RESP:= 0;
NEXT0:
"BEGIN" "ARRAY" X [1 : A, 1 : K];
NEXT1:
"FOR" I:= 1 "STEP" 1 "UNTIL" A "DO"
"BEGIN"
READ RESPONSE (J, BUF [J], K, SKIP);
RESP:= RESP + 1;
PUT ARRAY (INV, BUF);
DUP ROW VEC (1, K, I, X, BUF);
"END";
"FOR" I:= 1 "STEP" 1 "UNTIL" A "DO"
"BEGIN" II:= 0;
"FOR" J:= 1 "STEP" 1 "UNTIL" K "DO"
"IF" X [I, J] = SKIP "THEN"
"BEGIN"
OWVAR [J]:= OWVAR [J] + 1;
II:= II + 1
"END"
"ELSE" S [JJ]:= S [JJ] + X [I, J];
OWRESP [II]:= OWRESP [II] + 1
"END";
"IF" RESP <= NRESP - A "THEN" "GOTO" NEXT1;
A:= NRESP - RESP;
"IF" A > 0 "THEN" "GOTO" NEXT0;
"FOR" J:= 1 "STEP" 1 "UNTIL" K "DO"
"BEGIN"
M [J]:= "IF" OWVAR [J] = NRESP "THEN" 0
"ELSE" S [JJ] / (KRESP - OWVAR [J]);
S [JJ]:= NRESP * M [J]
"END";
"END";
P:= 0; A:= AA; REWIND (INV);
NEXT2:
"BEGIN" "ARRAY" X [1 : A, 1 : K];
NEXT3:
"FOR" I:= 1 "STEP" 1 "UNTIL" A "DO"
"BEGIN" GET ARRAY (INV, BUF);
DUP ROW VEC (1, K, I, X, BUF)
"END";
"FOR" J:= 1 "STEP" 1 "UNTIL" K "DO"
"BEGIN"
PJ:= (J - 1) * J / 2;
MJ:= M [J];
"FOR" I:= J "STEP" - 1 "UNTIL" 1 "DO"
"BEGIN" MI:= M [II];
"FOR" II:= 1 "STEP" 1 "UNTIL" A
"DO"
"BEGIN"
"IF" X [II, I] = SKIP
"THEN" X [II, I] := M1;
"IF" X [II, J] = SKIP
"THEN" X [II, J] := MJ;
C [PJ + I] := C [PJ + I] + 
X [II, I] * X [II, J]
"END"
"END";
"FOR" I := 1 "STEP" 1 "UNTIL" A "DO"
"BEGIN"
  DUP VEC ROW (1, K, I, BUF, X);
  PUT ARRAY (SCORES, BUF)
"END";
P := P + A;
"IF" P <= NRESP - A
"THEN" "GOTO" NEXT3;
A := NRESP - P;
"IF" A > 0 "THEN" "GOTO" NEXT2;
"FOR" J := 1 "STEP" 1 "UNTIL" K "DO"
"BEGIN"
  PJ := (J - 1) * J / 2;
  SJ := S [J];
  "FOR" I := J "STEP" 1 "UNTIL" 1
  "DO" C [PJ + I] := (C [PJ + I] * 
     NRESP - S [IJ + SJ])
     / (NRESP - 1) / NRESP
"END"
"END";
REWRITE (INV)
"END";

OUTP:
"IF" SKIP = "7" "THEN"
"BEGIN" OUTPUT (OC, "("S,
("FREQUENCY OF MISSING OBSERVATIONS FOR EACH VARIABLE: ")",
"",64("("-"")),",
"("SECOND DIGIT I" "",100(48b),",
"("FIRST DIGIT I"",138,"("I""",",
64("("-""),",138,"("I""",",
58b,78b,"("I"",68"",
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0);
"FOR" I := 1 "STEP" 1 "UNTIL" K "DO"
"BEGIN" J := I // 10;
"IF" J + 10 = I
"THEN" OUTPUT (OC,
"("/,58d,78b,"("I"",B""),", J);
J := OWVAR [IJ];
"IF" J + 0
"THEN" OUTPUT (OC, "("-""
"ELSE" OUTPUT (OC, "("B,58d,78b"", J);
"END";
OUTPUT (OC, "/,64("("-""",6/,
"("NUMBER OF RESPONDENTS, A[I], WITH J ")",
"("MISSING VALUES:" ")",
51("="),/,,38,
"("J A[J])",/,,28,8("="),/,,4")")
"FOR" I: = 1 "STEP" 1 "UNTIL" K "DO"
"BEGIN" J:= OWRESP[I];
"IF" J = 0
"THEN" OUTPUT (OC,
"("DIZD,B,DIZD," ")", I, J);

"END"
"END"
PUT ARRAY (COVM, C);
"END"
"ELSE"
"BEGIN" IN ARRAY (CIN, C);
"IF" ABS (MATRIX) = 2
"THEN" PUT ARRAY (COVM, C)
"ELSE"
"IF" ABS (MATRIX) = 3
"THEN" PUT ARRAY (CORRM, C)
"END"
OUTPUT (OC, "("*")")
"IF" ABS (MATRIX) = 1 "THEN"
"BEGIN" "IF" PR SCORES "THEN"
"BIGPRINT ("("RAW DATA")", SCORES, K, NRESP,
CC, RR);
"IF" STANDARDIZE "THEN"
"BEGIN" STANDARD SCORES (COVM, SCORES,
STANDARD, C, K, NRESP);
"IF" PR STANDARD "THEN"
"BIGPRINT ("("STANDARDIZED DATA")", STANDARD, K, NRESP, CC, RR)
"END" "ELSE"
"BEGIN"REWIND (COVM);
GET ARRAY (COVM, C) "END"
"ELSE"
"BEGIN"REWIND (COVCCORR);
GET ARRAY (COVCCORR, C) "END"
AUX[0]: = "-1; RANK:= RNKSYM10 (C, K, H, AUX);
PUT ARRAY (INV, C);
"IF" RANK < 0 "THEN"
"BEGIN" OUTPUT (OC, "("/,,
"("COVARIANCE OR CORRELATION")",
"("MATRIX IS NOT POSITIVE SEMIDEFINITE")", 
"//"))
RANK:= - RANK - 1
"END"
"IF" RANK >= NRESP "THEN" RANK:= NRESP - 1;
"FOR" I: = 1 "STEP" 1 "UNTIL" K "DO"
VARNR[I]: = CC[I];
"IF" RANK < K "THEN"
"BEGIN" "INTEGER" BI, NI;
"INTEGER" "ARRAY" M, N [I : RANK], B [I : K];
"IF" ABS (MATRIX) < 3 "THEN"
"BEGIN" REWIND (COVM); GET ARRAY (COVM, C);
COVCORR := REDCOV;
TRIANGLE (OC,
"{"COVARIANCE MATRIX OF ALL VARIABLES"},
C, K, VARNR);
COUNTER2 := 0;
"FOR" I := 1 "STEP" 1 "UNTIL" K "DO"
"BEGIN" DIAGI := DIAG [I] :=
SORT (C [COUNTER2 + I]);
"FOR" J := 1 "STEP" 1 "UNTIL" I "DO"
"BEGIN" COUNTER2 := COUNTER2 + 1;
C [COUNTER2] :=
"IF" DIAGI < "-8" OR
DIAG [J] < "-8"
"THEN" .9999999
"ELSE" C [COUNTER2] / DIAGI /
DIAG [J];
"END";
"END";
TRIANGLE (OC,
"{"CORRELATION MATRIX OF ALL VARIABLES"},
C, -K, VARNR);
PUT ARRAY (CORRM, C); REWIND (COVM);
GET ARRAY (COVM, C);
"END"
"ELSE"
"BEGIN"
REWIND (CORRM); GET ARRAY (CORRM, C);
COVCORR := REDCOV;
TRIANGLE (OC,
"{"CORRELATION MATRIX OF ALL VARIABLES"},
C, -K, VARNR)
"END";
OUTPUT (OC, "("3/"("THE CORRELATION ")",
"("MATRIX IS SINGULAR")",/,
"("SOME OF THE VARIABLES ")",
"("WILL BE DELETED IN ORDER TO ")","
"("MAKE IT NON-SINGULAR")",3/******/));
"FOR" I := 1 "STEP" 1 "UNTIL" K "DO"
"BEGIN" VARNR [I] := -1; B [I] := I "END";
"FOR" I := 1 "STEP" 1 "UNTIL" RANK "DO"
"BEGIN"
J := H [I]; VARNR [B [J]] := 1;
M [I] := N [I] := 1;
"END";
T := COUNTER2 := COUNT := 0;
"FOR" I := 1 "STEP" 1 "UNTIL" K "DO"
"BEGIN" "IF" VARNR [I] - 1 "THEN"
"BEGIN"
"FOR" J := 1 "STEP" 1 "UNTIL" I "DO"
"IF" VARNR [J] - 1 "THEN"
"BEGIN" COUNT := COUNT + 1;
C [COUNT] := C [COUNTER2 + J]

"END"

VARNR [I] := T

"END"

"ELSE" T := T + 1;

COUNTER2 := COUNTER2 + I

"END"

PUT ARRAY (COVCORR, C);

"FOR" I := 1 "STEP" 1 "UNTIL" RANK "DO"

"BEGIN" BI := B [I]; NI := N [I];

J := H [I]; M [J] := M [BI -

VARNR [BI]];

N [J] := NI

"END"

T := O;

"FOR" I := 1 "STEP" 1 "UNTIL" K

"DO" BCC [I] := "FALSE";

"FOR" I := 1 "STEP" 1 "UNTIL" K "DO"

"IF" VARNR [I] := 1 "THEN"

"BEGIN" BCC [I] := "TRUE"; T := T + 1;

VARNR [T] := CC [I]

"END"

K := RANK; T := (K + 1) * K / 2

"END"

REWIND (INV); GET ARRAY (INV, C);

INVSYM10 (C, K, H);

"FOR" I := 1 "STEP" 1 "UNTIL" K

"DO" D [I] := SQRT (C [(I + 1) * I / 2]);

REWIND (INV); PUT ARRAY (INV, C);

REWIND (COVCORR); GET ARRAY (COVCORR, C);

"IF" ABS (MATRIX) < 3 "THEN"

"BEGIN" TRIANGLE (OC, "("COVARIANCE MATRIX")", C, K, VARNR);

COUNTER2 := 0;

"FOR" I := 1 "STEP" 1 "UNTIL" K "DO"

"BEGIN" DIAGI := DIAG [I] :=

SQRT (C [COUNTER2 * I]);

"FOR" J := 1 "STEP" 1 "UNTIL" I "DO"

"BEGIN" COUNTER2 := COUNTER2 + 1;

C [COUNTER2] := C [COUNTER2] / DIAGI

/DIAG [J]

"END"

"END"

TRIANGLE (OC, "("CORRELATION MATRIX")", C, -K, VARNR);

PUT ARRAY ("IF" COVCORR = COWM "THEN" CORRM "ELSE"

"IF" COVCORR = REDCROW "THEN" REDCORR

"ELSE" TT, C);

REWIND (COVCORR); GET ARRAY (COVCORR, C);

"END"

"ELSE"

TRIANGLE (OC, "("CORRELATION MATRIX")", C,
-K, VARNR);

TRACE:= O; COUNTER2:= O; DET:= AUX [31];
"FOR" I:= 1 "STEP" I "UNTIL" K "DO"
"BEGIN" DI:= D [I];
RE:= 1 - 1 / C [COUNTER2 + IJ / DI / DI;
"IF" RE < 0 "THEN" RE:= 0; VEC [I, 31]:= RE;
"FOR" J:= 1 "STEP" 1 "UNTIL" I "DO"
"BEGIN" COUNTER2:= COUNTER2 + 1;
C [COUNTER2]:= C [COUNTER2] * DI * D [J]
"END"
;
TRACE:= TRACE + C [COUNTER2];
VEC [I, 1J]:= DI:= DI * DI; DET:= DET * DI;
VEC [I, 2J]:= SQT (RE)
"END"
;
RECTANGLE (OC,
"("DIAG INV MULT R MULT R*"");
VEC, 1, K, 1, 3, VAL, 0, VARNR, 1);
DUP VEC COL (1, K, 1, HELPEVE, VEC);
TRIANGLE (OC, ("MATRIX S*"), C, K, VARNR);
OUTPUT (OC,
"("3/,BB,"("TRACE:"),BB,-D.30*DD,,/
BB,"("DETERMINANT:"),BB,-D.30*DD,,/
TRACE, DET);
"IF" P > K "THEN" P:= K;
"IF" P = 0 "THEN" "GOTO" NEXTPROB;
EM [OJ]:= "-12; EM [2J]:="-10; EM [4J]:="-3;
EM [6J]:="-8; EM [8J]:= 5;
EIGSIM1 (C, K, P, VAL, VEC, EM);
RECTANGLE (OC,
"("LATENT ROOTS AND VECTORS OF S*"");
VEC, 1, K, 1, P, VAL, 1, VARNR, 1);
COUNT:= 0;
NEXTP: COUNT:= COUNT + 1;
"IF" COUNT < NVAL "THEN"
"BEGIN" P:= VALUE [COUNT];
"IF" P = 0 "THEN"
"BEGIN" P:= -P;
PR FSCORES:= ABS (MATRIX) = 1 "END"
"ELSE" PR FSCORES:= "FALSE";
"IF" P >= K "THEN" "GOTO" NEXTPROB
"END"
"ELSE" "GOTO" NEXTPROB;
OUTPUT (OC,
"("*ZD","(" - FACTOR SOLUTION"),
/20("(m"), 2")", P);
RE:= (TRACE - SUM (I, 1, P, VAL [II]) / (K - P);
"FOR" I:= 1 "STEP" I "UNTIL" K "DO"
"BEGIN" DI:= D [I];
"FOR" J:= 1 "STEP" 1 "UNTIL" P "DO" L [I, J]:= LR [I, J]:= VEC [I, J] *
SQT (VAL [J] - RE) / DI
"END"
;
"FOR" I:= 1 "STEP" 1 "UNTIL" P "DO" KOM2 [II]:= TAMMAT (1, K, I, 1, L, L);
VV:= SUM (I, 1, P, KOM2 [II]) * 100 /
"("GUTTMAN-CRITERION AFTER ROTATION")"
   KOM, 1, P, 2, 2, KOM2, 0, FACNR, 1)
"END";
OUTPUT (OC, "("4/, 8B,
"("PERCENTAGE OF VARIANCE EXPLAINED:"),
3ID.DD, ")", VV)
"END";
NOR1: COUNTER2 := 0; REWIND (COVCORR);
GET ARRAY (COVCORR, C);
"FOR" I := 1 "STEP" 1 "UNTIL" K "DO"
"BEGIN" "FOR" J := 1 "STEP" 1 "UNTIL" I "DO"
   "BEGIN" DI := MATTAM (1, P, I, J, L, L);
   C [COUNTER2] := C [COUNTER2] + 1;
   KOM [I, J] := DI
"END";
"IF" MATRIX = 3 "THEN" "GOTO" CONTINUE;
"IF" MATRIX > 0 "THEN"
"BEGIN" TRAPEZ (OC, "("COMMUNALITIES")",
   KOM, 1, K, 1, 1, KOM2, 0, VARNR, 1);
   TRIANGLE (OC,
   "("RESIDUAL COVARIANCE MATRIX")", C, K,
   VARNR);
"END";
"FOR" I := 1 "STEP" 1 "UNTIL" P "DO"
"BEGIN" DIAGI := DIAG [I];
   "FOR" J := 1 "STEP" 1 "UNTIL" P "DO"
"END"
"END";
"FOR" I := 1 "STEP" 1 "UNTIL" P "DO"
   KOM2 [I, L] := TAMMAT (1, K, I, L, L);
   SYSPARAM (OC, 3, I);
   "IF" MATRIX > 0 "OR" MATRIX = -3
   "THEN" OUTPUT (OC, "(""a"")")
   "ELSE" "IF" I > 3 "THEN" OUTPUT (OC, "(""3/"")");
OUTPUT (OC,
   "(""BASED ON CORRELATION MATRIX:"), ,
   2BC("="), ")
RECTANGLE (OC,
   "("CANNONICAL FACTORLOADINGS")",
   L, 1, K, 1, P, KOM2, 1, VARNR, 1);
"FOR" I := 1 "STEP" 1 "UNTIL" P "DO"
   KOM [I, 2] := 1 - 2 * RE / VAL [I];
   RECTANGLE (OC, "("GUTTMAN-CRITERION")",
   KOM, 1, P, 2, 2, KOM2, 0, FACNR, 1);
"FOR" J := 1 "STEP" 1 "UNTIL" P "DO"
   KOM2 [J, I] := TAMMAT (1, K, J, J, L, LR);
   VV := SUM (I, 1, P, KOM2 [I]) * 100 / K;
   "IF" P - P1 < 1 "THEN"
   "BEGIN"
OUTPUT (OC, "(""4/, 8B,""("PERCENTAGE OF "),
"END"),
"("VARIANCE EXPLAINED:")", JZD, DD, ");", VV); "GOTO" NOR2 "END"; RECTANGLE (OC, "("TRANSFORMATION MATRIX")", ROTMAT, 1, P, 1, P, DUPFACNR, 2, FACNR, 1); OUTPUT (OC, "("/", BB, "("NUMBER OF ITERATIONS:")", JZD, ");", HI); RECTANGLE (OC, "("VARIMAX ROTATED FACTOR LOADINGS")", LR, 1, K, 1, P, KOM2, 1, VARNR, 1); OUTPUT (OC, "("/", BB, "("NUMBER OF UNROTATED FACTORS:")", JZD, ");", P1 - 1); "BEGIN" "REAL" "ARRAY" G [1 : P]; DUPVEC COL (1, P, 2, 6, KOM); "FOR" I := 1 "STEP" 1 "UNTIL" P "DO" KOM [I, 2] := SUM (J, 1, P, G [J] * ROTMAT [J, I] ** 2); RECTANGLE (OC, "("GUTTMAN- CRITERION AFTER ROTATION")", KOM, 1, P, 2, 2, KOM2, 0, FACNR, 1) "END"; OUTPUT (OC, "("/", BB, "("PERCENTAGE OF VARIANCE EXPLAINED:")", JZD, DD, ");", VV); NOR2: COUNTER2 := 0; "FOR" I := 1 "STEP" 1 "UNTIL" K "DO" "BEGIN" DIAGI := DIAG [I]; "FOR" J := 1 "STEP" 1 "UNTIL" I "DO" "BEGIN" COUNTER2 := COUNTER2 + 1; C [COUNTER2] := C [COUNTER2] / DIAGI / DIAG [J] "END"; KOM [I, 1] := KOM [I, 1] / DIAGI / DIAGI; "END"; CONTINUE: RECTANGLE (OC, "("COMMUNALITIES")", KOM, 1, K, 1, 1, KOM2, 0, VARNR, 1); TRIANGLE (OC, "("RESIDUAL CORRELATION MATRIX")", C, K, VARNR); REI := VAL [I]; "FOR" I := 2 "STEP" 1 "UNTIL" P "DO" REI := REI + VAL [I]; REI := DET / REI; "IF" PR FScores "THEN" "BEGIN" "REAL" X; "INTEGER" COUNTER3; "REAL" "ARRAY" FSM [1 : HK, 1 : P]; REWIND (FSM); "FOR" I := 1 "STEP" 1 "UNTIL" K "DO" ROW CST (1, P, I, L, HELPVEC [II] * DIAG [I] ** 2); "FOR" I := 1 "STEP" 1 "UNTIL" P "DO" "BEGIN" X := VAL [I];
COL CST (1, K, I, L, 1 / SQRT (X * (X - RE)))

"END"
COUNTER3 := 0;
"FOR" I := 1 "STEP" 1 "UNTIL" HK "DO"
"BEGIN" "IF" BCC [I] "THEN"
"BEGIN" COUNTER3 := COUNTER3 + 1;
"FOR" J := 1 "STEP" 1 "UNTIL" P "DO"
FSM [I, J] := MATMAT (1, P, COUNTER3, J, L, ROTMAT)
"END"
"ELSE"
"FOR" J := 1 "STEP" 1 "UNTIL" P "DO" FSM [I, J] := 0
"END"
FACTOR SCORES (STANDARD, FSM, HK, P, NRESP, FSCORES);
OUTPUT_OC, "(\"\")";
RECTANGLE (OC,
"(FACTOR SCORE COEFFICIENTS, BASED ON (ROTATED) F-NORM)",
FSM, 1, HK, 1, P, DUPFACNR, 2, ICC, 1);
BIGPRINT ("(FACTOR SCORES, BASED ON (ROTATED) F-NORM)",
FSCORES, P, NRESP, DUPFACNR, RR)
"END"
OUTPUT OC, "("","(TEST OF FIT)" "/,12("","="),",",(MEAN OF THE "),22D,
("LOWEST LATENT ROOTS "),
("OF S-STAR:"),B,-D.3D*DD,","
("U-STATISTIC:"),35B,-D.3D*DD,",
("DEGREES OF FREEDOM:"),26B,32D,/"
K - P, RE, ((K - P) * LN (RE) - LN (RE1))
* (NRESP - 1), (K - P + 2) // 2)
"GOTO" NEXTP;
"END"
"END"
DUMP: OUTPUT OC, "("","10/,2DB,"(*** EOF / EOR ENCOUNTERED ON CHANNEL "),
1D,"(*** "),,2DB,"(*** "),42DB,
("RESPONDENTS HAVE BEEN READ ***")",IN, RESP);
EXIT;
NEXTPROB: OUTPUT OC, "("","/("END OF JORESKOG")",a")")
"END" JORESKOG;
"EOP"
2.3.1 Bin Low Bound

TITLE: Bin Low Bound

AUTHOR: J. Bethlehem

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION
The procedure computes the lower bound of the confidence interval for the probability $p$, given the number of successes in a sequence of $n$ independent experiments with probability $p$ of success.

KEYWORDS
Lower confidence bound for a probability

CALLING SEQUENCE
Heading
"REAL" "PROCEDURE" BIN LOW BOUND (x, n, alpha);
"VALUE" x, n, alpha;
"REAL" x, n, alpha;
"CODE" 42420;
Formal parameters
x: <arithmetic expression>, number of successes;
n: <arithmetic expression>, number of experiments;
alpha: <arithmetic expression>, one minus confidence coefficient.

DATA AND RESULTS
The value of the lower bound is assigned to the procedure identifier BIN LOW BOUND.
The following error messages may appear:
Error number 0 (if no lower bound is found)
Error number 1 (if $x$ is not an integer $\geq 0$, or $x > n$)
Error number 2 (if $n$ is not an integer $\geq 0$)
Error number 3 (if $\alpha < 0$ or $\alpha > 1$)

PROCEDURES USED
STATAL3 ERROR
BOUND
ZERODINDER

LANGUAGE
Algol 60
METHOD AND PERFORMANCE
The lower bound is computed by finding a solution \( p \) of the equation
\[
\sum_{k=x}^{N} \binom{N}{k} p^k (1-p)^{N-k} = \alpha.
\]
The precision is \( 10^{-10} \).

EXAMPLE OF USE

Program:
"BEGIN"
OUTPUT(61, "("3(2D,/)")")",
BIN LOW BOUND( 15, 20, .050),
BIN LOW BOUND( 34, 85, .025),
BIN LOW BOUND(102, 107, .100))
"END"

Output:
.544418
.295194
.915046

SOURCE TEXT
"CODE" 42420;
"REAL" "PROCEDURE" BIN LOW BOUND(X, NN, ALPHA);
"VALUE" X, NN, ALPHA; "REAL" X, NN, ALPHA;
"BEGIN" "INTEGER" N, T; "REAL" TOL;

N:= ENTIER(NN);
"IF" N < NN "OR" N < 1 "THEN"
STALAT3 ERROR("("BIN LOW BOUND")", 2, NN);
T:= ENTIER(X);
"IF" T < O "OR" T > N "OR" T < X "THEN"
STALAT3 ERROR("("BIN LOW BOUND")", 1, X);
"IF" ALPHA <= O "OR" ALPHA >= 1 "THEN"
STALAT3 ERROR("("BIN LOW BOUND")", 3, ALPHA);
TOL:= "-10;"
BIN LOW BOUND:=
"IF" T = 0 "THEN" 0 "ELSE"
"IF" T = N "THEN" ALPHA ** (1 / N) "ELSE"
"IF" T <= (N + 1) / 2 "THEN"
1 - BOUND(T - 1, N, 1 - ALPHA, TOL, "("BIN LOW BOUND")")
"ELSE" BOUND(N - T, N, ALPHA, TOL, "("BIN LOW BOUND")")
"END" BIN LOW BOUND;
"EOP"
2.3.2

TITLE: Bin upp bound

AUTHOR: J. Bethlehem

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION
The procedure computes the upper bound of the confidence interval for the probability $p$, given the number of successes in a sequence of $n$ independent experiments with probability $p$ of success.

KEYWORDS
Upper confidence bound for a probability

CALLING SEQUENCE

Heading
"REAL" "PROCEDURE" BIN UPP BOUND (X, N, ALPHA);
"VALUE" X, N, ALPHA;
"REAL" X, N, ALPHA;
"CODE" 42421;

Formal parameters
X: <arithmetic expression>, number of successes;
N: <arithmetic expression>, number of experiments;
ALPHA: <arithmetic expression>, one minus confidence coefficient.

DATA AND RESULTS
The value of the upper bound is assigned to the procedure identifier BIN UPP BOUND.
The following error messages may appear:

Error number 1 (if $x$ is not an integer $\geq 0$, or $x > n$)
Error number 2 (if $n$ is not an integer $> 0$)
Error number 3 (if $\alpha \leq 0$ or $\alpha > 1$)
Error number 0 (if no upper bound is found)

PROCEDURES USED
STATAL3 ERROR STATAL 40100
BOUND STATAL BOUND
ZEROINDER NUMAL 34453
Bin upp bound

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
The upper bound is computed by finding a solution $p$ of the equation

$$\sum_{k=0}^{N} p^k (1-p)^{N-k} = \alpha.$$ 

The precision is $10^{-10}$.

EXAMPLE OF USE

Program:

"BEGIN"
   OUTPUT(61, "("3(Z,6D,/)")"),
   BIN UPP BOUND(15, 20, .050),
   BIN UPP BOUND(34, 85, .025),
   BIN UPP BOUND(102, 107, .100)
"END"

Output:

.895919
.511994
.977096

SOURCE TEXT

"CODE" 42421;
"REAL" "PROCEDURE" BIN UPP BOUND(X, NN, ALPHA);
"VALUE" X, NN, ALPHA; "REAL" X, NN, ALPHA;
"BEGIN" "INTEGER" N, T; "REAL" TOL;

N := ENTIER(NN);
"IF" N < NN "OR" N < 1 "THEN"
   STATAL3 ERROR("BIN UPP BOUND")", 2, NN);
T := ENTIER(X);
"IF" T < 0 "OR" T > N "OR" T < X "THEN"
   STATAL3 ERROR("BIN UPP BOUND")", 1, X);
"IF" ALPHA <= 0 "OR" ALPHA >= 1 "THEN"
   STATAL3 ERROR("BIN UPP BOUND")", 3, ALPHA);
TOL := "-10;
BIN UPP BOUND :=
   "IF" T = 0 "THEN" 1 - ALPHA ** (1 / N) "ELSE"
   "IF" T = N "THEN" 1 "ELSE"
   "IF" T <= (N - 1) / 2 "THEN"
   1 - BOUND(T, N, ALPHA, TOL, "BIN UPP BOUND")"
   "ELSE" BOUND(N - T - 1, N, 1 - ALPHA, TOL,
   "BIN UPP BOUND")";
"END" BIN UPP BOUND;
"EOP"
2.4.1

TITLE: Sample Des

AUTHOR: A. Nonymous

INSTITUTE: Mathematical Centre

RECEIVED: 1981/1982

BRIEF DESCRIPTION
For a sample of independent observations x[lx1], ..., x[lx3], the procedure 
computes the following 12 descriptive statistics: mean, variance, standard deviation, 
standard error of the mean, third and fourth central moments, skewness, 
kurtosis, minimum, maximum, median and range.

KEYWORDS
Descriptive statistics

CALLING SEQUENCE
Heading
"PROCEDURE" SAMPLE DES (X, LX, UX, SORTED, STATISTICS);
"VALUE" LX, UX;
"ARRAY" X, STATISTICS;
"INTEGER" LX, UX;
"BOOLEAN" SORTED;
"CODE" 42430;

Formal parameters
X: <array identifier>, vector containing the sample
x[lx1], ..., x[lx3];
LX: <integer arithmetic expression>, smallest index of the
sample;
UX: <integer arithmetic expression>, largest index of the sam-
ple;
SORTED: <boolean expression>, indicating whether the sample is
sorted in non-decreasing order or not;
STATISTICS: <array identifier>, output parameter, array of dimen-
sion [1:12] which at exit contains the statistics.

DATA AND RESULTS
After a procedure call the array elements STATISTICS [1], ..., STATISTICS[12]
contain the mean, variance, standard deviation, standard error of the mean, 
third central moment, forth central moment, skewness, kurtosis, minimum, 
maximum, median, and range respectively of the sample.
The following error message may appear:
Errornumber 3 (if LX > UX)
PROCEDURES USED
VEC QSORT STATA L1020
STATAL3 ERROR STATA L40100

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
The computation of the statistics is straightforward. (The sample variance is
given by \( N^{-1} \sum (x_i - \text{mean})^2 \), where \( N = \text{ UX } - \text{ LX } + 1 \). In order to obtain more
accurate results, computations are performed on a linear transformation of the
data.

EXAMPLE OF USE

Program:

```
BEGIN
"ARRAY" X[1:10], STAT[1:12];
"BOOLEAN" SORTED;
SORTED := "TRUE";
INARRAY(60, X);
SAMPLE DESC(X, 1, 10, SORTED, STAT);
OUTPUT(61, """"("MEAN
+52D.6D, "/""("VARIANCE
+52D.6D, "/""("STANDARD DEVIATION
+52D.6D, "/""("STANDARD ERROR OF THE MEAN
+52D.6D, "/""("THIRD CENTRAL MOMENT
+52D.6D, "/""("FOURTH CENTRAL MOMENT
+52D.6D, "/""("SKEWNESS
+52D.6D, "/""("KURTOSIS
+52D.6D, "/""("MINIMUM
+52D.6D, "/""("MAXIMUM
+52D.6D, "/""("MEDIAN
+52D.6D, ""("RANGE
+52D.6D")", STAT)

"END"
```

Input:

.1 .2 .3 .4 .5 .6 .7 .8 .9 1.0
2.4.1.

Output:

- Mean: +0.550000
- Variance: +0.082500
- Standard Deviation: +0.287228
- Standard Error of the Mean: +0.090830
- Third Central Moment: -0.000000
- Fourth Central Moment: +0.012086
- Skewness: -0.000000
- Kurtosis: -1.224242
- Minimum: +0.100000
- Maximum: +1.000000
- Median: +0.550000
- Range: +0.900000

Source Text

"code" 42430;

"procedure" sample desc(sample, l, u, sorted, statistics);
"value" l, u; "array" sample, statistics; "integer" l, u;
"boolean" sorted;
"begin" "integer" i, n, l plus u;
"real" minimum, maximum, range, mean, median,
variance, standard deviation, standard error,
skewness, kurtosis, undefined, y, y1, s1, s2,
s3, s4, y mean, y variance,
y standard deviation, y mean 2, stat, m3, m4,
y m3, y m4, range2;

"comment" constant;
undefined:= max real;

"comment" parameter checking;
"if" l > u "then" statal error(""sample desc"", 3, u);

"comment" conditional sorting;
"if" "not" sorted "then"
"begin" vecq sort(sample, l, u); sorted:= "true" "end";
un:= u - l + 1; minimum:= sample[1]; maximum:= sample[un];
range:= maximum - minimum;
"if" range = 0 "then"
"begin" mean:= median:= minimum;
variance:= standard deviation:= standard error:=
m3:= m4:= 0;
skewness:= kurtosis:= undefined;
"else" "else"
"begin" l plus u:= l + u; i:= l plus u // 2;
median:= "if" l plus u = i * 2
"then" sample[i]
"else" sample[1] + sample[i + 1]) / 2;

"comment" build up sums of powers of
y = (x - median) / range ;
s1:= s2:= s3:= s4:= 0;
"FOR" I := I + 1 "UNTIL" U "DO"
"BEGIN" Y := (SAMPLE[I] - MEDIAN) / RANGE;
    S1 := S1 + Y; Y1 := Y * Y; S2 := S2 + Y1;
    Y11 := Y1 * Y; S3 := S3 + Y1; S4 := S4 + Y1 * Y;
"END"

"COMMENT" COMPUTE STATISTICS IN TERMS OF THE Y'S;
Y MEAN := S1 / N; Y MEAN 2 := Y MEAN * Y MEAN;
Y VARIANCE := S2 / N - Y MEAN 2;
Y STANDARD DEVIATION := SQRT(Y VARIANCE);
Y M3 := (S3 - 3 * Y MEAN * S2) / N + 2 * Y MEAN 2 * Y MEAN;
SKKEWNESS := Y M3 / Y VARIANCE / Y STANDARD DEVIATION;
Y M4 := (S4 - 4 * Y MEAN * S3 + 6 * Y MEAN 2 * S2) / N - 3 * Y MEAN 2 * Y MEAN 2;
KURTOSIS := Y M4 / Y VARIANCE / Y VARIANCE - 3;

"COMMENT" COMPUTE THE REQUIRED STATISTICS BY
EXECUTING THE LINEAR TRANSFORMATION IN
THE REVERSE DIRECTION.
SKEWNESS AND KURTOSIS ARE INVARIANT;
MEAN := Y MEAN * RANGE + MEDIAN;
RANGE2 := RANGE * RANGE;
VARIANCE := Y VARIANCE * RANGE2;
STANDARD DEVIATION := Y STANDARD DEVIATION * RANGE;
STANDARD ERROR := STANDARD DEVIATION / SQRT(N);
M3 := Y M3 * RANGE2 * RANGE;
M4 := Y M4 * RANGE2 * RANGE2;
"END"

"COMMENT" STORING THE STATISTICS;
I := 0;
"FOR" STAT := MEAN, VARIANCE, STANDARD DEVIATION,
STANDARD ERROR, M3, M4, SKEWNESS,
KURTOSIS, MINIMUM, MAXIMUM, MEDIAN, RANGE
"DO" "BEGIN" I := I + 1; STATISTICS[I] := STAT "END"
"END" SAMPLE DES;
"EOP"
2.4.2

TITLE: Freqtab Des

AUTHOR: A. Nonymous

INSTITUTE: Mathematical Centre

RECEIVED: 1981/1982

BRIEF DESCRIPTION
For a frequency table (histogram), the procedure computes the following 14 descriptive statistics: mean, variance, standard deviation, standard error of the mean, third and fourth central moments, skewness, kurtosis, minimum, maximum, median, range, mode, and total number of observations.

KEYWORDS
Descriptive statistics

CALLING SEQUENCE
Heading
"PROCEDURE" FREQTAB DES (VALUE, NUMBER, L, U, SORTED, STATISTICS);
"VALUE" L, U;
"ARRAY" VALUE, STATISTICS;
"INTEGER" "ARRAY" NUMBER;
"INTEGER" L, U;
"BOOLEAN" SORTED;
"CODE" 42431;

Formal parameters
VALUE: < array identifier>, vector containing the (different) observed values VALUE[I],...,VALUE[U];
NUMBER: <integer array identifier>, vector containing the corresponding frequencies, NUMBER[I] contains the number of times VALUE[I] is observed for I=L,...,U;
L: <integer arithmetic expression>, smallest index of VALUE and NUMBER;
U: <integer arithmetic expression>, largest index of VALUE and NUMBER;
SORTED: <boolean expression>, indicating whether the array VALUE is sorted in non decreasing order or not;

DATA AND RESULTS
After a procedure call the array elements STATISTICS[1],...,STATISTICS[14] contain the mean, variance, standard deviation, standard error of the mean, third central moment, fourth central moment, skewness, kurtosis, minimum, maximum, median, range, mode, and total number of observations respectively.
The following error messages may appear:
Error number 4  (if \( t > 0 \))
Error number 2  (if a frequency < 0)

**PROCEDURES USED**

**STATAL3 ERROR**        **STATAL 40100**

**LANGUAGE**
Algol 60

**METHOD AND PERFORMANCE**
The computation of the statistics is straightforward. (The sample variance is given by \( N^{-1} \sum_{i=1}^{N} (x_i - \bar{x})^2 \), where \( N \) is the total number of observations). In order to make results more accurate, computations are performed on linear transformations of the data.

**EXAMPLE OF USE**

**Program:**

```
"BEGIN"
"ARRAY" X[1:10], STAT[1:14];
"INTEGER" "ARRAY" NUMBER[1:10]; "BOOLEAN" SORTED;
SORTED := "TRUE"; INARRAY(G0, X);
ININTARRAY(G0, NUMBER);
FREQTAB DES(X, NUMBER, 1, 10, SORTED, STAT);
OUTPUT61, ""(""MEAN = "");
+5ZD.6D, "/"(""VARIANCE = "");
+5ZD.6D, "/"(""STANDARD DEVIATION = "");
+5ZD.6D, "/"(""STANDARD ERROR OF THE MEAN = "");
+5ZD.6D, "/"(""THIRD CENTRAL MOMENT = "");
+5ZD.6D, "/"(""FOURTH CENTRAL MOMENT = "");
+5ZD.6D, "/"(""SKEWNESS = "");
+5ZD.6D, "/"(""KURTOSIS = "");
+5ZD.6D, "/"(""MINIMUM = "");
+5ZD.6D, "/"(""MAXIMUM = "");
+5ZD.6D, "/"(""MEDIAN = "");
+5ZD.6D, "/"(""RANGE = "");
+5ZD.6D, "/"(""MODE = "");
+5ZD.6D, "/"(""TOTAL NUMBER OF OBSERVATIONS = "");
+5ZD.6D")", STAT)
"END"
```
2.4.2  Freqtab Des

Input:

```
  1  2  3  4  5  6  7  8  9  10
  1  2  3  4  5  5  4  3  2  1
```

Output:

```
MEAN = +0.550000
VARIANCE = +0.109259
STANDARD DEVIATION = +0.221736
STANDARD ERROR OF THE MEAN = +0.040483
THIRD CENTRAL MOMENT = -0.000000
FOURTH CENTRAL MOMENT = +0.027916
SKEWNESS = -0.000000
KURTOSIS = -0.661534
MINIMUM = +0.100000
MAXIMUM = +1.000000
MEDIAN = +0.550000
RANGE = +0.900000
MODE = +0.600000
TOTAL NUMBER OF OBSERVATIONS = +30.000000
```

Source text:

"CODE" 42431;
"PROCEDURE" FREQTAB DES(VALUE, NUMBER, L, U, SORTED, STATISTICS);
"VALUE" L, U;
"ARRAY" VALUE, STATISTICS;
"INTEGER" "ARRAY" NUMBER;
"INTEGER" L, U;
"BOOLEAN" SORTED;
"BEGIN" "INTEGER" I, N, N OF STATISTICS, FREQUENCY, MAX FREQ, CUM FREQ;
  "REAL" MINIMUM, MAXIMUM, RANGE, MEAN, MEDIAN, MODE, VARIANCE, STANDARD DEVIATION, STANDARD ERROR, SKEWNESS, KURTOSIS, UNDEFINED, Y, Y1, S1, S2, S3, S4, Y MEAN, Y VARIANCE, Y STANDARD DEVIATION, Y MEAN Z, STAT, N2, M3, M4, Y M3, Y M4, RANGE2;

"PROCEDURE" SORT(V1, V2, LV, UV);
"VALUE" LV, UV;
"INTEGER" LV, UV;
"ARRAY" V1;
"INTEGER" "ARRAY" V2;
"BEGIN" "INTEGER" P, Q, IX, IZ; "REAL" X, XX, Y, ZZ, Z;

"PROCEDURE" VEC2SORT;
"BEGIN" "INTEGER" L, U; L:= LV; U:= UV;
PART: P:= L; Q:= U; X:= V1[P]; Z:= V1[Q];
"IF" X > Z "THEN"
  "BEGIN" Y:= X; V1[P]:= X:= Z; V1[Q]:= Z:= Y;
  Y:= V2[P]; V2[Q]:= Y:= V2[Q]; V2[P]:= Y; V2[Q]:= Y

305
"ENDIF"
ENDIF U - L > 1 "THEN"
BEGIN XX := X; IX := P; ZZ := I; IZ := Q;
LEFT: "FOR" P := P + 1 "WHILE" P < Q "DO"
BEGIN X := V1[P];
ENDIF X >= XX "THEN" "GOTO" RIGHT "END";
P := Q - 1; "GOTO" OUT;
RIGHT: "FOR" Q := Q - 1 "WHILE" Q > P "DO"
BEGIN Z := V1[Q];
ENDIF Z <= ZZ "THEN" "GOTO" DIST "END";
Q := P; P := P - 1; Z := X; X := V1[P];
DIST: "IF" X > Z "THEN"
Y := V2[P]; V2[P] := V2[Q]; V2[Q] := Y "END";
ENDIF X >= XX "THEN" "BEGIN" XX := X; IX := P "END";
ENDIF Z <= ZZ "THEN" "BEGIN" ZZ := Z; IZ := Q "END";
"GOTO" LEFT;
OUT: "IF" P > IX "AND" X < XX "THEN"
BEGIN V1[IX] := XX; V1[IX] := X;
Y := V2[P]; V2[P] := V2[IX]; V2[IX] := Y "END";
ENDIF Q < IZ "AND" Z > ZZ "THEN"
BEGIN V1[Q] := ZZ; V1[IZ] := Z;
Y := V2[Q]; V2[Q] := V2[IZ]; V2[IZ] := Y "END";
ENDIF U > P - L "THEN"
BEGIN LV := L; UV := P - 1;
L := Q + 1 "END" "ELSE"
BEGIN LV := U; UV := Q + 1; U := P - 1 "END";
ENDIF UV > LV "THEN" VEC2SORT;
ENDIF U > L "THEN" "GOTO" PART "END" U - L > 1 "END" VEC2SORT;
"IF" UV > LV "THEN" VEC2SORT "END" SORT;

"COMMENT" CONSTANTS;
N OF STATISTICS := 14; UNDEFINED := MAX REAL;

"COMMENT" PARAMETER CHECKING.
COMPUTATION OF THE MODE;
ENDIF L > U "THEN" STATAL3ERROR("FREQTAB DES"), 4, U);
N := 0; MAX_FREQ := -1;
"FOR" I := L "STEP" 1 "UNTIL" U "DO"
"BEGIN" FREQUENCY := NUMBER[I];
"IF" FREQUENCY < 0 "THEN"
STATALERROR(""FREQTAB DES""), 2, FREQUENCY);
"IF" FREQUENCY >= MAX_FREQ "THEN"
"BEGIN" MAX_FREQ := FREQUENCY; MODE := VALUE[I] "END";
N := N + FREQUENCY
"END";

"IF" N = 0 "THEN"
"BEGIN"
"FOR" I := 1 "STEP" 1 "UNTIL" N OF STATISTICS - 1 "DO"
STATISTICS[I] := UNDEFINED;
STATISTICS[I] := 0;
"END" "ELSE"
"BEGIN" "COMMENT" CONDITIONAL SORTING;
"IF" NOT "SORTED" "THEN"
"BEGIN" SORT(VALUE, NUMBER, L, U);
SORTED := "TRUE"
"END";

"COMMENT" MINIMUM, MAXIMUM AND RANGE;
I := L - 1;
"FOR" I := I + 1 "WHILE" NUMBER[I] = 0 "DO"
MINIMUM := VALUE[I]; L := I;
I := U + 1;
"FOR" I := I - 1 "WHILE" NUMBER[I] = 0 "DO"
MAXIMUM := VALUE[I]; U := I;
RANGE := MAXIMUM - MINIMUM;
"IF" RANGE = 0 "THEN"
"BEGIN" MEAN := MEDIAN := MINIMUM;
VARIANCE := STANDARD DEVIATION := STANDARD ERROR :=
M3 := M4 := 0;
SKEWNESS := KURTOSIS := UNDEFINED;
"END" "ELSE"
"BEGIN" "COMMENT" COMPUTE THE MEDIAN;
N2 := N / 2; CUM_FREQ := 0; I := L - 1;
"FOR" I := I + 1 "WHILE" CUM_FREQ < N2 "DO"
CUM_FREQ := CUM_FREQ + NUMBER[I];
I := I - 1; MEDIAN := VALUE[I];
"IF" CUM_FREQ = N2 "THEN"
"BEGIN"
"FOR" I := I + 1 "WHILE" NUMBER[I] = 0 "DO"
MEDIAN := (MEDIAN + VALUE[I]) / 2;
"END";

"COMMENT" BUILD UP SUMS OF POWERS OF
Y := (X - MEDIAN) / RANGE;
S1 := S2 := S3 := S4 := 0;
"FOR" I := L "STEP" 1 "UNTIL" U "DO"
"BEGIN" FREQUENCY := NUMBER[I];
"IF" FREQUENCY > 0 "THEN"
"BEGIN" Y:= (VALUE[I] - MEDIAN) / RANGE;
  S1:= S1 + FREQUENCY * Y; Y1:= Y * Y;
  S2:= S2 + FREQUENCY * Y1; Y1:= Y1 * Y;
  S3:= S3 + FREQUENCY * Y1;
  S4:= S4 + FREQUENCY * Y1 * Y;
"END";

"COMMENT"
COMPUTE STATISTICS IN TERMS OF THE Y'S;
  Y MEAN:= S1 / N; Y MEAN 2:= Y MEAN * Y MEAN;
  Y VARIANCE:= S2 / N - Y MEAN 2;
  Y STANDARD DEVIATION:= SQRT(Y VARIANCE);
  Y M3:= (S3 - 3 * Y MEAN * S2) / N + 2 * Y MEAN 2 + Y MEAN;
  SKEWNESS:= Y M3 / Y VARIANCE /
               Y STANDARD DEVIATION;
  Y M4:= (S4 - 4 * Y MEAN * S3 + 6 * Y MEAN 2 * S2) / N -
          3 * Y MEAN 2 * Y MEAN 2;
  KURTOSIS:= Y M4 / Y VARIANCE /
             Y VARIANCE - 3;

"COMMENT" COMPUTE THE REQUIRED STATISTICS
BY EXECUTING THE LINEAR
TRANSFORMATION IN THE REVERSE
DIRECTION.
SKEWNESS AND KURTOSIS ARE
INVARIANT;
  MEAN:= Y MEAN * RANGE + MEDIAN;
  RANGE2:= RANGE + RANGE;
  VARIANCE:= Y VARIANCE * RANGE2;
  STANDARD DEVIATION:= Y STANDARD DEVIATION
                    * RANGE;
  STANDARD ERROR:= STANDARD DEVIATION
                   / SQRT(N);
  M3:= Y M3 * RANGE2 * RANGE;
  M4:= Y M4 * RANGE2 * RANGE2;
"END";

"COMMENT" STORE THE STATISTICS;
I:= 0;
"FOR" STAT:= MEAN, VARIANCE, STANDARD DEVIATION,
    STANDARD ERROR, M3, M4, SKEWNESS,
    KURTOSIS, MINIMUM, MAXIMUM, MEDIAN,
    RANGE, MODE, N
"DO" "BEGIN" I:= I + 1; STATISTICS[I]:= STAT "END";
"END"

"END" FREQTAB DES;
"EOP"
3. SORTING & RANKING

This section contains 21 procedures which sort or are related to sorting a list of items $E(L)$ up to and including $E(U)$. An item consists of one or more numerical values. A list of items is contained in a one- or two-dimensional array.

A list is said to be sorted when the items are either in 'non-decreasing' or in 'lexicographical' order. More specifically, in the case that every item $E(i)$ is a single value, we say that the list is sorted when the items are in non-decreasing order. In the case of multiple valued items, e.g.

$$E(i) = (X(i,K), X(i,K+1), \ldots, X(i,N)),$$

where $K$ and $N$ are fixed integers and every $X(i,H)$ is the content of an array element $M[I,H]$ (or $M[H,I]$), we say that

(i) $E(i)$ is 'equal to' $E(j)$ when $X(i,H) = X(j,H)$ for $H = K, \ldots, N$,

(ii) $E(i)$ is 'less than' $E(j)$ when there exists an index $P$, $K \leq P \leq N$, such that $X(i,H) = X(j,H)$ for $H = K, \ldots, P-1$ and $X(i,P) < X(j,P)$,

(iii) the list $E(L), \ldots, E(U)$ is sorted ('lexicographically') if for every pair of indices $(i,j)$, $L \leq i < j \leq U$ implies that $E(i)$ is 'less than' or 'equal to' $E(j)$.

With regard to their action, 20 of the procedures in this section can be divided into four types:

1. The type 'qsort': rearranges the list $E(L), \ldots, E(U)$ in non-decreasing (lexicographical) order.

2. The type 'indqsort': the list $E(L), \ldots, E(U)$ remains unaltered but the contents of an array $IND[L:U]$ is rearranged such that $E(IND[L:U]), \ldots, E(IND[U])$ is in non-decreasing (lexicographical) order.

3. The type 'perm': the list $E(L), \ldots, E(U)$ is permuted according to the permutation of indices which is given in a integer array IND.

4. The type 'ranktie': the (average) ranks of the items of the list $E(L), \ldots, E(U)$ is delivered in an array $RNK[L:U]$, while leaving $E(L), \ldots, E(U)$ unaltered. Furthermore, the same permutation of indices as in the type 'indqsort' is generated. The sum of the squares and the sum of the cubes of the sizes of the ties is computed.

With regard to the nature of the list of the items $E(L), \ldots, E(U)$ these 20
procedures can also be divided according to five prefixes:

1. If the list is stored as $\text{vLvuJ}, \ldots, \text{vLUvJ}$, (a segment of) a one-dimensional array, the procedure identifiers contain the prefix 'vec'.

2. If the list is stored as $\text{M[R,LCJ]}$, \ldots, $\text{M[R,UCJ]}$, (a segment of) a row of a two-dimensional array, the procedure identifiers contain the prefix 'row'.

3. If the list is stored as $\text{MLR,CJ}$, \ldots, $\text{MLR,CJ}$, (a segment of) a column of a two-dimensional array, the procedure identifiers contain the prefix 'col'.

4. If the list is stored as a matrix, of which the rows are to be reordered lexicographically, the procedure identifiers contain the prefix 'rmat'.

5. If the list is stored as a matrix, of which the columns are to be reordered lexicographically, the procedure identifiers contain the prefix 'cmat'.

In addition there is one more sorting procedure, $\text{VEC2 QSORT}$, which sorts pairs of numbers, stored in two distinct one-dimensional arrays, according to the values of the first components.

Unlike the other Stata procedures, the sorting and ranking procedures do not have any error messages. It is advised to check the values of the parameters before calling a sorting/ranking procedure. If a parameter does not satisfy the conditions of the calling sequence, a call of the procedure has no effect at all.

REFERENCES

[1] M.H. van Emden,
Increasing the efficiency of quicksort,

[2] M.H. van Emden,
Algorithm 402, increasing the efficiency of quicksort,
3.1 SORTING

This section contains six procedures (with suffixes 'QSORT'), which rearrange a list of successive items E(L) up to and including E(U) into non-decreasing (lexicographical) order. The prefixes of the procedure identifier ('VEC', 'ROW', 'COL', 'RMAT', 'CMAT' or 'VEC2') indicate the nature of the items to be processed.

**VEC QSORT** sorts the elements of a vector stored in a one-dimensional array.

**ROW QSORT** sorts the elements of (a segment of) a row of a matrix stored in a two-dimensional array.

**COL QSORT** sorts the elements of (a segment of) a column of such a matrix.

**RMAT QSORT** sorts lexicographically the rows of a matrix stored in a two-dimensional array.

**CMAT QSORT** sorts likewise the columns of such a matrix.

**VEC2 QSORT** sorts the pairs of numbers \((V1[L1], V2[L1])\) into non-decreasing order of the first components (i.e. \(V1[L1] \leq V1[L1+1] \leq \ldots \leq V1[LU1]\)).
Vec Qsort

TITLE: Vec Qsort

AUTHOR: A.C. IJsselstein

INSTITUTE: Mathematical Centre

RECEIVED: 760701

BRIEF DESCRIPTION
The procedure sorts vector elements, stored in a one-dimensional array.

KEYWORDS
Sorting of vector elements

CALLING SEQUENCE

Heading
"PROCEDURE" VEC QSORT (V, LV, UV);
"VALUE" LV, UV;
"INTEGER" LV, UV;
"ARRAY" V;
"CODE" 11020;

Formal parameters
V: <array identifier>, a one-dimensional array V[L:U];
LV, UV: <integer arithmetic expression>, smallest and largest index of
the segment which has to be sorted. LV and UV should satisfy the
condition L <= LV <= UV <= U.

DATA AND RESULTS
After a call of VEC QSORT (V, LV, UV), V[L:U] up to and including V[UV] are
rearranged into non-decreasing order. All other elements of V remain unaltered.

PROCEDURES USED
None

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
Except for notational details, this sorting algorithm is identical to the one
described in van Emden (1970) (see references section 3.1). The number of
operations is of the order (UV−LV+1) * LN(UV−LV+1). No auxiliary arrays are
declared. The recursion depth is at most LN(UV−LV+1) / LN(2).

312
3.1.1 Vec Qsort

**Example of Use**

**Program:**

```
"BEGIN" "ARRAY" X[1..10];
INARRAY(60, X);
VEC QSORT(X, 4, 8);
OUTPUT(61, "("10+(2D2B")")", X)
"END"
```

**Input:**

```
+22 -13 +99 +45 +7 -1 -13 +7 -22 +5
```

**Output:**

```
+22 -13 +99 -13 -1 +7 +7 +45 -22 +5
```

**Source Text**

```
"CODE" 110210;
"PROCEDURE" VECQSORT(V, LV, UV);
"VALUE" LV, UV; "INTEGER" LV, UV; "ARRAY" V;
"BEGIN" "INTEGER" P, Q, IX, IZ; "REAL" X, XX, Y, ZZ, Z;

"PROCEDURE" VECSORT;
"BEGIN" "INTEGER" L, U; L := LV; U := UV;

PART: P:= L; Q:= U; X:= V[P]; Z:= V[Q]; "IF" X > Z "THEN"
"BEGIN" Y:= X; V[P]:= X:= Z; V[Q]:= I:= Y "END";

"IF" U - L > 1 "THEN"
"BEGIN" XX:= X; IX:= P; ZZ:= Z; IZ:= Q;

LEFT: "FOR" P:= P + 1 "WHILE" P < Q "DO"
"BEGIN" X:= V[P];
"IF" X >= XX "THEN" "GOTO" RIGHT "END";
P:= Q - 1; "GOTO" OUT;

RIGHT: "FOR" Q:= Q - 1 "WHILE" Q > P "DO"
"BEGIN" Z:= V[Q];
"IF" Z <= ZZ "THEN" "GOTO" DIST "END";
Q:= P; P:= P - 1; Z:= X; X:= V[P];

DIST: "IF" X > Z "THEN"
"BEGIN" Y:= X; V[P]:= X:= Z; V[Q]:= Z:= Y "END";
"IF" X > XX "THEN" "BEGIN" XX:= X; IX:= P "END";
"IF" Z < ZZ "THEN" "BEGIN" ZZ:= Z; IZ:= Q "END";
"GOTO" LEFT;

OUT: "IF" P > IX "AND" X < XX "THEN"
"BEGIN" V[PI]:= XX; V[IX]:= X "END";
```

313
"IF" Q < IZ "AND" Z > IZ "THEN"
"BEGIN" V[Q]:= Z; V[IZ]:= Z "END";
"IF" U - Q > P - L "THEN"
"BEGIN" LV:= L; UV:= P - 1; L:= Q + 1 "END"
"ELSE"
"BEGIN" UV:= U; LV:= Q + 1; U:= P - 1 "END";
"IF" UV > LV "THEN" VECSORT;
"IF" U > L "THEN" "GOTO" PART
"END" U - L > 1
"END" OF VECSORT;

"IF" UV > LV "THEN" VECSORT
"END" OF VECSORT;
"EOP"
3.1.2
Row Qsort

Title: Row Qsort

Author: A.C. IJsselstein

Institute: Mathematical Centre

Received: 760701

Brief Description
The procedure sorts elements of (a segment of) a matrix row, stored in a two-
dimensional array.

Keywords
sorting of elements of a matrix row

Calling Sequence
Heading
"procedure" row qsort (m, r, lc, uc);
"value" r, lc, uc;
"integer" r, lc, uc;
"array" m;
"code" 11030;

Formal Parameters
m: <array identifier>, a two-dimensional array m[l1:u1, l2:u2];
r: <integer arithmetic expression>, the index of the row of m, in
which the sorting has to be performed. r should satisfy the condi-
tion l1 ≤ r ≤ u1;

lc, uc: <integer arithmetic expression>, smallest and largest index of
the segment of row r which has to be sorted. lc and uc should
satisfy the condition l2 ≤ lc ≤ uc ≤ u2.

Data and Results
After a call of row qsort (m, r, lc, uc), m[r, lc] up to and including m[r, uc]
are rearranged into non-decreasing order. All other elements of m remain unal-
tered.

Procedures Used
None

Language
Algol 60

315
Method and Performance

The sorting algorithm described in van Emde (1970), (see references section 3.1.), is used on successive elements of a matrix row. The number of operations is of the order \((uc – lc + 1) \times ln(uc – lc + 1)\). No auxiliary arrays are declared. The recursion depth is at most \(ln(uc – lc + 1) / ln(2)\).

Example of Use

Program:

"BEGIN" "ARRAY" Z[1:3, 1:9];
INARRAY(60, Z);
ROW QSORT(Z, 2, 2, 9);
OUTPUT(61, "("3(9+ZD2B)/)"), Z)
"END"

Input:

\[+1 \quad +2 \quad +3 \quad +4 \quad +5 \quad +6 \quad +7 \quad +8 \quad +9
+2 \quad +27 \quad +26 \quad +30 \quad +28 \quad +25 \quad +27 \quad +25 \quad +25
+3 \quad +19 \quad +18 \quad +17 \quad +16 \quad +15 \quad +14 \quad +13 \quad +12\]

Output:

\[+1 \quad +2 \quad +3 \quad +4 \quad +5 \quad +6 \quad +7 \quad +8 \quad +9
+2 \quad +25 \quad +25 \quad +25 \quad +26 \quad +27 \quad +28 \quad +30
+3 \quad +19 \quad +18 \quad +17 \quad +16 \quad +15 \quad +14 \quad +13 \quad +12\]

Source text

"CODE" 11030;
"PROCEDURE" ROWQSORT(M, R, LC, UC);
"VALUE" R, LC, UC; "INTEGER" R, LC, UC; "ARRAY" M;
"BEGIN" "INTEGER" P, Q, IX, IZ; "REAL" X, XX, Y, ZZ, Z;

"PROCEDURE" ROWSORT;
"BEGIN" "INTEGER" L, U; L:= LC; U:= UC;

PART: P:= L; Q:= U; X:= M[R, P]; Z:= M[R, Q];
"IF" X > Z "THEN"
"BEGIN" Y:= X; M[R, P]:= X:= Z;
M[R, Q]:= Z:= Y
"END";

"IF" U - L > 1 "THEN"
"BEGIN" XX:= X; IX:= P; ZZ:= Z; IZ:= Q;

LEFT: "FOR" P:= P + 1 "WHILE" P < Q "DO"
"BEGIN" X:= M[R, P];
"IF" X >= XX "THEN" "GOTO" RIGHT
"END";
P:= Q - 1; "GOTO" OUT;
RIGHT:  "FOR" Q:= Q - 1 "WHILE" Q > P "DO"
"BEGIN" Z:= M[R, Q];
  "IF" Z <= ZZ "THEN" "GOTO" DIST
"END";
Q:= P; P:= P - 1; Z:= X; X:= M[R, P];

DIST:  "IF" X > Z "THEN"
"BEGIN" Y:= X; M[R, P]:= X:= Z;
M[R, Q]:= Z:= Y
"END";
  "IF" X > XX "THEN" "BEGIN" XX:= X; IX:= P "END";
  "IF" Z < ZZ "THEN" "BEGIN" ZZ:= Z; IZ:= Q "END";
"GOTO" LEFT;

OUT:    "IF" P > IX "AND" X < XX "THEN"
"BEGIN" MR, P]:= XX; M[R, IX]:= X "END";
  "IF" Q < IZ "AND" Z > ZZ "THEN"
"BEGIN" MR, Q]:= ZZ; M[R, IZ]:= Z "END";
  "IF" U - Q > P - L "THEN"
"BEGIN" LC:= L; UC:= P - 1; L:= Q + 1 "END"
 "ELSE"
"BEGIN" UC:= U; LC:= Q + 1; U:= P - 1 "END";
  "IF" UC > LC "THEN" ROWSORT;
  "IF" U > L "THEN" "GOTO" PART
  "END" U - L > 1
"END" OF ROWSORT;

"IF" UC > LC "THEN" ROWSORT
"END" OF ROWQSORT;
"EOP"
Col Qsort

TITLE:   Col Qsort

AUTHOR:  A.C. IJsselstein

INSTITUTE: Mathematical Centre

RECEIVED: 760701

BRIEF DESCRIPTION
The procedure sorts elements of (a segment of) a matrix column, stored in a
two-dimensional array.

KEYWORDS
Sorting of elements of a matrix column

CALLING SEQUENCE

Heading
"PROCEDURE" COL QSORT (M, C, LR, UR);
"VALUE" C, LR, UR;
"INTEGER" C, LR, UR;
"ARRAY" M;
"CODE" 11040;

Formal parameters
M:        <array identifier>, a two-dimensional array M1:1, L2:U2;
C:        <integer arithmetic expression>, the index of the column of M,
in which the sorting has to be performed. C should satisfy the
condition L2 =< C < U2;
LR, UR:   <integer arithmetic expression>, smallest and largest index of
the segment of column C which has to be sorted. LR and UR
should satisfy the condition L1 = LR < UR < U1.

DATA AND RESULTS
After a call of COL QSORT (M, C, LR, UR), M1:L2, C1 up to and including
M:U2, C1 are rearranged into non-decreasing order. All other elements of M
remain unaltered.

PROcedures used
None

LANGUAGE
Algol 60
3.1.3 Col Qsort

METHOD AND PERFORMANCE
The sorting algorithm described in van Emden (1970), (see references section 3.1.), is used on successive elements of a matrix column. The number of operations is of the order \((UR-LR+1) \times \ln(UR-LR+1)\). No auxiliary arrays are declared. The recursion depth is at most \(\ln(UR-LR+1) / \ln(2)\).

EXAMPLE OF USE

Program:

"BEGIN" "ARRAY" Z[1:9, 1:3];
    INARRAY(60, Z);
    COL QSORT(Z, 2, 2, 9);
    OUTPUT(61, "("9(3+2D28),(/))", Z)
"END"

Input:

+1 +2 +3
+2 +27 +19
+3 +26 +18
+4 +30 +17
+5 +28 -16
+6 +25 +15
+7 +27 +14
+8 +25 -13
+9 +25 +12

Output:

+1 +2 +3
+2 +25 +19
+3 +25 +18
+4 +25 +17
+5 +26 -16
+6 +27 +15
+7 +27 +14
+8 +28 -13
+9 +30 +12

SOURCE TEXT

"CODE" 11040;
"PROCEDURE" COLSORT(M, C, LR, UR);
"VALUE" C, LR, UR; "INTEGER" C, LR, UR; "ARRAY" M;
"BEGIN" "INTEGER" P, Q, IX, IZ; "REAL" X, XX, Y, IZ, Z;

"PROCEDURE" COLSORT;
"BEGIN" "INTEGER" L, U; L:= LR; U:= UR;
PART: P:= L; Q:= U; X:= M[P, C]; Z:= M[Q, C];
    "IF" X > Z "THEN"
    "BEGIN" Y:= X; M[P, C]:= X:= Z;
MEQ, CJ:= Z:= Y
"END";
"IF" U - L > 1 "THEN"
"BEGIN" XX:= X; IX:= P; ZZ:= Z; IZ:= Q;
LEFT: "FOR" P:= P + 1 "WHILE" P < Q "DO"
"BEGIN" X:= MEP, CJ;
"IF" X >= XX "THEN" "GOTO" RIGHT
"END";
P:= Q - 1; "GOTO" OUT;
RIGHT: "FOR" Q:= Q - 1 "WHILE" Q > P "DO"
"BEGIN" Z:= MEQ, CJ;
"IF" Z <= ZZ "THEN" "GOTO" DIST
"END";
Q:= P; P:= P - 1; Z:= X; X:= MEP, CJ;
DIST: "IF" X > Z "THEN"
"BEGIN" Y:= X; MEP, CJ:= X:= Z; MEQ, CJ:= Z:= Y
"END";
"IF" X < ZZ "THEN" "BEGIN" XX:= X; IX:= P "END";
"IF" Z < ZZ "THEN" "BEGIN" ZZ:= Z; IZ:= Q "END";
"GOTO" LEFT;
OUT: "IF" P > IX "AND" X < XX "THEN"
"BEGIN" MEP, CJ:= XX; MEX, CJ:= X "END";
"IF" Q < ZZ "AND" Z > ZZ "THEN"
"BEGIN" MEQ, CJ:= ZZ; MEX, CJ:= Z "END";
"IF" U - Q > P - L "THEN"
"BEGIN" LR:= L; UR:= P - 1; L:= Q + 1 "END"
"ELSE"
"BEGIN" UR:= U; LR:= Q + 1; U:= P - 1 "END";
"IF" UR > LR "THEN" COLSORT;
"IF" U > L "THEN" "GOTO" PART
"END" U - L > 1
"END" OF COLSORT;
"IF" UR > LR "THEN" COLSORT
"END" OF COLQSORT;
"EOP"
3.1.4

**Title:**  Rmat Qsort

**Author:**  A.C. IJsselstein

**Institute:**  Mathematical Centre

**Received:**  760701

**Brief Description**
The procedure sorts the (segments of) rows of a matrix, stored in a two-dimensional array, into lexicographical order.

**Keywords**
Sorting of rows of a matrix

**Calling Sequence**

```
PROCEDURE Rmat Qsort (M, LR, UR, LC, UC);
VALUE LR, UR, LC, UC;
INTEGER LR, UR, LC, UC;
ARRAY M;
CODE 11050;
```

**Formal Parameters**

- **M:**  
  <array identifier>, a two-dimensional array M[L1:U1, L2:U2];
- **LR, UR:**  
  <integer arithmetic expression>, smallest and largest index indicating the (segments of) rows of M which have to be sorted. LR and UR should satisfy the condition L1 \leq LR \leq UR \leq U1;
- **LC, UC:**  
  <integer arithmetic expression>, smallest and largest index of the columns involved. LC and UC should satisfy the conditions L2 \leq LC \leq UC \leq U2.

**Data and Results**

After a call of **Rmat Qsort(M, LR, UR, LC, UC)** the (segments of) rows (M[LR, LC],..., M[UR, UC]) up to and including (M[UR, LC],..., M[UR, UC]) are rearranged into lexicographical order. All other elements of M remain unaltered.

**Procedures Used**

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>COL INDSORT</td>
<td>STATAL 11041</td>
</tr>
<tr>
<td>RMAT INDSORT</td>
<td>STATAL 11051</td>
</tr>
<tr>
<td>RMAT PERM</td>
<td>STATAL 11052</td>
</tr>
</tbody>
</table>
Rmat Qsort

Language
Algol 60

Method and performance
By means of Rmat Indqsort, a permutation of the indices LR up to and including UR is obtained which indicates a lexicographical order of the rows involved; using Rmat Perm, this permutation is executed on these rows.

The number of operations and the amount of required memory depend strongly on the contents of the matrix involved. In any case 2 * (UR – LR + 1) + (UR – LC + 1) words are used to declare auxiliary arrays.

Example of use

Program:

"BEGIN" "ARRAY" Z[1:9, 1:5];
  INARRAY(60, Z);
  RMAT QSORT(Z, 2, 9, 2, 5);
  OUTPUT(61, "(""9(5+Z02B),/)", Z);
"END"

Input:
+1 +2 +3 +4 +5
+2 +12 +7 +12 +2
+3 +11 +6 -13 +2
+4 +14 +10 +7 +42
+5 +12 -8 +14 +1
+6 +11 +5 +12 -3
+7 +12 +7 +12 +1
+8 +11 +5 +12 -3
+9 +11 +5 -11 -20

Output:
+1 +2 +3 +4 +5
+2 +11 +5 -11 -20
+3 +11 +5 +12 -3
+4 +11 +5 +12 -3
+5 +11 +6 -13 +2
+6 +12 -8 +14 +1
+7 +12 +7 +12 +1
+8 +12 +7 +12 +2
+9 +14 +10 +7 +42
3.1.4

SOURCE TEXT

"CODE" 11050;
"PROCEDURE" RMATQ SORT(M, LR, UR, LC, UC);
"VALUE" LR, UR, LC, UC; "INTEGER" LR, UR, LC, UC; "ARRAY" M;
"BEGIN" "INTEGER" K; "INTEGER" "ARRAY" IND[L: : UR];

"FOR" K:= LR "STEP" 1 "UNTIL" UR "DO" IND[K]:= K;
RMATINDQ SORT(M, IND, LR, UR, LC, UC);
RMATPERM(IND, LR, UR, LC, UC, M);
"END" RMATQ SORT;
"EOP"
CMAT Qsort

Title: CMAT Qsort

Author: A.C. IJsselstein

Institute: Mathematical Centre

Received: 760701

Brief Description
The procedure sorts the (segments of) columns of a matrix, stored in a two-dimensional array, into lexicographical order.

Keywords
Sorting of columns of a matrix

Calling Sequence

Heading
"PROCEDURE" CMAT QSORT (M, LR, UR, LC, UC);
"VALUE" LR, UR, LC, UC;
"INTEGER" LR, UR, LC, UC;
"ARRAY" M;
"CODE" 11060;

Formal parameters

M: <array identifier>, a two-dimensional array M[1:U1, L2:U2];
LR, UR: <integer arithmetic expression>, smallest and largest index of the rows involved. LR and UR should satisfy the condition L1 ≤ LR ≤ UR ≤ U1;
LC, UC: <integer arithmetic expression>, smallest and largest index indicating the (segments of) columns of M which have to be sorted. LC and UC should satisfy the condition L2 ≤ LC ≤ UC ≤ U2.

Data and Results
After a call of CMAT QSORT (M, LR, UR, LC, UC) the (segments of ) columns (M[LR, LC], ..., M[UR, LC]) up to and including (M[LR, UC], ..., M[UR, UC]) are rearranged into lexicographical order. All other elements of M remain unaltered.

Procedures Used

ROW INDQSOFT STATAL 11031
CMAT INDQSOFT STATAL 11061
CMAT PERM STATAL 11062
3.1.5 Cmat Qsort

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
By means of Cmat Indqsort, a permutation of the indices LC up to and including UC is obtained which indicates a lexicographical order of the columns involved; using Cmat Perm this permutation is executed on these columns. The number of operations and the amount of required memory depend strongly on the contents of the matrix involved. In any case \(2 \times (UC - LC + 1) + (UR - LR + 1)\) words are used to declare auxiliary arrays.

EXAMPLE OF USE

Program:

"BEGIN "ARRAY" Z[1:5, 1:9];
INARRAY(60, Z);
CMAT QSORT(Z, 2, 5, 2, 9);
OUTPUT(61, \"("5(9+2D2B,)//")\", Z)
"END"

Input:

\[
\begin{array}{cccccccc}
+1 & +2 & +3 & +4 & +5 & +6 & +7 & +8 & +9 \\
+2 & +12 & +11 & +14 & +12 & +11 & +12 & +11 & +11 \\
+3 & +7 & +6 & +10 & -8 & +5 & +7 & +5 & +5 \\
+4 & +12 & -13 & +7 & +14 & +12 & +12 & +12 & -11 \\
+5 & +2 & +2 & +42 & +1 & -3 & +1 & -3 & -20 \\
\end{array}
\]

Output:

\[
\begin{array}{cccccccc}
+1 & +2 & +3 & +4 & +5 & +6 & +7 & +8 & +9 \\
+2 & +11 & +11 & +11 & +12 & +12 & +12 & +14 & +14 \\
+3 & +5 & +5 & +5 & +6 & -8 & +7 & +7 & +10 \\
+4 & -11 & +12 & +12 & -13 & +14 & +12 & +12 & +7 \\
+5 & -20 & -3 & -3 & +2 & +1 & +1 & +2 & +42 \\
\end{array}
\]

SOURCE TEXT

"CODE" 11060;
"PROCEDURE" CMATQSORT(M, LR, UR, LC, UC);
"VALUE" LR, UR, LC, UC; "INTEGER" LR, UR, LC, UC; "ARRAY" M;
"BEGIN" "INTEGER" K; "INTEGER" "ARRAY" IND[LC : UC];

"FOR" K:= LC "STEP" 1 "UNTIL" UC "DO" IND[KJ]:= K;
CMATINDQSORT(M, IND, LR, UR, LC, UC);
CMATPERM(IND, LR, UR, LC, UC, M);
"END" CMATQSORT;
"EOQ"

325
Vec2 Qsort

TITLE: Vec2 Qsort

AUTHOR: A.C. IJsselstein

INSTITUTE: Mathematical Centre

RECEIVED: 760701

BRIEF DESCRIPTION
The procedure sorts pairs of numbers, stored in (segments of) two distinct one-dimensional arrays according to the values of the first components.

KEYWORDS
Sorting of pairs of numbers

CALLING SEQUENCE
Heading
"PROCEDURE" Vec2 Qsort (V1, V2, LV, UV);
"VALUE" LV, UV;
"INTEGER" LV, UV;
"ARRAY" V1, V2;
"CODE" 11024;

Formal parameters
V1:   <array identifier>, a one-dimensional array V1[L1:U1];
V2:   <array identifier>, a one-dimensional array V2[L2:U2];
LV, UV: <integer arithmetic expression>, smallest and largest index of the segments of V1 and V2 containing the pairs which have to be sorted. LV and UV should satisfy the conditions L1 ≤ LV ≤ UV ≤ U1 and L2 ≤ LV ≤ UV ≤ U2.

DATA AND RESULTS
After a call of Vec2 Qsort (V1, V2, LV, UV), the elements of V1 and V2 are rearranged simultaneously such that V1[L1]<⋯<V1[U1]. All other elements remain unaltered.

PROCEDURES USED
None

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
The sorting algorithm used is described in van Emden (1970) (see references section 3.1.). Here, simultaneously with every exchange of elements in V1, the corresponding elements in V2 are exchanged. The number of operations is of the order (U1-LV+1) * L1(U1-LV+1). No auxiliary arrays are declared. The
recursion depth is at most $\ln(uv-LV+1) / \ln(2)$.

Example of use

Program:

```
"BEGIN" "ARRAY" X, Y[1:10];
   INARRAY(60, X); INARRAY(60, Y);
   VECCQSORT(X, Y, 4, 8);
   OUTPUT($1, "("2(10(+202B),/))", X, Y)
"END"
```

Input:

```
+22 -13 +99 +45 +7 -1 -13 +7 -22 +5
+1 +2 +3 +4 +5 +6 +7 +8 +9 +10
```

Output:

```
+22 -13 +99 -13 -1 +7 +7 +45 -22 +5
+1 +2 +3 +7 +6 +8 +5 +4 +9 +10
```

Source text

```
"CODE" 11024;
"PROCEDURE" VECCQSORT(V1, V2, LV, UV);
"VALUE" LV, UV; "INTEGER" LV, UV; "ARRAY" V1, V2;
"BEGIN" "INTEGER" P, Q, IX, IZ; "REAL" X, XX, Y, ZZ, Z;

"PROCEDURE" VECCSORT;
"BEGIN" "INTEGER" L, U; L:= LV; U:= UV;

PART: P:= L; Q:= U; X:= V1[P]; Z:= V1[Q];
   "IF" X > Z "THEN"
   "BEGIN" Y:= X; V1[P]:= X:= Z; V1[Q]:= Z:= Y;
       Y:= V2[P]; V2[Z]:= Z:= Y;
   "END";
   "IF" U - L > 1 "THEN"
   "BEGIN" XX:= X; IX:= P; ZZ:= Z; IZ:= Q;

LEFT: "FOR" P:= P + 1 "WHILE" P < Q "DO"
   "BEGIN" X:= V1[P];
       "IF" X >= XX "THEN" "GOTO" RIGHT
   "END";
   P:= Q - 1; "GOTO" OUT;

RIGHT: "FOR" Q:= Q - 1 "WHILE" Q > P "DO"
   "BEGIN" Z:= V1[Q];
       "IF" Z <= ZZ "THEN" "GOTO" DIST
   "END";
   Q:= P; P:= P - 1; Z:= X; X:= V1[P];

DIST: "IF" X > Z "THEN"

```

327
"BEGIN" \( Y := X; \) \( V1[P] := X; \) \( V1[Q] := Z; \) \( Y := V2[P]; \) \( V2[P] := V2[Q]; \) \( V2[Q] := Y \)
"END";
"IF" \( X > XX \) "THEN" "BEGIN" \( X := X; \) \( IX := P \) "END";
"IF" \( Z < ZZ \) "THEN" "BEGIN" \( ZZ := Z; \) \( IZ := Q \) "END";
"GOTO" LEFT;

OUT:
"IF" \( P > IX \) "AND" \( X < XX \) "THEN"
"BEGIN" \( V1[P] := XX; \) \( V1[IX] := X; \)
\( Y := V2[P]; \) \( V2[P] := V2[IX]; \) \( V2[IX] := Y \)
"END";
"IF" \( Q < IZ \) "AND" \( Z < ZZ \) "THEN"
"BEGIN" \( V1[Q] := ZZ; \) \( V1[IIZ] := Z; \)
\( Y := V2[Q]; \) \( V2[Q] := V2[IIZ]; \) \( V2[IIZ] := Y \)
"END";
"IF" \( Q > L \) "THEN"
"BEGIN" \( LV := L; \) \( UV := P - 1; \) \( L := Q + 1 \) "END" "ELSE"
"BEGIN" \( LV := U; \) \( UV := Q + 1; \) \( U := P - 1 \) "END";
"IF" \( UV > LV \) "THEN" VEC2SORT;
"IF" \( U > L \) "THEN" "GOTO" PART
"END" \( U - L > 1 \)
"END" VEC2SORT;

"IF" \( UV > LV \) "THEN" VEC2SORT
"END" VEC2QSORT;
"EOP"
3.2 SORTING VIA INDICES

This section contains five procedures (with suffixes ‘INDQSSORT’), which leave a list of successive items $E(L)$ up to and including $E(U)$ unaltered, but rearrange the indices stored in (a segment of) a one-dimensional integer array $IND$. Beforehand $IND[L]$ up to and including $IND[U]$ are assumed to contain user supplied indices, which are distinct and within the range $L$ up to and including $U$ (not every value within this range needs be present in $IND$). Afterwards $IND[L]$ up to and including $IND[U]$ are permuted such that $E(IND[L]) \leq E(IND[L+1]) \leq \cdots \leq E(IND[U])$.

The prefixes (‘VEC’, ‘ROW’, ‘COL’, ‘RMAT’ or ‘CMAT’) of the procedure identifiers indicate the nature of the items to be processed.

**VEC INDQSSORT** delivers such a permutation for vector elements stored in (a segment of) a one-dimensional array.

**ROW INDQSSORT** delivers such a permutation for elements of one row of a matrix stored in (a segment of) a two-dimensional array.

**COL INDQSSORT** delivers such a permutation for elements of one column of such a matrix.

**RMAT INDQSSORT** delivers such a permutation for the lexicographical order of rows of a matrix, stored in (part of) a two-dimensional array.

**CMAT INDQSSORT** delivers such a permutation for the lexicographical order of columns of such a matrix.
TITLE: Vec Indqsort

AUTHOR: A.C. IJsselstein

INSTITUTE: Mathematical Centre

RECEIVED: 760701

BRIEF DESCRIPTION
The procedure permutes (a segment of) a one-dimensional array of indices in such a way that the values of a given one-dimensional array, ordered according to the permuted indices, are in non-decreasing order.

KEYWORDS
Sorting via indices of vector elements

CALLING SEQUENCE

Heading
"PROCEDURE" Vec Indqsort (V, IND, LVI, UVI);
"VALUE" LVI, UVI;
"INTEGER" LVI, UVI;
"ARRAY" V;
"INTEGER" "ARRAY" IND;
"CODE" 11021;

Formal parameters
V: <array identifier>, a one-dimensional array V[L:U];
IND: <integer array identifier>, a one-dimensional integer array IND[L:U] which contains in the positions LVI up to and including UVI distinct indices within the range (L, U);
LVI, UVI: <integer arithmetic expression>, smallest and largest index of the segment of IND which has to be permuted. LVI and UVI should satisfy the condition L < LVI < UVI < U.

DATA AND RESULTS
Before calling the procedure, IND[L:U] up to and including IND[UVI] should contain user supplied indices which are distinct and within the range L up to and including U (not every value in this range needs to be present in IND).
After a call of Vec Indqsort (V, IND, LVI, UVI), IND[L:U] up to and including IND[UVI] are permuted in such a way that V[IND[L:U]] == V[IND[UVI]]. All other elements of IND and all elements of V remain unaltered.
3.2.1 Vec Indqsort

PROCEDURES USED
None

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
The sorting algorithm described in van Emden (1970) (see references section 3.1.) is used in such a way that the rearrangements are made in the integer array IND and the comparisons in the array V. The number of operations is of the order \( (UVI - LVI + 1) \times LN(UVI - LVI + 1) \). No auxiliary arrays are declared. The recursion depth is at most \( LN(UVI - LVI + 1) / LN(2) \).

EXAMPLE OF USE

Program:
"BEGIN" "INTEGER" I; "ARRAY" X[1:10];
"INTEGER" "ARRAY" P[1:10];
INARRAY(60, X);
"FOR" I:=1 "STEP" 1 "UNTIL" 10 "DO" P[I]:=I;
VEC INDSORT(X, P, 4, 8);
OUTPUT(61, "("2(10(+Id2B))/")", X, P)
"END"

Input:
+22   -13   +99   +45   +7   -1   -13   +7   -22   +5

Output:
+22   -13   +99   +45   +7   -1   -13   +7   -22   +5
+1   +2   +3   +7   +6   +8   +5   +4   +9   +10

SOURCE TEXT
"CODE" 11021;
"PROCEDURE" VECINDQRSORT(V, IND, LVI, UVI);
"VALUE" LVI, UVI; "INTEGER" LVI, UVI; "INTEGER" "ARRAY" IND;
"ARRAY" V;
"BEGIN" "INTEGER" P, Q, JX, JZ, H, N, K;
"REAL" X, XX, Y, ZZ, Z;

"PROCEDURE" VECINDSORT;
"BEGIN" "INTEGER" L, U; L:= LVI; U:= UVI;

PART: P:= L; H:= IND[P]; X:= V[H];
Q:= U; N:= IND[Q]; Z:= V[N];
"IF" X > Z "THEN"
"BEGIN" K:= H; Y:= X;
IND[P]:= H:= N; X:= Z; IND[Q]:= N:= K; Z:= Y

331
"END";

"IF" U - L > 1 "THEN"
"BEGIN" XX := X; JX := P; ZZ := Z; JZ := Q;

LEFT:
"FOR" P := P + 1 "WHILE" P < Q "DO"
"BEGIN" H := IND[P]; X := V[H];
"IF" X >= XX "THEN" "GOTO" RIGHT
"END";
P := Q - 1; "GOTO" OUT;

RIGHT:
"FOR" Q := Q - 1 "WHILE" Q > P "DO"
"BEGIN" N := IND[Q]; Z := VN];
"IF" Z <= ZZ "THEN" "GOTO" DIST
"END";
Q := P; N := H; Z := X; P := P - 1;
H := IND[P]; X := V[H];

DIST:
"IF" X > Z "THEN"
"BEGIN" K := H; Y := X;
"END";
"IF" X >= XX "THEN" "BEGIN" XX := X; JX := P "END"
"IF" Z < ZZ "THEN" "BEGIN" ZZ := Z; JZ := Q "END"
"GOTO" LEFT;

OUT:
"IF" P > JX "AND" X < XX "THEN"
"IF" Q < JZ "AND" Z > ZZ "THEN"
"BEGIN" IND[Q] := IND[Z]; IND[JZ] := N "END";
"IF" U > Q > P - L "THEN"
"BEGIN" LTV := L; UTV := P - 1; L := Q + 1 "END"
"ELSE"
"BEGIN" UTV := U; LTV := Q + 1; U := P - 1 "END"
"IF" UTV > LTV "THEN" VECINDSORT;
"IF" U < L "THEN" "GOTO" PART
"END" U - L > 1
"END" VECINDSORT;

"IF" UTV > LTV "THEN" VECINDSORT
"END" VECINDSORT;
"EOP"
3.2.2

ROW INDSORT

A.C. Ijsselstein

Mathematical Centre

760701

The procedure permutes (a segment of) a one-dimensional array of indices in such a way that the values of (a part of) a specified row of a two-dimensional array, ordered according to the permuted indices, are in non-decreasing order.

Keywords
Sorting via indices of elements of a matrix row

Calling sequence

```
PROCEDURE ROW INDSORT (M, R, IND, LCI, UCI);
```

Formal parameters

\[ M \]: \quad \text{<array identifier>, a two-dimensional array } M[l_1:u_1, l_2:u_2];

\[ R \]: \quad \text{<integer arithmetic expression>, the index of the row of } M \text{ whose non-decreasing order has to be recorded in the array } IND.

\[ \text{IND} \]: \quad \text{<integer array identifier>, a one-dimensional integer array } IND[l_1:u_1] \text{ which contains in the positions } LCI \text{ up to and including } UCI \text{ distinct indices within the range } (L_2, U_2);

\[ LCI,UCI \]: \quad \text{<integer arithmetic expression>, smallest and largest index of the segment of } IND \text{ which has to be permuted. } LCI \text{ and } UCI \text{ should satisfy the condition } L_1 \leq LCI \leq UCI \leq U_1.

Data and results

Before calling the procedure, IND[l_1] up to and including IND[u_1] should contain user supplied indices which are distinct and within the range l to and including u (not every value in this range needs to be present in IND).

After a call of ROW INDSORT (M, R, IND, LCI, UCI), IND[l_1] up to and including IND[u_1] are permuted in such a way that \( M[R, IND[l_1]] \leq \cdots \leq M[R, IND[u_1]] \). All other elements of IND and all elements of M remain unaltered.
Row Indqsort

PROCEDURES USED
None

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
The sorting algorithm described in van Emden (1970) (see references section 3.1.) is used. The rearrangements are made in the array IND and the comparisons in the row R of the array N.

The number of operations is of the order $$(UCI-\text{LCI}+1) \times \ln(UCI-\text{LCI}+1)$$. No auxiliary arrays are declared. The recursion depth is at most $$\ln(UCI-\text{LCI}+1)/\ln(2)$$.

EXAMPLE OF USE

Program:

"BEGIN" "INTEGER" I; "ARRAY" Z[1:3, 1:9];
"INTEGER" "ARRAY" P[1:9];
INARRAY(60, Z);
"FOR" I:=1 "STEP" 1 "UNTIL" 9 "DO" P[I]:=I;
ROW INQDSORT(Z, 2, P, 2, 9);
OUTPUT(61, "("3(9+Z28),/,)9+(Z28)")", Z, P)
"END"

Input:
+1   +2   +3   +4   +5   +6   +7   +8   +9
+2   +27  +26  +30  +28  +25  +27  +25  +25
+3   +19  +18  +17  +16  +15  +14  +13  +12

Output:
+1   +2   +3   +4   +5   +6   +7   +8   +9
+2   +27  +26  +30  +28  +25  +27  +25  +25
+3   +19  +18  +17  +16  +15  +14  +13  +12
+1   +9   +8   +6   +3   +2   +7   +5   +4

334
PROCEDURE ROWNDSORT(M, R, IND, LCI, UCI);
VALUE R, LCI, UCI; INTEGER R, LCI, UCI;
INTEGER "ARRAY" IND; "ARRAY" M;
BEGIN "INTEGER" P, Q, JX, JZ, H, N, K;
REAL X, XX, Y, ZZ, Z;

PROCEDURE ROWNDSORT;
BEGIN "INTEGER" L, U; L := LCI; U := UCI;

PART: P := L; H := IND[P]; X := MCR, HJ;
Q := U; N := IND[Q]; Z := MCR, NJ;
"IF" X > Z "THEN"
"BEGIN" K := H; Y := X;
"END";

"IF" U - L > 1 "THEN"
"BEGIN" XX := X; JX := P; ZZ := Z; JZ := Q;

LEFT: "FOR" P := P + 1 "WHILE" P < Q "DO"
"BEGIN" H := IND[P]; X := MCR, HJ;
"IF" X >= XX "THEN" "GOTO" RIGHT
"END";
P := Q - 1; "GOTO" OUT;

RIGHT: "FOR" Q := Q - 1 "WHILE" Q > P "DO"
"BEGIN" N := IND[Q]; Z := MCR, NJ;
"IF" Z <= ZZ "THEN" "GOTO" DIST
"END";
Q := P; N := H; Z := X;
P := P - 1; H := IND[P]; X := MCR, HJ;

DIST: "IF" X > Z "THEN"
"BEGIN" K := H; Y := X;
"END";
"IF" X > XX "THEN" "BEGIN" XX := X; JX := P "END";
"IF" Z < ZZ "THEN" "BEGIN" ZZ := Z; JZ := Q "END";
"GOTO" LEFT;

OUT: "IF" P > JX "AND" X < XX "THEN"
"BEGIN" IND[P] := IND[X]; IND[X] := H "END";
"IF" Q < JZ "AND" Z > ZZ "THEN"
"BEGIN" IND[Q] := IND[Z]; IND[Z] := N "END";
"IF" U - Q > P - L "THEN"
"BEGIN" LCI := L; UCI := P - 1; L := Q + 1 "END"
"ELSE"
"BEGIN" UCI := U; LCI := Q + 1; U := P - 1 "END";
"IF" UCI > LCI "THEN" ROWNDSORT;
"IF" U > L "THEN" "GOTO" PART
"END" U - L > 1
"END" ROWNDSORT;
"IF" UCI > LCI "THEN" ROWINDSORT
"END" ROWINDSORT;
"EOD"
3.2.3

**Title:** Col Indqsort

**Author:** A.C. IJsselstein

**Institute:** Mathematical Centre

**Received:** 760701

**Brief Description**
The procedure permutes (a segment of) a one-dimensional array of indices in such a way that the values of (a part of) a specified column of a two-dimensional array, ordered according to the permuted indices, are in non-decreasing order.

**Keywords**
Sorting via indices of elements of a matrix column

**Calling Sequence**

*Heading*

"PROCEDURE" COL INDQSORT (M, C, IND, LRI, URI);
"VALUE" C, LRI, URI;
"INTEGER" C, LRI, URI;
"ARRAY" M;
"INTEGER" "ARRAY" IND;
"CODE" 11041;

*Formal parameters*

M: <array indentifier>, a two-dimensional array $M[l1:u1, l2:u2]$;
C: <integer arithmetic expression>, the index of the column of M, whose non-decreasing order has to be recorded in the array IND. C should satisfy the condition $l2 \leq c \leq u2$;
IND: <integer array indentifier>, a one-dimensional integer array IND[l1:u1] which contains in the positions LRI up to and including URI distinct indices within the range (l1, u1);
LRI, URI: <integer arithmetic expression>, smallest and largest index of the segment of IND which has to be permuted. LRI and URI should satisfy the condition $l1 \leq LRI \leq URI \leq u1$.

**Data and Results**
Before calling the procedure, IND[LRI] up to and including IND[URI] should contain user supplied indices which are distinct and within the range L up to and including U (not every value in this range needs to be present in IND). After a call of COL INDQSORT(M, C, IND, LRI, URI), IND[LRI] up to and including IND[URI] are permuted in such a way that $M[IND[LRI], c] \leq \cdots \leq M[IND[URI], c]$. All other elements of IND and all elements of M remain unaltered.
PROCEDURES USED
None

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
The sorting algorithm described in van Emden (1970) (see references section 3.1.) is used. The rearrangements are made in the array IND and the comparisons in the column C of the array N.

The number of operations is of the order \((UFI - LRI + 1) \times LN (UFI - LRI + 1)\). No auxiliary arrays are declared. The recursion depth is at most \(LN (UFI - LRI + 1) / LN(2)\).

EXAMPLE OF USE

Program:

"BEGIN" "INTEGER" I, J; "ARRAY" Z[1:9, 1:3];
   "INTEGER" "ARRAY" P[1:9];
   INARRAY(Z0, Z);
   "FOR" I := 1 "STEP" 1 "UNTIL" 9 "DO" P[I] := I;
   COL INDQSORT(Z, 2, P, 2, 9);
   "FOR" I := 1 "STEP" 1 "UNTIL" 9 "DO"
      "BEGIN" "FOR" J := 1 "STEP" 1 "UNTIL" 5 "DO"
          OUTPUT (61, "," ("1" + ZD2B""), Z[I, J]);
      OUTPUT (61, "(" "5B," + ZD2B, "/")", P[I])
   "END"
"END"

Input:

+1 +2 +3
+2 +27 +19
+3 +26 +18
+4 +30 +17
+5 +28 -16
+6 +25 +15
+7 +27 +14
+8 +25 -13
+9 +25 +12
Output:
+1 +2 +3 +1
+2 +27 +19 +9
+3 +26 +18 +8
+4 +30 +17 +6
+5 +28 -16 +3
+6 +25 +15 +2
+7 +27 +14 +7
+8 +25 -13 +5
+9 +25 +12 +4

Source text
"CODE" 11041;
"PROCEDURE" COLINDQSORT(M, C, IND, LRI, URI);
"VALUE" C, LRI, URI; "INTEGER" C, LRI, URI;
"INTEGER" "ARRAY" IND; "ARRAY" M;
"BEGIN" "INTEGER" P, Q, JX, JZ, H, N, K;
"REAL" X, XX, Y, ZZ, Z;

"PROCEDURE" COLINDQSORT;
"BEGIN" "INTEGER" L, U; L := LRI; U := URI;

PART: P := L; H := IND[P]; X := M[H, C];
\[ Q := U; N := IND[Q]; Z := M[N, C]; "IF" X > Z "THEN"
\[ "BEGIN" K := H; Y := X;
\[ "END";
\[ "IF" U - L > 1 "THEN"
\[ "BEGIN" XX := X; JX := P; ZZ := Z; JZ := Q;

LEFT: "FOR" P := P + 1 "WHILE" P < Q "DO"
\[ "BEGIN" H := IND[P]; X := M[H, C];
\[ "IF" X >= XX "THEN" "GOTO" RIGHT "END";
\[ P := Q - 1; "GOTO" OUT;

RIGHT: "FOR" Q := Q - 1 "WHILE" Q > P "DO"
\[ "BEGIN" N := IND[Q]; Z := M[N, C];
\[ "IF" Z <= ZZ "THEN" "GOTO" DIST "END";
\[ Q := P; N := H; Z := X;
\[ P := P - 1; H := IND[P]; X := M[H, C];

DIST: "IF" X > Z "THEN"
\[ "BEGIN" K := H; Y := X;
\[ "END";
\[ "IF" X > XX "THEN" "BEGIN" XX := X; JX := P "END";
\[ "IF" Z < ZZ "THEN" "BEGIN" ZZ := Y; JZ := Q "END";
\[ "GOTO" LEFT;
OUT:  "IF" P > JX "AND" X < XX "THEN"
       "BEGIN" IND[P]:= IND[JX]; IND[JX]:= H "END";
       "IF" Q < JZ "AND" Z > ZZ "THEN"
       "BEGIN" IND[Q]:= IND[JZ]; IND[JZ]:= N "END";
       "IF" U - Q > P - L "THEN"
       "BEGIN" LRI:= L; URI:= P - 1; L:= Q + 1 "END"
       "ELSE"
       "BEGIN" URI:= U; LRI:= Q + 1; U:= P - 1 "END";
       "IF" URI > LRI "THEN" COLINDSORT;
       "IF" U > L "THEN" "GOTO" PART
       "END" U - L > 1
       "END" COLINDSORT;

       "IF" URI > LRI "THEN" COLINDSORT
       "END" COLINDQSORT;
       "EOP"
3.2.4

**Rmat Indqsort**

**AUTHOR:** A.C. IJsselstein

**INSTITUTE:** Mathematical Centre

**RECEIVED:** 760701

**BRIEF DESCRIPTION**

The procedure permutes (a segment of) a one-dimensional array of indices in such a way that the (segments of) rows of a matrix stored in a two-dimensional array, ordered according to the permuted indices, are in lexicographical order.

**KEYWORDS**

Sorting via indices of rows of a matrix

**CALLING SEQUENCE**

*Heading*

"PROCEDURE" RMAT INDQSORT (M, IND, LRI, URI, LC, UC);
"VALUE" LRI, URI, LC, UC;
"INTEGER" LRI, URI, LC, UC;
"ARRAY" M;
"INTEGER" "ARRAY" IND;
"CODE" 11051;

*Formal parameters*

M: <array identifier>, a two-dimensional array M[L1:U1, L2:U2];
IND: <integer array identifier>, a one-dimensional integer array INDLI:U1J which contains in the positions LRI up to and including URI distinct indices within the range (L1, U1);
LRI, URI: <integer arithmetic expression>, smallest and largest index of the segment of IND which has to be permuted. LRI and URI should satisfy the condition L1 ≤ LRI ≤ URI ≤ U1;
LC, UC: <integer arithmetic expression>, smallest and largest index of the columns involved. LC and UC should satisfy the condition L2 ≤ LC ≤ UC ≤ U2.

**DATA AND RESULTS**

Before calling the procedure, INDLRIJ up to and including IND[URI] should contain user supplied indices which are distinct and within the range L up to and including U (not every value in this range needs to be present in IND). After a call of RMAT INDQSORT (M, IND, LRI, URI, LC, UC), INDLRIJ up to and including IND[URI] are permuted in such a way that they refer to the lexicographical order of the row-segments M[IND[I], LC],...,M[IND[I], UC]) with LRI ≤ I ≤ URI. All other elements of IND and all elements of M remain unaltered.
PROCEDURES USED
COL INDQSORT

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
According to the involved elements of column LC of the matrix, the contents of IND[LRIJ] up to and including IND[URIJ] are rearranged by COL INDQSORT. For any tie, (M[IND[LTI], LC]= ... = M[IND[UTI], LC]), COL INDQSORT rearranges IND[LTI] up to and including IND[UPTI] with regard to those elements of column LC+1 whose row indices belong to the set (IND[LTI], IND[LTI+1], ..., IND[UPTI]). These values in column LC+1 may still have ties and the process is repeated in column LC+2. This continues in subsequent columns C as long as there are ties, unless C exceeds UC.
The number of operations and the amount of required memory depend strongly on the contents of the matrix involved. No auxiliary arrays are declared.

EXAMPLE OF USE

Program:

"BEGIN" "INTEGER" I, J; "ARRAY" Z[1:9, 1:5];
"INTEGER" "ARRAY" P[1:9];
INARRAY(60, Z);
"FOR" I:=1 "STEP" 1 "UNTIL" 9 "DO" P[I]:=I;
RMAT INDQSORT(Z, P, 2, 9, 2, 5);
"FOR" I:=1 "STEP" 1 "UNTIL" 9 "DO"
"BEGIN" "FOR" J:=1 "STEP" 1 "UNTIL" 5 "DO"
OUTPUT(61, "(""ZD2B""", Zu, J));
OUTPUT(61, "(""5B,""ZD2B,""/")", P[I])
"END"
"END"

Input:

\[+1 \ 2 \ 3 \ 4 \ 5 \\
+2 \ +12 \ +7 \ +12 \ +2 \\
+3 \ +11 \ +6 \ -13 \ +2 \\
+4 \ +14 \ +10 \ +7 \ +62 \\
+5 \ +12 \ -8 \ +14 \ +1 \\
+6 \ +11 \ +5 \ +12 \ -3 \\
+7 \ +12 \ +7 \ +12 \ +1 \\
+8 \ +11 \ +5 \ +12 \ -3 \\
+9 \ +11 \ +5 \ -11 \ -20\]
Output:
+1 +2 +3 +4 +5 +1
+2 +12 +7 +12 +2 +9
+3 +11 +6 -13 +2 +6
+4 +14 +10 +7 +42 +8
+5 +12 -8 +14 +1 +3
+6 +11 +5 +12 -3 +5
+7 +12 +7 +12 +1 +7
+8 +11 +5 +12 -3 +2
+9 +11 +5 -11 -20 +4

"CODE" 11051;
"PROCEDURE" RMATINDQSORT(M, IND, LRI, URI, LC, UC);
"VALUE" LRI, URI, LC, UC; "INTEGER" LRI, URI, LC, UC;
"INTEGER" "ARRAY" IND; "ARRAY" M;
"BEGIN" "REAL" X;

"PROCEDURE" PCTI(U);
"VALUE" U; "INTEGER" U;
"BEGIN" "INTEGER" H; "REAL" Y;
COLINDQSORT(M, LC, IND, LRI, U);
X:= M[IND[LRI], LC];
"FOR" H:= LRI + 1 "STEP" 1 "UNTIL" U "DO"
"BEGIN" Y:= M[IND[H], LC]; "IF" X < Y "THEN"
"BEGIN" "IF" LRI < H - 1 "AND" LC < UC
"THEN"
"BEGIN" LC:= LC + 1; PCTI(H-1) "END"
"ELSE" LRI:= H;
X:= Y
"END" X < Y
"END" FOR H;
"IF" LRI < U "AND" LC < UC "THEN"
"BEGIN" LC:= LC + 1; PCTI(U) "END";
LRI:= U + 1; LC:= LC - 1
"END" OF PCTI;

"IF" LRI < URI "AND" LC < UC "THEN" PCTI(URI)
"END" RMATINDQSORT;
"EOP"
Cmat Indqsort

TITLE: Cmat Indqsort

AUTHOR: A.C. IJsselstein

INSTITUTE: Mathematical Centre

RECEIVED: 760701

BRIEF DESCRIPTION
The procedure permutes (a segment of) a one-dimensional array of indices in such a way that the (segments of) columns of a matrix stored in a two-dimensional array, ordered according to the permuted indices, are in lexicographical order.

KEYWORDS
Sorting via indices of columns of a matrix

CALLING SEQUENCE

Heading
"PROCEDURE" CMAT INDOSSORT (M, IND, LR, UR, LCI, UCI);
"VALUE" LR, UR, LCI, UCI;
"INTEGER" LR, UR, LCI, UCI;
"ARRAY" M;
"INTEGER" "ARRAY" IND;
"CODE" 11061;

Formal parameters

M: <array identifier>, a two-dimensional array M(L1:L2, U1:U2);
IND: <integer array identifier>, a one-dimensional integer array IND[L1:L2] which contains in the positions LCI up to and including UCI distinct indices within the range (L2, U2);
LR, UR: <integer arithmetic expression>, smallest and largest index of the rows involved. LR and UR should satisfy the condition L1 < LR < UR < U1;
LCI, UCI: <integer arithmetic expression>, smallest and largest index of the segment of IND which has to be permuted. LCI and UCI should satisfy the condition L1 < LCI < UCI < U1.

DATA AND RESULTS
Before calling the procedure, IND[L1:L2] up to and including IND[UCI] should contain user supplied indices which are distinct and within the range L up to and including U (not every value in this range needs to be present in IND). After a call of CMAT INDOSSORT (M, IND, LR, UR, LCI, UCI), IND[L1:L2] up to and including IND[UCI] are permuted in such a way that they refer to the lexicographical order of the column-segments (MLR, IND[1], ..., MLR, IND[I]) with LCI <= I <= UCI. All other elements of IND and all elements of M remain unaltered.
PROCEDURES USED
ROW INDQ SORT

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
According to the involved elements of row LR of the matrix, the contents of IND[LCL] up to and including IND[UCI] are rearranged by ROW INDQ SORT. For any tie, (MLR, IND[LTI]) = ... = MLR, IND[UT]), ROW INDQ SORT rearranges IND[LTI] up to and including IND[UT] with regard to those elements of row LR+1 whose column indices belong to the set (IND[LTI], IND[LTI+1], ... , IND[UT]), these values in row LR+1 may still have ties and the process is repeated in row LR+2. This continues in subsequent rows R as long as there are ties, unless R exceeds UR.

The number of operations and the amount of required memory depend strongly on the contents of the matrix involved. No auxiliary arrays are declared.

EXAMPLE OF USE

Program:

"BEGIN" "INTEGER" I; "ARRAY" ZZ[1:5, 1:9];
"INTEGER" "ARRAY" P[1:9];
INARRAY(60, Z);
"FOR" I:=1 "STEP" 1 "UNTIL" 9 "DO" P[I]:=I;
CMAT INDQ SORT(Z, P, 2, 5, 2, 9);
OUTPUT(60, "((S(9+Z2B),/),/9(Z2B))", Z, P)
"END"

Input:
+1 +2 +3 +4 +5 +6 +7 +8 +9
+2 +12 +11 +14 +12 +11 +12 +11 +11
+3 +7 +6 +10 -8 +5 +7 +5 +5
+4 +12 -13 +7 +14 +12 +12 +12 -11
+5 +2 +2 +42 +1 -3 +1 -3 -20
Output:
+1  +2  +3  +4  +5  +6  +7  +8  +9
+2  +12 +11 +14 +12 +11 +12 +11 +11
+3  +7  +6  +10 -8  +5  +7  +5  +5
+4  +12 -13  +7  +14  +12  +12 +12 -11
+5  +2  +2  +42  +1  -3  +1  -3 -20
+1  +9  +6  +8  +3  +5  +7  +2  +4

CODE 11061:
PROCEDURE CMATINQDSORT(M, IND, LR, UR, LCI, UCI);
VALUE LR, UR, LCI, UCI; "INTEGER" LR, UR, LCI, UCI;
INTEGER "ARRAY" IND; "ARRAY" M;
BEGIN "REAL" X;

PROCEDURE PRTI(U);
VALUE U; "INTEGER" U;
BEGIN "INTEGER" H; "REAL" Y;
ROWINDEXSORT(M, LR, IND, LCI, U);
X:= M[LCI, IND[LCI]];
FOR H:= LCI + 1 "STEP" 1 "UNTIL" U "DO"
BEGIN Y:= M[LR, IND[H]];
IF X < Y THEN
BEGIN "IF" LCI < H - 1 "AND" LR < UR THEN
BEGIN" LR:= LR + 1;
PRTI(H-1) "END"
ELSE LCI:= H;
END"
X:= Y
END "X < Y"
END FOR H;
IF LCI < U "AND" LR < UR THEN
BEGIN LR:= LR + 1;
PRTI(U) "END"
LCI:= U + 1; LR:= LR - 1
END "OF PRTI"

IF LCI < UCI "AND" LR < UR THEN PRTI( UCI )
END CMATINQDSORT;
"EOP"
3.3 PERMUTING

This section contains five procedures (with suffixes 'PERM') which rearrange a list of successive items \( E(L) \) up to and including \( E(U) \) according to the permutation of indices contained in a one-dimensional integer array \( \text{IND} \) (i.e. the original contents of \( E[\text{IND}[1:1]] \) is assigned to \( E[I] \), for every \( L \leq I \leq U \)).

In the case that those indices are obtained from \( E(L), \ldots, E(U) \) by a call of a procedure of the 'INDGSORT'-type, the result is in non-decreasing (lexicographical) order.

The prefixes ('VEC', 'ROW', 'COL', 'RMAT' or 'CMAT') of the procedure identifiers indicate the nature of the items to be processed.

- **VEC PERM** rearranges vector elements stored in a one-dimensional array.
- **ROW PERM** rearranges elements of (a segment of) a row of a matrix stored in a two-dimensional array.
- **COL PERM** rearranges elements of (a segment of) a column of such a matrix.
- **RMAT PERM** rearranges the rows of a matrix stored in a two-dimensional array.
- **CMAT PERM** rearranges the columns of such a matrix.
**Vec Perm**

**AUTHOR:** A.C. IJsselstein

**INSTITUTE:** Mathematical Centre

**RECEIVED:** 760701

**BRIEF DESCRIPTION**
The procedure rearranges the elements of a vector, stored in a one-dimensional array, according to the permutation of indices contained in (a segment of) a one-dimensional integer array \( \text{IND} \).

**KEYWORDS**
Rearrangement of vector elements via permutation of indices

**CALLING SEQUENCE**

*Heading*

"PROEDURE" Vec Perm (\( \text{IND}, \text{LV}, \text{UV}, \text{V} \));
"VALUE" \( \text{LV}, \text{UV} \);
"INTEGER" \( \text{LV}, \text{UV} \);
"ARRAY" \( \text{V} \);
"INTEGER" "ARRAY" \( \text{IND} \);
"CODE" 11022;

*Formal parameters*

**\( \text{IND} \):** <integer array identifier>, a one-dimensional integer array \( \text{IND}[\text{LI}:\text{UI}] \) which contains in the position \( \text{LV} \) up to and including \( \text{UV} \) the distinct indices within the range \( \text{LV}, \text{UV} \);

**\( \text{LV}, \text{UV} \):** <integer arithmetic expression>, smallest and largest index of the segment of \( \text{V} \) which has to be permuted and of the segment of \( \text{IND} \) which has to be used. \( \text{LV} \) and \( \text{UV} \) should satisfy the conditions \( L \leq \text{LV} \leq \text{UV} \leq U \) and \( \text{LI} \leq \text{LV} \leq \text{UV} \leq \text{UI} \);

**\( \text{V} \):** <array identifier>, a one-dimensional array \( \text{V}[\text{LI}:\text{UI}] \).

**DATA AND RESULTS**
Before calling the procedure, \( \text{IND}[\text{LV}:\text{UI}] \) up to and including \( \text{IND}[\text{UV}:\text{UI}] \) should contain user supplied indices which are distinct and within the range \( \text{LV} \) up to and including \( \text{UV} \). After a call of Vec Perm(\( \text{IND}, \text{LV}, \text{UV}, \text{V} \)), \( \text{V}[\text{LV}:\text{UI}] \) up to and including \( \text{V}[\text{UV}:\text{UI}] \) are rearranged according to the permutation of indices contained in (the segment of) \( \text{IND} \). All other elements of \( \text{V} \) remain unaltered. Vec Perm has no effect on the contents of \( \text{IND} \).
PROCEDURES USED
None

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
A search is made for all cycles that constitute the permutation of indices stored in the array IND. At the discovery of such a cycle all values stored in the array V which belong to that cycle are stored successively in their proper places. The number of operations is of the order \((uv-uv+1)\). One auxiliary boolean array of \((uv-uv+1)\) words is declared.

EXAMPLE OF USE

Program:

"BEGIN" "ARRAY" X[1:10];
"INTEGER" "ARRAY" P[1:10];
INARRAY(60, X); ININTARRAY(60, P);
VEC PERM(P, 4, 8, X);
OUTPUT(61, "("2(10(Z02B),/)")", X, P)
"END"

Input:

+22 -13 +99 +45 +7 -7 -13 +7 -22 +5
+1 +2 +3 +7 +6 +8 +5 +4 +9 +10

Output:

+22 -13 +99 -13 -7 +7 +7 +45 -22 +5
+1 +2 +3 +7 +6 +8 +5 +4 +9 +10

SOURCE TEXT

"CODE" 11022;
"PROCEDURE" VECPERM(IND, LV, UV, V);
"VALUE" LV, UV; "INTEGER" LV, UV; "INTEGER" "ARRAY" IND;
"ARRAY" V;
"BEGIN" "INTEGER" T, J, K; "REAL" X;
"BOOLEAN" "ARRAY" TODO(LV : UV);

"FOR" T := LV "STEP" 1 "UNTIL" UV "DO" TODO[T] := "TRUE";

"FOR" T := LV "STEP" 1 "UNTIL" UV "DO"
"IF" TODO[T] "THEN"
"BEGIN" "COMMENT"
THE BEGINNING OF (ANOTHER) ONE OF THE CYCLES
WHICH CONSTITUTE THE PERMUTATION;
K := T; X := VK[3];
Vec Perm

"FOR" J := IND[K] "WHILE" J ≠ T "DO"
V[K] := X; TODO[K] := "FALSE"
"END" CYCLE

"END" VECPERM;
"EOP"
3.3.2 Row Perm

TITLE: Row Perm

AUTHOR: A.C. IJsselstein

INSTITUTE: Mathematical Centre

RECEIVED: 760701

BRIEF DESCRIPTION
The procedure rearranges the elements of (a segment of) a matrix row, stored in a two-dimensional array, according to the permutation of indices contained in (a segment of) a one-dimensional integer array IND.

KEYWORDS
Rearrangement of matrix row elements via permutation of indices

CALLING SEQUENCE
Heading
"PROCEDURE" ROW PERM (IND, R, LC, UC, M);
"VALUE" R, LC, UC;
"INTEGER" R, LC, UC;
"ARRAY" M;
"INTEGER" "ARRAY" IND;
"CODE" 11032;

Formal parameters
IND: <integer array identifier>, a one-dimensional integer array IND[1:U1] which contains in the positions LC up to and including UC the distinct indices within the range (LC, UC);
R: <integer arithmetic expression>, the index of the row of M which has to be rearranged. R should satisfy the condition L1 < R < U1;
LC, UC: < integer arithmetic expression>, smallest and largest index of the segment of row R which has to be permuted and of the segment of IND which has to be used. LC and UC should satisfy the conditions L2 < LC < UC < U2 and L1 < LC < UC < U1;
M: <array identifier>, a two-dimensional array M[LC1:U1, L2:U2].

DATA AND RESULTS
Before calling the procedure, IND[LC1 up to and including IND[UC1 should contain user supplied indices which are distinct and within the range LC up to and including UC. After a call of ROW PERM (IND, R, LC, UC, M), M[R, LC1 up to and including M[UC] are rearranged according to the permutation of indices contained in (the segment of) IND. All other elements of M remain unaltered. ROW PERM has no effect on the contents of IND.

351
PROCEDURES USED
None

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
A search is made for all cycles that constitute the permutation of indices stored in the array IND. At the discovery of such a cycle all values stored in the matrix row which belong to that cycle are stored successively in their proper places. The number of operations is of the order \((uc - lc + 1)\). One auxiliary boolean array of \((uc - lc + 1)\) words is declared.

EXAMPLE OF USE

Program:

"BEGIN" "ARRAY" Z[1:3, 1:9];
"INTEGER" "ARRAY" P[1:9];
INARRAY(60, Z);
ININTARRAY(60, P);
ROW PERM(P, 2, 2, 9, Z);
OUTPUT(61, "("3(9(+Z02B),/),1,9(+Z02B))", Z, P)
"END"

Input:

+1 +2 +3 +4 +5 +6 +7 +8 +9
+2 +27 +26 +30 +28 +25 +27 +25 +25
+3 +19 +18 +17 +16 +15 +14 +13 +12
+1 +8 +9 +6 +3 +2 +7 +6 +4

Output:

+1 +2 +3 +4 +5 +6 +7 +8 +9
+2 +25 +25 +26 +27 +27 +28 +30
+3 +19 +18 +17 +16 +15 +14 +13 +12
+1 +8 +9 +6 +3 +2 +7 +6 +4
3.3.2 Row Perm

SOURCE TEXT

"CODE" 11032;
"PROCEDURE" ROWPERM(IND, R, LC, UC, M);
"VALUE" R, LC, UC; "INTEGER" R, LC, UC;
"INTEGER" "ARRAY" IND; "ARRAY" M;
"BEGIN" "INTEGER" T, J, K; "REAL" X;
"BOOLEAN" "ARRAY" TODO[LC : UC];

"FOR" T := LC "STEP" 1 "UNTIL" UC "DO" TODO[T] := "TRUE";

"FOR" T := LC "STEP" 1 "UNTIL" UC "DO"
"IF" TODO[T] "THEN"
"BEGIN" "COMMENT"
THE BEGINNING OF (ANOTHER) ONE OF THE CYCLES
WHICH CONSTITUTE THE PERMUTATION;
K := T; X := M[R, K];
"FOR" J := IND[K] "WHILE" J ≠ T "DO"
"BEGIN" M[R, K] := M[R, J];
TODO[K] := "FALSE"; K := J
"END";
M[R, K] := X; TODO[K] := "FALSE"
"END" CYCLE

"END" ROWPERM;
"EOP"
TITLE: Col Perm

AUTHOR: A.C. IJsselstein

INSTITUTE: Mathematical Centre

RECEIVED: 760701

BRIEF DESCRIPTION
The procedure rearranges the elements of (a segment of) a matrix column, stored in a two-dimensional array, according to the permutation of indices contained in (a segment of) a one-dimensional integer array IND.

KEYWORDS
Rearrangement of matrix column elements via permutation of indices

CALLING SEQUENCE

Heading
"PROCEDURE" COL PERM (IND, C, LR, UR, M);
"VALUE" C, LR, UR;
"INTEGER" C, LR, UR;
"ARRAY" M;
"INTEGER" "ARRAY" IND;
"CODE" 11042;

Formal parameters

IND: <integer array identifier>, a one-dimensional integer array IND[LI:UJ] which contains in the positions LR up to and including UR the distinct indices within the range (LR, UR);

C: <integer arithmetic expression>, the index of the column of M which has to be rearranged. C should satisfy the condition L2 ≤ C ≤ U2;

LR, UR: <integer arithmetic expression>, smallest and largest index of the segment of column C which has to be permuted and of the segment of IND which has to be used. LR and UR should satisfy the conditions L1 ≤ LR ≤ UR ≤ U1 and LI ≤ LR ≤ UR ≤ UI;

M: <array identifier>, a two-dimensional array M[LI:U1, L2:U2].

DATA AND RESULTS
Before calling the procedure, IND[LR] up to and including IND[UR] should contain user supplied indices which are distinct and within the range LR up to and including UR. After a call of COL PERM(IND, C, LR, UR, M), M[LR, CJ] up to and including M[UR, C] are rearranged according to the permutation of indices contained in IND. All other elements of M remain unaltered. COL PERM has no effect on the contents of IND.
Procedures used
None

Language
Algol

Method and Performance
A search is made for all cycles that constitute the permutation of indices stored in the array IND. At the discovery of such a cycle all values stored in the matrix column which belong to that cycle are stored successively in their proper places. The number of operations is of the order \((ur-lr+1)\). One auxiliary boolean array of \((ur-lr+1)\) words is declared.

Example of use

Program:

```
"BEGIN" "INTEGER" I, J; "ARRAY" Z[1:9, 1:3];
"INTEGER" "ARRAY" P[1:9];
"FOR" I:= 1 "STEP" 1 "UNTIL" 9 "DO"
"BEGIN" "FOR" J:= 1 "STEP" 1 "UNTIL" 3 "DO"
    INREAL(60, Z[I, J]);
    ININTEGER(60, P[I])
"END";
COL PERM(P, 2, 2, 9, Z);
"FOR" I:= 1 "STEP" 1 "UNTIL" 9 "DO"
"BEGIN" "FOR" J:= 1 "STEP" 1 "UNTIL" 3 "DO"
    OUTPUT(61, "("+ZD2B")", Z[I, J]);
    OUTPUT(61, "("S8+ZD2B,/"), P[I])
"END"
"END"
```

Input:

```
+1 +2 +3 +1
+2 +27 +19 +8
+3 +26 +18 +9
+4 +30 +17 +6
+5 +28 -16 +3
+6 +25 +15 +2
+7 +27 +14 +7
+8 +25 -13 +5
+9 +23 +12 +4
```

355
Output:
+1  +2  +3  +1
+2  +25 +19  +8
+3  +25 +18  +9
+4  +25 +17  +6
+5  +26  +16  +3
+6  +27  +15  +2
+7  +27  +14  +7
+8  +28  -13  +5
+9  +30  +12  +4

Source text:

"CODE" 11042;
"PROCEDURE" COLPERM(IND, C, LR, UR, M);
"VALUE" C, LR, UR; "INTEGER" C, LR, UR;
"INTEGER" "ARRAY" IND; "ARRAY" M;
"BEGIN" "INTEGER" T, J, K; "REAL" X;
"BOOLEAN" "ARRAY" TODO[LR : UR];

"FOR" T:= LR "STEP" 1 "UNTIL" UR "DO" TODO[T]:= "TRUE";

"FOR" T:= LR "STEP" 1 "UNTIL" UR "DO"
"IF" TODO[T] "THEN"
"BEGIN" "COMMENT"
THE BEGINNING OF (ANOTHER) ONE OF THE CYCLES WHICH CONSTITUTE THE PERMUTATION;
K:= T; X:= M[K, C];
"FOR" J:= IND[K] "WHILE" J &lt; T "DO"
"BEGIN" M[K, C]:= M[J, C];
TODO[K]:= "FALSE"; K:= J
"END";
M[K, C]:= X; TODO[K]:= "FALSE"
"END" CYCLE

"END" COLPERM;
"EOP"
3.3.4

**TITLE:**  Rmat Perm

**AUTHOR:**  A.C. IJsselstein

**INSTITUTE:**  Mathematical Centre

**RECEIVED:**  760701

**BRIEF DESCRIPTION**
The procedure rearranges the (segments of) rows of a matrix, stored in a two-dimensional array, according to the permutation of indices contained in (a segment of) a one-dimensional integer array IND.

**KEYWORDS**
Rearrangement of matrix rows via permutation of indices

**CALLING SEQUENCE**

*Heading*

"PROCEDURE" RMAT PERM (IND, LR, UR, LC, UC, M);
"VALUE" LR, UR, LC, UC;
"INTEGER" LR, UR, LC, UC;
"ARRAY" M;
"INTEGER" "ARRAY" IND;
"CODE" 11052;

*Formal parameters*

IND:  <integer array indentifier>, a one-dimensional integer array IND[LI:UJ] which contains in the positions LR up to and including UR the distinct indices within the range (LR, UR);

LR, UR:  <integer arithmetic expression>, smallest and largest index indicating the rows which have to be permuted, and of the segment of IND which has to be used. LR and UR should satisfy the conditions L1 ≤ LR ≤ UR ≤ U1 and L1 ≤ LR ≤ UR ≤ U1;

LC, UC:  <integer arithmetic expression>, smallest and largest index of the columns involved. LC and UC should satisfy the condition L2 ≤ LC ≤ UC ≤ U2;

M:  <array indentifier>, a two-dimensional array M[L1:U1, L2:U2].

**DATA AND RESULTS**

Before calling the procedure, IND[LR] up to and including IND[UR] should contain user supplied indices which are distinct and within the range LR up to and including UR. After a call of RMAT PERM (IND, LR, UR, LC, UC, M), the row-segments M[IND[LR], LC],...,M[IND[LR], UC] up to and including M[IND[UR], LC],...,M[IND[UR], UC] are rearranged according to the permutation of indices contained in IND. All other parts of M remain unaltered. RMAT PERM has no effect on the contents of IND.
PROCEDURES USED
None

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
A search is made for all cycles that constitute the permutation of indices stored in the array IND. At the discovery of such a cycle all matrix rows which belong to that cycle are stored successively in their proper places. The number of operations is of the order (UR-LR+1) * (UC-LC+1). One auxiliary boolean array of (UR-LR+1) words and one auxiliary real array of (UC-LC+1) words are declared.

EXAMPLE OF USE

Program:

"BEGIN" "INTEGER" I, J; "ARRAY" Z[1:9, 1:5];
"INTEGER" "ARRAY" P[1:9];
"FOR" I:= 1 "STEP" 1 "UNTIL" 9 "DO"
"BEGIN" "FOR" J:= 1 "STEP" 1 "UNTIL" 5 "DO"
INREAL(60, Z[I, J]);
ININTEGER(60, P[I])
"END";
RMAT PERM(P, 2, 9, 2, 5, 2);
"FOR" I:= 1 "STEP" 1 "UNTIL" 9 "DO"
"BEGIN" "FOR" J:= 1 "STEP" 1 "UNTIL" 5 "DO"
OUTPUT(61, "("ZD2B")", Z[I, J]);
OUTPUT(61, "("5B,+ZD2B,/)"), P[I])
"END"
"END"

Input:

+1 +2 +3 +4 +5 +1
+2 +12 +7 +12 +2 +9
+3 +11 +6 -13 +2 +6
+4 +14 +10 +7 +42 +8
+5 +12 -8 +14 +1 +3
+6 +11 +5 +12 -3 +5
+7 +12 +7 +12 +1 +7
+8 +11 +5 +12 -3 +2
+9 +11 +5 -11 -20 +4
Output:

+1  +2  +3  +4  +5  +1
+2  +11 +5  -11 -20  +9
+3  +11 +5  +12 -3   +6
+4  +11 +5  +12 -3   +8
+5  +11 +6  -13  +2  +3
+6  +12 -8  +14  +1  +5
+7  +12 +7  +12  +1  +7
+8  +12 +7  +12  +2  +2
+9  +14 +10 +7 +42  +4

Source Text

"CODE" 11052;
"PROCEDURE" RMATPERM(IND, LR, UR, LC, UC, M);
"VALUE" LR, UR, LC, UC; "INTEGER" LR, UR, LC, UC;
"INTEGER" "ARRAY" IND; "ARRAY" M;
"BEGIN" "INTEGER" T, K, H, J;
"REAL" "ARRAY" ROW[LC : UC];
"BOOLEAN" "ARRAY" TODO[LR : UR];

"FOR" T := LR "STEP" 1 "UNTIL" UR "DO" TODO[T] := "TRUE";

"FOR" T := LR "STEP" 1 "UNTIL" UR "DO"
"IF" TODO[T] "THEN"
"BEGIN" "COMMENT"
    THE BEGINNING OF (ANOTHER) ONE OF THE CYCLES
    WHICH CONSTITUTE THE PERMUTATION;
    K := T;
    "FOR" H := LC "STEP" 1 "UNTIL" UC "DO"
       ROW[H] := M[K, H];
    "FOR" J := IND[K] "WHILE" J ≈ T "DO"
    "BEGIN" "FOR" H := LC "STEP" 1 "UNTIL" UC "DO"
       M[K, H] := M[J, H];
       TODO[K] := "FALSE"; K := J
    "END";
    "FOR" H := LC "STEP" 1 "UNTIL" UC "DO"
       M[K, H] := ROW[H];
    TODO[K] := "FALSE"
"END" CYCLE

"END" RMATPERM;
"EOP"
Cmat Perm

3.3.5

TITLE: Cmat Perm

AUTHOR: A.C. IJsselstein

INSTITUTE: Mathematical Centre

RECEIVED: 760701

BRIEF DESCRIPTION
The procedure rearranges the (segments of) columns of a matrix, stored in a
two-dimensional array, according to the permutation of indices contained in (a
segment of) a one-dimensional integer array IND.

KEYWORDS
Rearrangement of matrix columns via permutation of indices

CALLING SEQUENCE
Heading
"PROCEDURE" CMAT PERM (IND, LR, UR, LC, UC, M);
"VALUE" LR, UR, LC, UC;
"INTEGER" LR, UR, LC, UC;
"ARRAY" M;
"INTEGER" "ARRAY" IND;
"CODE" 11062;

Formal parameters
IND: <integer array identifier>, a one-dimensional integer array
IND[LI:U1] which contains in the positions LC up to and including
UC the distinct indices within the range (LC, UC);

LR, UR: <integer arithmetic expression>, smallest and largest index of
the rows involved. LR and UR should satisfy the condition
L1 < LR < UR < U1;

LC, UC: <integer arithmetic expression>, smallest and largest index indica-
ting the columns which have to be permuted, and of the seg-
ment of IND which has to be used. LC and UC should satisfy the
conditions L2 < LC < UC < U2 and L1 < LC < UC < U1;

M: <array identifier>, a two-dimensional array M[LI1:U1, LI2:U2].

DATA AND RESULTS
Before calling the procedure, IND[LC] up to and including IND[UC] should con-
tain user supplied indices which are distinct and within the range LC up to and
including UC. After a call of CMAT PERM (IND, LR, UR, LC, UC, M), the column
(-segment) (MLR, IND[LC2], ..., M[UR, IND[LC2]]) up to and including
(MLR, IND[UC2], ..., M[UR, IND[UC2]]) are rearranged according to the permu-
tation of indices contained in IND. All other parts of M remain unaltered.
CMAT PERM has no effect on the contents of IND.
PROCEDURES USED
None

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
A search is made for all cycles that constitute the permutation of indices stored in the array INO. At the discovery of such a cycle all matrix columns which belong to that cycle are stored successively in their proper places. The number of operations is of the order \((UC-\text{LC}+1) \times (UR-LR+1)\). One auxiliary boolean array of \((UC-\text{LC}+1)\) words and one auxiliary real array of \((UR-LR+1)\) words are declared.

EXAMPLE OF USE

Program:

"BEGIN" "ARRAY" Z[1:5, 1:9]; "INTEGER" "ARRAY" P[1:9];
INARRAY(D0, Z); ININTARRAY(D0, P);
CMAT PERM(P, Z, S, I, Z);
OUTPUT(D1, "\("59(+ZD2B)/,/,9(+ZD2B)\")\", Z, P)
"END"

Input:

+1 +2 +3 +4 +5 +6 +7 +8 +9
+2 +12 +11 +14 +12 +11 +12 +11 +11
+3 +7 +6 +10 +8 +5 +7 +5 +5
+4 +12 +13 +7 +14 +12 +12 +12 +11
+5 +2 +2 +42 +1 +3 +1 +3 +20

Output:

+1 +2 +3 +4 +5 +6 +7 +8 +9
+2 +11 +11 +11 +12 +12 +12 +12 +14
+3 +5 +5 +5 +6 +8 +7 +7 +10
+4 -11 +12 +12 +13 +14 +12 +12 +7
+5 -20 -3 -3 +2 +1 +1 +2 +42

+1 +9 +6 +8 +3 +5 +7 +2 +4
"CODE" 11062;
"PROCEDURE" CMATPERM(IND, LR, UR, LC, UC, M);
"VALUE" LR, UR, LC, UC; "INTEGER" LR, UR, LC, UC;
"INTEGER" "ARRAY" IND; "ARRAY" M;
"BEGIN" "INTEGER" T, K, H, J;
"REAL" "ARRAY" COL[LR : UR];
"BOOLEAN" "ARRAY" TODO[LC : UC];
"FOR" T := LC "STEP" 1 "UNTIL" UC "DO" TODO[T] := "TRUE";
"FOR" T := LC "STEP" 1 "UNTIL" UC "DO"
"IF" TODO[T] "THEN"
"BEGIN" "COMMENT"
THE BEGINNING OF (ANOTHER) ONE OF THE CYCLES
WHICH CONSTITUTE THE PERMUTATION;
K := T;
"FOR" H := LR "STEP" 1 "UNTIL" UR "DO"
COL[H] := M[H, K];
"FOR" J := IND[K] "WHILE" J ~ T "DO"
"BEGIN" "FOR" H := LR "STEP" 1 "UNTIL" UR "DO"
M[H, K] := M[H, J];
TODO[K] := "FALSE"; K := J
"END";
"FOR" H := LR "STEP" 1 "UNTIL" UR "DO"
M[H, K3] := COL[H3];
TODO[K3] := "FALSE"
"END" CYCLE
"END" CMATPERM;
"EOP"
3.4 RANKS AND TIES

This section contains five procedures (with suffixes `RANKTIE`) which leave a list of successive items \( E(L) \) up to and including \( E(U) \) unaltered and deliver in a real array \( RNK \) their (average) ranks, in a 'real' \( TIES2 \) the sum of the squares of the sizes of the ties, and in a 'real' \( TIES3 \) the sum of the cubes of the sizes of these ties. Furthermore, the same permutation of indices as in the type \( 'INDSORT' \) is delivered in an integer array \( IND \) (but in contrast to these the 'RANKTIE' procedures themselves generate the required contents of the array \( IND \)).

The (average) rank of \( E(K) \) is defined as \( RNK[K]=(S(K)+T(K))/2 \), where \( S(K)=1+\{ \text{the number of elements } E(J) \text{ with } E(J)<E(K) \} \), and \( T(K)=\{ \text{the number of elements } E(H) \text{ with } E(H)\leq E(K) \}. \) Consequently an element which is smaller than another element has a lower rank than the other one, and elements belonging to the same tie, i.e. a subset of all elements which are equal to each other, have the same average rank. (Remark: the size of a tie is the number of elements in that subset).

Note that \( 1 \leq RNK[K] \leq U-L+1 \).

In order to avoid problems which may arise from integer capacity of the computer, the variables \( TIES2 \) and \( TIES3 \) are of real-type, although they are actually integers. The prefixes ('VEC', 'ROW', 'COL', 'RMAT' or 'CMAT') of the procedure identifiers indicate the nature of the items to be ranked.

- **VEC RANKTIE** performs these tasks for vector elements stored in a one-dimensional array.
- **ROW RANKTIE** performs these tasks for elements of a row(-segment) of a matrix stored in a two-dimensional array.
- **COL RANKTIE** performs these tasks for elements of a column(-segment) of such a matrix.
- **RMAT RANKTIE** performs these tasks for rows of a matrix, stored in a two-dimensional array, according to the lexicographical order of these rows.
- **CMAT RANKTIE** performs these tasks for columns of a matrix, stored in a two-dimensional array, according to the lexicographical order of these columns.
TITLE: Vec Ranktie

AUTHOR: A.C. IJsselstein

INSTITUTE: mathematical Centre

RECEIVED: 760701

BRIEF DESCRIPTION
The procedure computes the ranks of vector elements, stored in a one-
dimensional array, and delivers also the sum of the squares of the sizes of the
ties, the sum of the cubes of these sizes, and the permutation of indices which
indicates a non-decreasing order of the vector elements.

KEYWORDS
Ranks of vector elements.

CALLING SEQUENCE
Heading
"PROCEDURE" VEC RANKTIE (V, LV, UV, IND, RNK, TIES2, TIES3);
"VALUE" LV, UV;
"INTEGER" LV, UV;
"REAL" TIES2, TIES3;
"INTEGER" "ARRAY" IND;
"REAL" "ARRAY" V, RNK;
"CODE" 11023;
Formal parameters
V: <array indentifier>, a one-dimensional array V[L:U];
LV, UV: <integer arithmetic expression>, smallest and largest index of
the segment of V which has to be ranked. LV and UV should
satisfy the conditions LI ≤ LV ≤ UV ≤ UI and L ≤ LV ≤ UV ≤ U;
IND: <integer array indentifier>, output parameter, a one-
dimensional integer array IND[L:U] which at exit contains the
permutation of indices which indicates a non-decreasing order in
the positions LV up to and including UV;
RNK: <array indentifier>, output parameter, a one-dimensional real
array RNK[L:U] with RL ≤ LV ≤ UV ≤ RU, which at exit contains
the ranks of the vector elements in the positions LV up to and
including UV;
TIES2: <real variable>, output parameter, which at exit contains the
value of the sum of the squares of the sizes of the ties;
TIES3: <real variable>, output parameter, which at exit contains the
value of the sum of the cubes of the sizes of the ties.
3.4.1 Vec Ranktie

Data and results
After a call of vec ranktie (v, lv, uv, ina, rs, t2, t3), rsci has as value the (average) rank of v[i], i=lv,...,uv.

T2 and T3 (respectively) contain the sums of the squares and of the cubes of the sizes of the ties, and IND[v] up to and including IND[UV] are such that v[IN[v]]<...<v[IND[UV]]. All other elements of IND and RS and all elements of V remain unaltered.

Procedures used
vec indqsort                          STATA 11021

Language
Algol 60

Method and performance
The permutation of indices which indicates the non-decreasing order is generated by vec indqsort. Using this information the ranks and the sizes of the ties are computed. Provided that v is a real array, the user is allowed to let vec ranktie overwrite v(lv) up to and including v(uv) by their ranks, i.e. to call vec ranktie (v, lv, uv, ina, v, t2, t3). The number of operations is of the order (uv-lv+1)*((uv-lv+1)). No auxiliary arrays are declared. The recursion depth is at most ln(uv-lv+1)/ln(2).

Example of use

Program:

"BEGIN" "INTEGER" I; "REAL" T2, T3;
"INTEGER" "ARRAY" P[1:10]; "ARRAY" X, RN[1:10];
INARRAY(60, X);
"FOR" I:= 1 "STEP" 1 "UNTIL" 10 "DO"
"BEGIN" P[I]:= 0; RN[I]:= 0 "END";
VEC RANKTIE(X, 4, 8, P, RN, T2, T3);
OUTPUT(61,
"("10(+ZD28),/,10(-ZD.0),/,10(+ZD28),/,2(-5ZD")",
X, RN, P, T2, T3)
"END"
Input:
+22 -13 +99 +45 +7 -1 -13 +7 -22 +5

Output:
+22 -13 +99 +45 +7 -1 -13 +7 -22 +5
0.0 0.0 0.0 5.0 3.5 2.0 1.0 3.5 0.0 0.0
+0 +0 +0 +7 +6 +8 +5 +4 +0 +0
7 11

Source text

"CODE" 11023;
"PROCEDURE" VECRANKTIE(V, LV, UV, IND, RRK, TIES2, TIES3);
"VALUE" LV, UV; "INTEGER" LV, UV; "REAL" TIES2, TIES3;
"INTEGER" "ARRAY" IND; "REAL" "ARRAY" V, RRK;
"BEGIN" "INTEGER" H, J, P, Q, F, F2; "REAL" X, Y, Z;

"FOR" H:= LV "STEP" 1 "UNTIL" UV "DO" IND[H]:= H;
VECINDQSORT(V, IND, LV, UV); TIES2:= TIES3:= 0;
X:= V[IND[LV]]; P:= LV;

"FOR" H:= LV + 1 "STEP" 1 "UNTIL" UV "DO"
"BEGIN" Y:= V[IND[H]]; "IF" X < Y "THEN"
"BEGIN" Q:= H - 1; Z:= (P + Q) / 2 - LV + 1;
"FOR" J:= P "STEP" 1 "UNTIL" Q "DO"
RRK[IND[J]]:= Z;
F:= H - P; F2:= F * F; TIES2:= TIES2 + F2;
TIES3:= TIES3 + F2 * F; X:= Y; P:= H
"END" X < Y
"END" FOR H;
Z:= (P + UV) / 2 - LV + 1;
"FOR" J:= P "STEP" 1 "UNTIL" UV "DO" RRK[IND[J]]:= Z;
F:= UV - P + 1; F2:= F * F; TIES2:= TIES2 + F2;
TIES3:= TIES3 + F2 * F
"END" VECRANKTIE;
"EOP"
3.4.2 Row Ranktie

TITLE:  Row Ranktie

AUTHOR:  A.C. Jsselstein

INSTITUTE:  Mathematical Centre

RECEIVED:  760701

BRIEF DESCRIPTION
The procedure computes the ranks of elements of a matrix row (-segment) stored in a two-dimensional array, and delivers also the sum of the squares of the sizes of the ties, the sum of the cubes of these sizes, and the permutation of indices which indicates a non-decreasing order of the row elements.

KEYWORDS
Ranks of matrix row elements

CALLING SEQUENCE
Heading
"PROCEDURE" ROW RANKTIE (M, R, LC, UC, IND, RNK, TIES2, TIES3);
"VALUE" R, LC, UC;
"INTEGER" R, LC, UC;
"REAL" TIES2, TIES3;
"INTEGER" "ARRAY" IND;
"REAL" "ARRAY" M, RNK;
"CODE" 11033;

Formal parameters
M:  <array identifier>, a two-dimensional array M[L1:U1, L2:U2];
R:  <integer arithmetic expression >, the index of the row of M in which the ranking has to be performed, R should satisfy the condition L1 <= R <= U1;
LC, UC:  <integer arithmetic expression>, smallest and largest index of the segment of row R which has to be ranked. LC and UC should satisfy the conditions L1 <= LC <= UC <= U1 and L2 <= LC <= UC <= U2;
IND:  <integer array identifier>, output parameter, a one-dimensional integer array IND[L1:U1] which at exit contains the permutation of indices which indicates the non-decreasing order in the positions LC up to and including UC;
RNK:  <array identifier>, output parameter, a one-dimensional real array RNK[L1:U1] with RL <= LC <= UC <= RU, which at exit contains the ranks of the row elements in the positions LC up to and including UC;
TIES2:  <real variable>, output parameter, which at exit contains the value of the sum of the squares of the sizes of the ties;
TIES3:  <real variable>, output parameter, which at exit contains the value of the sum of the cubes of the sizes of the ties.
DATA AND RESULTS
After a call of `ROW RANKTIE (M, R, LC, UC, IND, RS, T2, T3)`, `R[N]` has as value the (average) rank of `M[R, N]` for `N = 1, 2, ..., UC`
`T2` and `T3` contain the sums of the squares and of the cubes of the sizes of the ties, and `IND[LC]` up to and including `IND[UC]` are such that `M[R, IND[LC]] < M[R, IND[LC+1]] < ... < M[R, IND[UC]]`. All other elements of `IND` and `RS` and all elements of `M` remain unaltered.

PROCEDURES USED
ROW INDSORT
STATAL 11031

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
The permutation of indices which indicates the non-decreasing order is generated by `ROW INDSORT`. Using this information the ranks and the sizes of the ties are computed. The number of operations is of the order `(UC- LC+1) * (1+LN(UC-LC+1))`. No auxiliary arrays are declared. The recursion depth is at most `LN(UC-LC+1)/LN(2)`.

EXAMPLE OF USE

Program:

```
"BEGIN" "INTEGER" I, J; "REAL" T2, T3;
"INTEGER" "ARRAY" P[1:9]; "ARRAY" Z[1:3, 1:9], RN[1:9];
INARRAY(60, Z);
"FOR" I := 1 "STEP" 1 "UNTIL" 9 "DO"
"BEGIN" P[I] := 0; RN[I] := 0 "END";
ROW RANKTIE(2, Z, 2, 9, P, RN, T2, T3);
OUTPUT(61,
 "(" "9*2D2B", ",", "9*-2D.0", ",", "9*+2D2B", ",", "2*-2D")",
 Z, RN, P, T2, T3)
"END"
```

Input:

```
+1 +2 +3 +4 +5 +6 +7 +8 +9
+2 +27 +26 +30 +28 +25 +27 +25 +25
+3 +19 +18 +17 -16 +15 +14 -13 +12
```

368
3.4.2

Output:
+1 +2 +3 +4 +5 +6 +7 +8 +9
+2 +27 +26 +30 +28 +25 +27 +25 +25
+3 +19 +18 +17 +16 +15 +14 +13 +12

0.0 5.5 4.0 8.0 7.0 2.0 5.5 2.0 2.0
+0 +9 +8 +6 +3 +2 +7 +5 +4
16 38

Source Text

"CODE" 11033;
"PROCEDURE"
    ROWRANKTIE(M, R, LC, UC, IND, RNK, TIES2, TIES3);
"VALUE" R, LC, UC; "INTEGER" R, LC, UC; "REAL" TIES2, TIES3;
"INTEGER" "ARRAY" IND; "REAL" "ARRAY" M, RNK;
"BEGIN" "INTEGER" H, J, P, Q, F, F2; "REAL" X, Y, Z;

    "FOR" H:= LC "STEP" 1 "UNTIL" UC "DO" IND[H]:= H;
    ROWINDEXSORT(M, R, IND, LC, UC); TIES2:= TIES3:= 0;
    X:= M[R, IND[L]]; P:= LC;

    "FOR" H:= LC + 1 "STEP" 1 "UNTIL" UC "DO";
    "BEGIN" Y:= M[R, IND[H]]; "IF" X < Y "THEN"
    "BEGIN" Q:= H - 1; Z:= (P + Q) / (Z - LC + 1;
    "FOR" J:= P "STEP" 1 "UNTIL" Q "DO"
      RNK[IND[J]]:= Z;
    F:= H - P; F2:= F * F; TIES2:= TIES3:= TIES2 + F2;
    TIES3:= TIES3 + F2 * F; X:= Y; P:= H
    "END" X < Y
    "END" "FOR" H;

    Z:= (P + UC) / (Z - LC + 1;
    "FOR" J:= P "STEP" 1 "UNTIL" UC "DO" RNK[INDE[J]]:= Z;
    F:= UC - P + 1; F2:= F * F; TIES2:= TIES3:= TIES2 + F2;
    TIES3:= TIES3 + F2 * F
    "END" "ROWRANKTIE";

"EOP"
Col Ranktie

AUTHOR: A.C. IJsselstein

INSTITUTE: Mathematical Centre

RECEIVED: 760701

BRIEF DESCRIPTION
The procedure computes the ranks of elements of a matrix column (segment) stored in a two-dimensional array, and delivers also the sum of the squares of the sizes of the ties, the sum of the cubes of these sizes, and the permutation of indices which indicates a non-decreasing order of the column elements.

KEYWORDS
Ranks of matrix column elements

CALLING SEQUENCE

Heading
"PROCEDURE" COL RANKTIE (M, C, LR, UR, IND, RNK, TIES2, TIES3);
"VALUE" C, LR, UR;
"INTEGER" C, LR, UR;
"REAL" TIES2, TIES3;
"INTEGER" "ARRAY" IND;
"REAL" "ARRAY" M, RNK;
"CODE" 11043;

Formal parameters

M: <array indentifier>, a two-dimensional array M[1:U1, L2:U2];
C: <integer arithmetic expression>, the index of the column of M in which the ranking has to be performed. C should satisfy the condition \( L2 < C < U2 \);
LR, UR: <integer arithmetic expression>, smallest and largest index of the segment of column C which has to be ranked. LR and UR should satisfy the conditions \( L1 < LR < UR < U1 \) and \( L1 < LR < UR < U1 \);
IND: <integer array indentifier>, output parameter, a one-dimensional integer array IND[1:U1] which at exit contains the permutation of indices which indicates a non-decreasing order in the positions LR up to and including UR;
RNK: <array identifier>, output parameter, a one-dimensional array RNK[LR:RU] with RL < LR < UR < RU, which at exit contains the ranks of the column elements in the positions LR up to and including UR;
TIES2: <real variable>, output parameter, which at exit contains the value of the sum of the squares of the sizes of the ties;
TIES3: <real variable>, output parameter, which at exit contains the value of the sum of the cubes of the sizes of the ties.

370
3.4.3

**DATA AND RESULTS**

After a call of \texttt{COL RANKTIE} (\(M, C, LR, UR, IND, RS, T2, T3\)), \(RS[i] \) has as value the (average) rank of \(MI, CJ, I=LR, \ldots, UR\).

\(T2\) and \(T3\) contain the sums of the squares and of the cubes of the sizes of the ties, and \(IND[LR] \) up to and including \(IND[UR] \) are such that \(M[IND[LR], CJ] \leq M[IND[LR+1], CJ] \leq \cdots \leq M[IND[UR], CJ] \). All other elements of \(IND\) and \(RS\) and all elements of \(M\) remain unaltered.

**PROCEDURES USED**

\texttt{COL INDSORT} \hspace{1cm} \texttt{STATAL 11041}

**LANGUAGE**

Algol 60

**METHOD AND PERFORMANCE**

The permutation of indices which indicates the non-decreasing order is generated via \texttt{COL INDSORT}. Using this information the ranks and the sizes of the ties are computed. The number of operations is of the order \((UR-LR+1) \times \lfloor 1+\ln{(UR-LR+1)} \rfloor \). No auxiliary arrays are declared. The recursion depth is at most \(\ln{(UR-LR+1)}/\ln{2} \).

**EXAMPLE OF USE**

**Program:**

```
"BEGIN" "INTEGER" I, J; "REAL" T2, T3;
 "INTEGER" "ARRAY" P[1:9]; "ARRAY" Z[1:9, 1:33], RN[1:9];
 INARRAY(60, Z);
 "FOR" I:= 1 "STEP" 1 "UNTIL" 9 "DO"
 "BEGIN" P[I] := 0; RN[I] := 0 "END";
 COL RANKTIE(Z, 2, 9, P, RN, T2, T3);
 "FOR" I:= 1 "STEP" 1 "UNTIL" 9 "DO"
 "BEGIN" "FOR" J:= 1 "STEP" 1 "UNTIL" 3 "DO"
 OUTPUT(61, "(" "+ZD2B")", Z[I, J]);
 OUTPUT(61, "(" "SB, -ZD.D")", RN[I]);
 OUTPUT(61, "(" "SB, +ZD2B, /")", P[I])
 "END";
 OUTPUT(61, "(" "-ZD, -ZD")", T2, T3)
 "END"
```

371
Input:
+1  +2  +3  +4  +5  +6  +7  +8  +9
+2  +27 +19 +26 +20 +17 +28 +25 +27 +25
+3  +26 +18 +30 +28 +25 +27 +25 +25 +25
+4  +30 +17 +28 +26 +25 +27 +25 +25 +25

Output:
+1  +2  +3  0.0 +4
+2  +27 +19 5.5 +9
+3  +26 +18 4.0 +8
+4  +30 +17 8.0 +6
+5  +28 +16 7.0 +3
+6  +25 +15 2.0 +2
+7  +27 +14 5.5 +7
+8  +25 +13 2.0 +5
+9  +25 +12 2.0 +4

16  38

Source text:

"CODE" 11043;
"PROCEDURE"
  COLRANKTIE(M, C, LR, UR, IND, Rnk, TIES2, TIES3);
"VALUE" C, LR, UR; "INTEGER" C, LR, UR; "REAL" TIES2, TIES3;
"INTEGER" "ARRAY" IND; "REAL" "ARRAY" M, Rnk;
"BEGIN" "INTEGER" H, J, P, Q, F, F2; "REAL" X, Y, Z;

  "FOR" H:= LR "STEP" 1 "UNTIL" UR "DO" IND[H]:= H;
  COLINDSORT(M, C, IND, LR, UR); TIES2:= TIES3:= 0;
  X:= M[IND[LR]]; CJ; P:= LR;

  "FOR" H:= LR + 1 "STEP" 1 "UNTIL" UR "DO"
  "BEGIN" Y:= M[IND[H], CJ]; "IF" X < Y "THEN"
    "BEGIN" Q:= H - 1; Z:= (P + Q) / Z - LR + 1;
    "FOR" J:= P "STEP" 1 "UNTIL" 4 "DO"
      Rnk[IND[J]]:= Z;
      F:= H - P; F2:= F * F; TIES2:= TIES2 + F2;
      TIES3:= TIES3 + F2 * F; X:= Y; P:= H
    "END" X < Y
  "END" FOR H;

  Z:= (P + UR) / Z - LR + 1;
  "FOR" J:= P "STEP" 1 "UNTIL" UR "DO" Rnk[IND[JJ]]:= Z;
  F:= UR - P + 1; F2:= F * F; TIES2:= TIES2 + F2;
  TIES3:= TIES3 + F2 * F;
"END" COLRANKTIE;
"E00"
3.4.4

**Title:** Rmat Ranktie

**Author:** A.C. Ijsselstein

**Institute:** Mathematical Centre

**Received:** 760701

**Brief Description**
The procedure computes the lexicographical ranks of matrix row (-segment)s stored in a two-dimensional array, and delivers also the sum of the squares of the sizes of the ties, the sum of the cubes of these sizes, and the permutation of indices which indicates a lexicographical order of the row (-segment)s.

**Keywords**
Ranks of matrix rows

**Calling Sequence**

```
PROCEDURE RMAT RANKTIE (M, LR, UR, LC, UC, IND, RNK, TIES2, TIES3);
"VALUE" LR, UR, LC, UC;
"INTEGER" LR, UR, LC, UC;
"REAL" TIES2, TIES3;
"INTEGER" "ARRAY" IND;
"REAL" "ARRAY" M, RNK;
"CODE" 11053;
```

**Formal parameters**

- **M:** <array identifier>, a two-dimensional array M[LI:U1, L2:U2];
- **LR, UR:** <integer arithmetic expression>, smallest and largest index of the rows of M which have to be ranked. LR and UR should satisfy the conditions LI <= LR <= UR <= U1 and L1 <= LR <= UR <= U1;
- **LC, UC:** <integer arithmetic expression>, smallest and largest index of the columns involved. LC and UC should satisfy the condition L2 <= LC <= UC <= U2;
- **IND:** <integer array identifier>, output parameter, a one-dimensional integer array IND[LI:U1] which at exit contains the permutation of indices which indicates the lexicographical order of the rows in the positions LR up to and including UR;
- **RNK:** <array identifier>, output parameter, a one-dimensional array RNK[RL:RU] with RL <= LR <= UR <= RU, which at exit contains the ranks of the rows in the positions LR up to and including UR;
- **TIES2:** <real variable>, output parameter, which at exit contains the value of the sum of the squares of the sizes of the ties;
- **TIES3:** <real variable>, output parameter, which at exit contains the value of the sum of the cubes of the sizes of the ties.
DATA AND RESULTS
After a call of `rmat ranktie (M, LR, UR, LC, UC, IND, RS, T2, T3)`, RS[I] has as value the (average) rank of the row segment (M[I, LC[, . . ., M[I, UC[)), I=LR, . . . , UR.
T2 and T3 contain the sums of the squares and of the cubes of the sizes of the ties, while IND[LR] up to and including IND[UR] contain indices which refer to the lexicographical order of the row segments (M[IND[1], LC[, . . . , M[IND[I], UC[)) with LR <= I <= UR. All other elements of IND and RS and all elements of M remain unaltered.

PROCEDURES USED
`col indqsort` `stata 11041`

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
Except for changes to compute the ranks and the sizes of the ties, the algorithm is the same as in `rmat indqsort` (see section 3.2.4). The number of operations and the amount of required memory depend strongly upon the contents of the matrix involved. No auxiliary arrays are declared.

EXAMPLE OF USE

Program:
```
"BEGIN" "INTEGER" I, J; "REAL" T2, T3;
"INTEGER" "ARRAY" P[1:9]; "ARRAY" Z[1:9], 1:5], RN[1:9];
INARRAY(60, Z);
"FOR" I:= 1 "STEP" 1 "UNTIL" 9 "DO"
"BEGIN" P[I]:= 0; RN[I]:= 0 "END";
RMAT RANKTIE(Z, 2, 9, 2, 5, P, RN, T2, T3);
"FOR" I:= 1 "STEP" 1 "UNTIL" 9 "DO"
"BEGIN" "FOR" J:= 1 "STEP" 1 "UNTIL" 5 "DO"
OUTPUT(61, "(""+ZD2B")", Z[I, J]);
OUTPUT(61, "(""5B, -ZD.0")", RN[I]);
OUTPUT(61, "(""5B, +ZD2B, /")", P[I])
"END";
OUTPUT(61, "(""/", -ZD, -ZD")", T2, T3)
"END"
```
Input:
+1 +2 +3 +4 +5
+2 +12 +7 +12 +2
+3 +11 +6 -13 +2
+4 +14 +10 +7 +42
+5 +12 -8 +14 +1
+6 +11 +5 +12 -3
+7 +12 +7 +12 +1
+8 +11 +5 +12 -3
+9 +11 +5 -11 -20

Output:
+1 +2 +3 +4 +5 0.0 +0
+2 +12 +7 +12 +2 7.0 +9
+3 +11 +6 -13 +2 4.0 +6
+4 +14 +10 +7 +42 8.0 +8
+5 +12 -8 +14 +1 5.0 +3
+6 +11 +5 +12 -3 2.5 +5
+7 +12 +7 +12 +1 6.0 +7
+8 +11 +5 +12 -3 2.5 +2
+9 +11 +5 -11 -20 1.0 +4
10 14

SOURCE TEXT
"CODE" 11053;
"PROCEDURE"
RMATRANKTIE(M, LR, UR, LC, UC, IND, RNK, TIES2, TIES3);
"VALUE" LR, UR, LC, UC; "INTEGER" LR, UR, LC, UC;
"REAL" TIES2, TIES3; "INTEGER" "ARRAY" IND;
"REAL" "ARRAY" M, RNK;
"BEGIN" "INTEGER" J, Q; "REAL" X, Z, S, F, F2;

"PROCEDURE" PCTR( U );
"VALUE" U ; "INTEGER" U ;
"BEGIN" "INTEGER" H ; "REAL" Y ;
COLINDQSORT(M, LC, IND, LR, U);
X := M[ IND[LR], LC ];
"FOR" H := LR + 1 "STEP" 1 "UNTIL" U "DO"
"BEGIN" Y := M[ IND[H], LC ]; "IF" X < Y "THEN"
"BEGIN" Q := H - 1;
"IF" LR = Q "OR" LC = UC "THEN"
"BEGIN" Z := ( LR + Q ) / 2 - S;
"FOR" J := LR "STEP" 1 "UNTIL" Q "DO"
RNK[IND[J]] := Z ;
F := H - LR ; F2 := F * F ;
TIES2 := TIES2 + F2 ;
TIES3 := TIES3 + F2 * F ; LR := H
"END" "ELSE"
"BEGIN" LC := LC + 1 ; PCTR( Q ) "END";
X := Y
"END" X < Y
"END" FOR H ;
"IF" LR = U "OR" LC = UC "THEN"
"BEGIN" Z := ( LR + U ) / 2 - S ;
  "FOR" J := LR "STEP" 1 "UNTIL" U "DO"
    RNK[IND[J]] := Z ;
    F := U - LR + 1 ; F2 := F * F ;
    TIES2 := TIES2 + F2 ;
    TIES3 := TIES3 + F2 * F
  "END" "ELSE"
  "BEGIN" LC := LC + 1 ; PCTR(U) "END" ;
LR := U + 1 ; LC := LC - 1
"END" OF PCTR ;

"IF" LR < UR "AND" LC < UC "THEN"
"BEGIN" S := LR - 1 ; TIES2 := TIES3 := 0 ;
  "FOR" J := LR "STEP" 1 "UNTIL" UR "DO"
    IND[J] := J ;
PCTR(UR)
  "END"
"END" RMATRANKTIE ;
"EOP"
3.4.5

TITLE: Cmat Ranktie

AUTHOR: A.C. IJsselstein

INSTITUTE: Mathematical Centre

RECEIVED: 760701

BRIEF DESCRIPTION
The procedure computes the lexicographical ranks of matrix column (-segment)s stored in a two-dimensional array, and delivers also the sum of the squares of the sizes of the ties, the sum of the cubes of these sizes, and the permutation of indices which indicates the lexicographical order of the column (-segment)s.

KEYWORDS
Ranks of matrix columns

CALLING SEQUENCE
Heading
"PROCEDURE" CMAT RANKTIE (M, LR, UR, LC, UC, IND, RNK, TIES2, TIES3);
"VALUE" LR, UR, LC, UC;
"INTEGER" LR, UR, LC, UC;
"REAL" TIES2, TIES3;
"INTEGER" "ARRAY" IND;
"REAL" "ARRAY" M, RNK;
"CODE" 11063;

Formal parameters
M: <array identifier>, a two-dimensional array M(L1:U1, L2:U2);
LR, UR: <integer arithmetic expression>, smallest and largest index of the row involved. LR and UR should satisfy the condition L1 < LR < UR < U1;
LC, UC: <integer arithmetic expression>, smallest and largest index of the columns of M which have to be ranked. LC and UC should satisfy the conditions L1 < LC < UC < U1 and L2 < LC < UC < U2;
IND: <integer array identifier>, output parameter, a one-dimensional integer array IND(L1:U1) which at exit contains the permutation of indices which indicates the lexicographical order of the columns in the positions LC up to and including UC;
RNK: <array identifier>, output parameter, a one-dimensional real array RNK(L1:U1) with RL < LC < UC < RU, which at exit contains the ranks of the columns in the positions LC up to and including UC;
TIES2: <real variable>, output parameter, which at exit contains the value of the sum of the squares of the sizes of the ties;
TIES3:  <real variable>, output parameter, which at exit contains the value of the sum of the cubes of the sizes of the ties.

DATA AND RESULTS
After a call of CMAT RANKTIE (M, LR, UR, LC, UC, IND, RS, T2, T3), RS[IJ] has as value the (average) rank of the column-segment (MLR, IJ, ..., MUR, IJ), I = LC, ..., UC.
T2 and T3 contain the sums of the squares and of the cubes of the sizes of the ties, while IND[LC] up to and including IND[UC] contain indices which refer to the lexicographical order of the column-segments (MLR, IND[IJ], ..., MUR, IND[IJ]) with LC ≤ I ≤ UC. All other elements of IND and RS and all elements of M remain unaltered.

PROCEDURES USED
ROW INDSORT
STATAL 11031

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
Except for changes to compute the ranks and the sizes of the ties the algorithm is the same as in CMAT INDSORT (see section 3.2.5). The number of operations and the amount of required memory depend strongly upon the contents of the matrix involved.
No auxiliary arrays are declared.

EXAMPLE OF USE

Program:

"BEGIN" "INTEGER" I; "REAL" T2, T3;
 "INTEGER" "ARRAY" P[1:9]; "ARRAY" Z[1:5, 1:9], RN[1:9];
 INARRAY(60, Z);
 "FOR" I:= 1 "STEP" 1 "UNTIL" 5 "DO"
 "BEGIN" P[I]:= 0; RN[I]:= 0 "END";
 CMAT RANKTIE(Z, 2, 5, 2, 9, P, RN, T2, T3);
 OUTPUT(61, "(""S9(Z+ZD2B),/),/9(-ZD.D),/9(+ZD2B),/2(-ZD)""), Z, RN, P, T2, T3)
 "END"
Input:
+1  +2  +3  +4  +5  +6  +7  +8  +9
+2  +12 +11 +14 +12 +11 +12 +11 +11
+3  +7  +6  +10 -8  +5  +7  +5  +5
+4  +12 -13 +7  +14 +12 +12 +12 -11
+5  +2  +2  +42 +1 -3  +1 -3 -20

Output:
+1  +2  +3  +4  +5  +6  +7  +8  +9
+2  +12 +11 +14 +12 +11 +12 +11 +11
+3  +7  +6  +10 -8  +5  +7  +5  +5
+4  +12 -13 +7  +14 +12 +12 +12 -11
+5  +2  +2  +42 +1 -3  +1 -3 -20
0.0  7.0  4.0  8.0  5.0  2.5  6.0  2.5  1.0
+0  +9  +6  +8  +3  +5  +7  +2  +4
10  14

Source text
"CODE" 11063;
"PROCEDURE"
CMATRANKTIE(M, LR, UR, LC, UC, IND, RNK, TIES2, TIES3);
"VALUE" LR, UR, LC, UC; "INTEGER" LR, UR, LC, UC;
"REAL" TIES2, TIES3; "INTEGER" "ARRAY" IND;
"REAL" "ARRAY" M, RNK;
"BEGIN" "INTEGER" J, Q; "REAL" X, Z, S, F, F2 ;

"PROCEDURE" PRTR( U );
"VALUE" U ; "INTEGER" U ;
"BEGIN" "INTEGER" H; "REAL" Y ;
ROWINDQSORT(M, LR, IND, LC, U);
X := M[ LR, IND[LC] ];
"FOR" H := LC + 1 "STEP" 1 "UNTIL" U "DO"
"BEGIN" Y := M[ LR, IND[J] ]; "IF" X < Y "THEN"
"BEGIN" Q := H - 1;
"IF" LC = Q "OR" LR = UR "THEN"
"BEGIN" Z := ( LC + Q ) / 2 - S;
"FOR" J := LC "STEP" 1 "UNTIL" Q "DO"
RNK[IND[J]] := Z;
F := H - LC; F2 := F * F;
TIES2 := TIES2 + F2;
TIES3 := TIES3 + F2 * F; LC := H
"END" "ELSE"
"BEGIN" LR := LR + 1; PRTR( Q ) "END";
X := Y
"END" X < Y
"END" FOR H;
"IF" LC = U "OR" LR = UR "THEN"
"BEGIN" Z := ( LC + U ) / 2 - S;
"FOR" J := LC "STEP" 1 "UNTIL" U "DO"
RNK[IND[J]] := Z;
F := U - LC + 1 ; F2 := F * F ;
TIES2 := TIES2 + F2 ;
TIES3 := TIES3 + F2 * F ;
"END" "ELSE"
"BEGIN" LR := LR + 1 ; PRTR( U ) "END" ;
LC := U + 1 ; LR := LR - 1
"END" OF PRTR ;
"IF" LC < UC "AND" LR < UR "THEN"
"BEGIN" S := LC - 1 ; TIES2 := TIES3 := 0 ;
"FOR" J := LC "STEP" 1 "UNTIL" UC "DO" IND(J) := J ;
PRTR( UC )
"END"
"END" CMATRANKTIE;
"EOP"
4. PERMUTATIONS AND COMBINATIONS

This section contains procedures which generate permutations and combinations of the elements of a set of items. The items are represented by integers which are larger than or equal to unity and smaller than or equal to some \( N \). Several items may be represented by the same integer; say that there are \( f(i) \) items corresponding to the value \( i \); \( i=1, \ldots, N \). Thus the total number of items in the set equals \( M = \sum_{i=1}^{N} f(i) \).

A permutation of the set of items is an arrangement of the items, and it is represented as a sequence of the corresponding \( M \) integers. The total number of distinct arrangements of the \( M \) integers equals

\[
M! \prod_{i=1}^{N} \frac{1}{f(i)!}
\]

Each of these (distinct) arrangements appears \( \prod_{i=1}^{M} f(i)! \) times in the set of all possible arrangements of the \( M \) items.

The procedure \textsc{permutation} assumes that all items in the set are represented by different integers (thus \( f(i)=1 \) for \( i=1, \ldots, N \)), while the procedure \textsc{general perm} allows identical values for items (thus \( f(i) > 0 \), \( i=1, \ldots, N \)). Repeated calls of these procedures generate successively all distinct arrangements of the integers in lexicographical order. Furthermore, these procedures give the number of times each of these (distinct) arrangements appear in the set of all possible arrangements.

For example, let \( N=3 \) and \( f(1)=f(2)=f(3)=1 \), then the six distinct arrangements are:

\[
\begin{align*}
1 & \quad 2 & \quad 3 \\
1 & \quad 3 & \quad 2 \\
2 & \quad 1 & \quad 3 \\
2 & \quad 3 & \quad 1 \\
3 & \quad 1 & \quad 2 \\
3 & \quad 2 & \quad 1
\end{align*}
\]

and each arrangement appears only once.
As a second example consider the case that \( N=3 \), \( f(1)=1 \), \( f(2)=2 \) and \( f(3)=1 \).

The distinct arrangements are:

\[
\begin{array}{cccc}
1 & 2 & 2 & 3 \\
1 & 2 & 3 & 2 \\
1 & 3 & 2 & 2 \\
2 & 1 & 2 & 3 \\
2 & 1 & 3 & 2 \\
2 & 2 & 1 & 3 \\
2 & 2 & 3 & 1 \\
2 & 3 & 1 & 2 \\
3 & 1 & 2 & 2 \\
3 & 2 & 1 & 2 \\
3 & 2 & 2 & 1 \\
\end{array}
\]

Each of these arrangements appears twice in the set of all possible permutations of the four items.

A combination of the set of \( M \) items is a subset of \( R \) items, and it is represented as a subsequence (of size \( R \)) of the sequence of \( M \) integers representing the items. Since the order of the items in the subset is discarded, the subsequence is given in non-decreasing order.

The procedure **COMBINATION** assumes that all items in the set are represented by different integers (thus \( f(i) = 1 \) for \( i=1, \ldots, N \)); while the procedure **GENERAL COMB** allows identical values of items (thus \( f(i) \geq 0 \) for \( i=1, \ldots, N \)). Repeated calls of these procedures generate successively all distinct non-decreasing subsequences of size \( R \) in lexicographical order. Furthermore, these procedures give, for each distinct subsequence, the number of arrangements of \( R \) items which yield the subsequence. This number of arrangements equals

\[
\frac{R! \prod_{i=1}^{M} f(i)}{\prod_{i=1}^{R} g(i)},
\]

where \( g(i) \) is the number of items in the particular subsequence which have value \( i \).

For example, let \( N=5 \), and \( f(1)= \cdots =f(5)=1 \), thus the distinct non-decreasing subsequences of size \( R=3 \) are:
To each subsequence there exist 6 arrangements of three items.
Another example is the case $N=3$, $f(1)=2$, $f(2)=3$, $f(3)=2$ and $R=3$. The different non-decreasing subsequences are (the last column contains the number of arrangements which yield the corresponding subsequence):

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>36</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>72</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>36</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>18</td>
</tr>
</tbody>
</table>

The procedures in this section generate only one permutation or combination which is delivered in a one-dimensional integer array. For each next (in lexicographical order) permutation or combination the procedure must be called again. All procedures have a Boolean variable as parameter. Before the first call of a procedure this variable should have the value "TRUE" to require the first permutation (combination) to be generated. This first permutation (combination) is delivered in a one-dimensional array (as is mentioned above), and the Boolean variable becomes "FALSE". For each next call the array should contain the previous permutation (combination) and the Boolean variable should be "FALSE". The Boolean variable keeps the value "FALSE" until the last (in lexicographical order) permutation (combination) is generated; it then becomes "TRUE".

The procedures GENERAL PERM, GENERAL COMB and COMBINATION deliver an output parameter NUM (an integer variable). It contains the number of arrangements of $M(R)$ items which yield the permutation (combination) concerned. (This value is the same for each permutation and also for each combination when $f(I)=1$ for $I=1,\ldots,N$. Therefore, it is only computed once in GENERAL PERM and in COMBINATION. In GENERAL COMB this value is different for each combination).
Some applications of the procedure are the following:
In order to generate all permutations, discarding the order of items with the same value, use the procedure GENERAL PERM. In the case that the order of items with the same value must be taken into account, take NUM copies of each permutation generated by GENERAL PERM.

In order to generate all combinations of r items, discarding the order of the items (even with different values), use GENERAL COMB. In the case that the order of the items must be taken into account, use GENERAL COMB, apply GENERAL PERM to each combination delivered by GENERAL COMB, and take NUM copies of each permutation.
4.1.1

Permutation

TITLE: Permutation

AUTHOR: J. Bethlehem

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION
The procedure generates a permutation of a sequence of items with values $1, 2, \ldots, n$. The value $i$ appears exactly once ($i=1, 2, \ldots, n$).

KEYWORDS
Permutation of natural numbers

CALLING SEQUENCE

Heading
"PROCEDURE" PERMUTATION (V, N, FIRST);
"VALUE" N;
"INTEGER" "ARRAY" V;
"REAL" N;
"BOOLEAN" FIRST;
"CODE" 40501;

Formal parameters

V: <integer array identifier>, output parameter, one-dimensional integer array V(L:U), where L < 1 and U > N, which at exit contains the result of the permutation;

N: <arithmetic expression>, number of items, largest item value;

FIRST: <boolean variable>, before the first call of PERMUTATION, FIRST should have the value "TRUE". In this call FIRST is set "FALSE" and keeps this value until after successive calls all permutations are generated. The generation of the last permutation sets FIRST to "TRUE" again.

DATA AND RESULTS
The results of a call of the procedure is stored in the array V.
The following error message may appear:
Error number 2 (if N is not an integer $\geq 1$)

PROCEDURES USED

STATA3 ERROR STATA4 40100
Permutation 4.1.1

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
The permutations of the integers 1, 2, ..., N are generated in a lexicographical order, starting with an initial permutation 1, 2, ..., N. Each call of the procedure, except the first one, must have the previous permutation in the array v. The algorithm is that of CACM Alg.202 (see Collected algorithm's of the CACM).

EXAMPLE OF USE

Program:
"BEGIN" "INTEGER" I; "INTEGER" "ARRAY" W[1:3]; "BOOLEAN" F;
F:= "TRUE";
"FOR" I:= 1, I + 1 "WHILE" "NOT" F "DO"
"BEGIN" PERMUTATION(W, 3, F);
"OUTPUT(61, "("3(02B),")")", W)
"END"
"END"

Output:
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

SOURCE TEXT
"CODE" 40501;
"PROCEDURE" PERMUTATION(A, N, FIRST); "VALUE" N;
"INTEGER" "ARRAY" A; "REAL" N; "BOOLEAN" FIRST;
"BEGIN" "INTEGER" I, U, UP, W;
"IF" N < 1 "OR" ENTER(N) < N "THEN"
STATAL3 ERROR("("PERMUTATION")", 2, N);
"IF" FIRST "THEN"
"BEGIN" "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO" A[I]:= I;
FIRST:= "FALSE"; "GOTO" EXIT Proc
"END";
W:= N;

CHECK ORDER:
"BEGIN" W:= W - 1; "GOTO" CHECK ORDER "END";

U:= A[W - 1];
"FOR" I:= N "STEP" -1 "UNTIL" W "DO"
"BEGIN" "IF" A[I] > U "THEN"
"GOTO" NEXT STEP
"END"
"END";

NEXT STEP:
UP:= (N - W - 1) / 2 + 0.1;
"FOR" I := 0 "STEP" 1 "UNTIL" UP "DO"
A[I + 1] := U
"END";

FIRST:= "TRUE";
"FOR" I := 2 "STEP" 1 "UNTIL" N "DO"
"BEGIN" FIRST:= "FALSE"; "GOTO" EXIT PROC "END";

EXIT PROC:
"END" PERMUTATION;
"EOP"
General Perm

AUTHOR: J. Bethlehem

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION
The procedure generates a permutation of a sequence of items with values 1, 2, ..., N. The value I appears F(I) times (I = 1, 2, ..., N). F(I) may be equal to zero.

KEYWORDS
General permutation of natural numbers

CALLING SEQUENCE

Heading
"PROCEDURE" GENERAL PERM (V, M, F, N, NUM, FIRST);
"VALUE" M, N;
"INTEGER ARRAY" V, F;
"REAL" N, M;
"INTEGER" NUM;
"BOOLEAAN" FIRST;
"CODE" 40502;

Formal parameters
V: <integer array identifier>, output parameter, one-dimensional integer array V[L..U], where L < 1 and U > N, which at exit contains the result of the permutation;
M: <arithmetic expression>, number of items, M = \sum F[I] where the sum is over I = 1, ..., N;
F: <integer array identifier>, F[I] is the number of items with value I, I = 1, ..., N;
N: <arithmetic expression>, largest item value;
NUM: <integer variable>, output parameter, which at exit contains the number of arrangements of the items that yield the generated permutation;
FIRST: <boolean variable>, before the first call of GENERAL PERM, FIRST should have the value "TRUE". In this call FIRST is set "FALSE" and keeps this value until after successive calls all permutations are generated. The generation of the last permutation sets FIRST to "TRUE" again.
DATA AND RESULTS
The result of a call of the procedure is stored in the array \( v \).
The following error messages may appear:
Error number 3  (if one of the \( f[i,j] \)’s is not an integer \( \geq 0 \), or \( m \) is not equal to \( \sum f[i,j] \))
Error number 4  (if \( n \) is not an integer \( \geq 1 \))

PROCEDURES USED
statal3  error
statal 40100

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
The distinct permutations of the sequence are generated in a lexicographical order. Each call of the procedure, except the first one, must have the previous permutation in the array \( v \). The algorithm is based on the cacm alg. 202 (see Collected algorithm’s of the cacm).

EXAMPLE OF USE

Program:

```
"BEGIN" "INTEGER" "ARRAY" w[1:4], fw[1:3];
"INTEGER" i, nm; "BOOLEAN" F;
F:="TRUE";
fw[i]:= 1; fw[2]:= 2; fw[3]:= 1;
"FOR" i:= 1, i + 1 "WHILE" "NOT" F "DO"
"BEGIN" GENERAL PERM(w, 4, fw, 3, nm, f);
OUTPUT(61, "(""A(D2B),SZD,"", w, nm)
"END"
"END"
```

Output:

```
1 2 2 3 2
1 2 3 2 2
1 3 2 2 2
2 1 2 3 2
2 1 3 2 2
2 2 1 3 2
2 2 3 1 2
2 3 1 2 2
2 3 2 1 2
3 1 2 2 2
3 2 1 2 2
3 2 2 1 2
```

389
General Perm

Source text:

"CODE" 40502;
"PROCEDURE" GENERAL_PERM(A, M, F, N, NUM, FIRST);
"VALUE" M, N;
"INTEGER" "ARRAY" A, F; "REAL" M, N; "INTEGER" NUM;
"BOOLEAN" FIRST;
"BEGIN" "INTEGER" I, J, TEL, U, UP, W, S; "REAL" FI;

"IF" N < 1 "OR" ENTIER(N) < N "THEN"
STATAL3_ERROR("("GENERAL_PERM")", 4, N);
S := 0; "FOR" I := 1 "STEP" 1 "UNTIL" N "DO"
"BEGIN" FI := FI[I];
"IF" FI < 0 "OR" ENTIER(FI) < FI "THEN"
STATAL3_ERROR("("GENERAL_PERM")", 3, FI);
S := S + FI
"END";
"IF" M = S "THEN"
STATAL3_ERROR("("GENERAL_PERM")", 3, M);

"IF" FIRST "THEN"
"BEGIN" TEL := 0; NUM := 1; FIRST := "FALSE"
"FOR" I := 1 "STEP" 1 "UNTIL" N "DO"
"BEGIN" FI := FI[I];
"FOR" J := 1 "STEP" 1 "UNTIL" FI "DO"
"BEGIN" TEL := TEL + 1;
ALTJ := J; NUM := NUM * J
"END";
"END"; "GOTO" CHECK_LAST
"END";
W := M;

CHECK ORDER:
"IF" W = 1 "THEN" "GOTO" CHECK_LAST "ELSE"
"BEGIN" W := W - 1; "GOTO" CHECK ORDER "END";
U := A[W - 1];
"FOR" I := M "STEP" -1 "UNTIL" W "DO"
"BEGIN" "IF" A[I] > U "THEN"
"GOTO" NEXT_STEP
"END";
"END"

NEXT STEP:
UP := (M - W - 1) / 2 + 0.1;
"FOR" I := 0 "STEP" 1 "UNTIL" UP "DO"
"BEGIN" U := ACM + I;
ACM + I := ACM + I; ACM + I := U
"END";

CHECK LAST:
FIRST := "TRUE";
4.1.2

"FOR" I:= 2 "STEP" 1 "UNTIL" M "DO"
"BEGIN" FIRST:= "FALSE"; "GOTO" EXIT PROC "END";

EXIT PROC:
"END" GENERAL Perm;
"EOP"
Combination

TITLE: Combination

AUTHOR: J. Bethlehem

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION
The procedure generates a subsequence of size $r$ from a sequence of items with values $1, 2, \ldots, n$. The value 1 appears exactly once in the sequence ($i = 1, 2, \ldots, n$). The items of the subsequence are arranged in non-decreasing order.

KEYWORDS
Combination of natural numbers

CALLING SEQUENCE

Heading
"PROCEDURE" COMBINATION (V, R, N, NUM, FIRST);
"VALUE" R, N;
"INTEGER" "ARRAY" V;
"REAL" R, N;
"INTEGER" NUM;
"BOOLEAN" FIRST;
"CODE" 40503;

Formal parameters

V: <integer array identifier>, output parameter, one-dimensional array V[1..u], where $l < 1$ and $u > r$, which at exit contains the result of the combination;

R: <arithmetic expression>, size of the subsequence;

N: <arithmetic expression>, number of items in the sequence;

NUM: <integer variable>, output parameter, which at exit contains the number of arrangements of $r$ items that yield the subsequence;

FIRST: <boolean variable>, before the first call of COMBINATION, FIRST should have the value "TRUE". In this call FIRST is set "FALSE" and keeps this value until after successive calls all subsequences are generated. The generation of the last subsequence sets FIRST to "TRUE" again.

DATA AND RESULTS
The results of a call of the procedure is stored in the array V.
The following error messages may appear:

Error number 2 (if $r$ is not an integer $\geq 1$, or $r > n$)

Error number 3 (if $n$ is not an integer $\geq 1$)

392
PROCEDURES USED
STATAL 3 ERROR

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
The subsequences of size \( r \) are generated in a lexicographical order according to the item values, starting with an initial subsequence with item values 1, 2, ..., \( r \). Each call of the procedure, except the first one, must have the previous subsequence in the array \( v \). The algorithm is that of CACM Alg. 154. (see Collected algorithm's of the CACM).

EXAMPLE OF USE

Program:

```
"BEGIN" "INTEGER" "ARRAY" W[1:3];
"INTEGER" I, NM; "BOOLEAN" F;
F := "TRUE";
"FOR" I := 1, I + 1 "WHILE" "NOT" F "DO"
"BEGIN" COMBINATION(W, 3, 5, NM, F);
OUTPUT(61, "("3(DEL),5ID,/)"", W, NM)
"END"
"END"
```

Output:

```
1 2 3 6
1 2 4 6
1 2 5 6
1 3 4 6
1 3 5 6
1 4 5 6
2 3 4 6
2 3 5 6
2 4 5 6
3 4 5 6
```

SOURCE TEXT

```
"CODE" 40503;
"PROCEDURE" COMBINATION(A, R, N, NUM, FIRST); "VALUE" R, N;
"INTEGER" "ARRAY" A; "REAL" R, N; "INTEGER" NUM;
"BOOLEAN" FIRST;
"BEGIN" "INTEGER" I, J;

"IF" N < 1 "OR" ENTIER(N) < N "THEN"
STATAL 3 ERROR("("COMBINATION")", 3, N) "ELSE"
"IF" R < 1 "OR" R > N "OR" ENTIER(R) < R "THEN"
STATAL 3 ERROR("("COMBINATION")", 2, R);
```

393
"IF" FIRST "THEN"
"BEGIN" FIRST:= "FALSE"; NUM:= 1;
"FOR" I:= 1 "STEP" 1 "UNTIL" R "DO"
    "BEGIN" A[I]:= I; NUM:= NUM * I "END"
"END" "ELSE"
"BEGIN" "FOR" I:= R - 1 "STEP" -1 "UNTIL" 1 "DO"
    "IF" A[I] < N - R + I "THEN"
        "BEGIN" A[I]:= A[I] + 1;
        "FOR" J:= I + 1 "STEP" 1 "UNTIL" R "DO"
        "GOTO" CHECK LAST;
    "END"
"END"

CHECK LAST:
"IF" A[I] = N - R + 1 "THEN" FIRST:= "TRUE"
"END" COMBINATION;
"EOP"
4.2.2

General Comb

AUTHOR: J. Bethlehem

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION
The procedure generates a subsequence of size \( r \) from a sequence of items with values \( 1, 2, \ldots, n \). The value \( i \) appears \( f(i) \) times in the sequence \( i = 1, 2, \ldots, n \). \( f(i) \) may be equal to zero. The items of the subsequence are arranged in non-decreasing order.

KEYWORDS
General combination of natural numbers

CALLING SEQUENCE

Heading
"PROCEDURE" GENERAL Comb (v, r, f, n, num, first);
"VALUE" R, N;
"INTEGER" "ARRAY" V, F;
"REAL" R, N;
"INTEGER" NUM;
"BOOLEAN" FIRST;
"CODE" 40504;

Formal parameters

\( v: \) <integer array identifier>, output parameter, one-dimensional integer array \( v[l..u] \), where \( l < 1 \) and \( u > r \), which at exit contains the result of the combination;

\( r: \) <arithmetic expression>, size of the subsequence;

\( f: \) <integer array identifier>, \( f[i] \) is the number of items with value \( i, i = 1, \ldots, n \);

\( n: \) <arithmetic expression>, largest item value;

\( num: \) <integer variable>, output parameter, which at exit contains the number of arrangements of \( r \) items that yield the subsequence;

\( first: \) <boolean variable>, before the first call of GENERAL Comb, FIRST should have the value "TRUE". In this call FIRST is set "FALSE" and keeps this value until after successive calls all subsequences are generated. The generation of the last subsequence sets FIRST to "TRUE" again.

DATA AND RESULTS
The results of a call of the procedure is stored in the array \( v \).

The following error messages may appear:

Error number 2 (if \( r \) is not an integer \( \geq 1 \), or \( r > \sum f[i] \))

Error number 3 (if one of the \( f[i] \)'s is not an integer \( \geq 0 \))
Error number 4  (if \texttt{N} is not an integer $\geq 1$)

\textbf{PROCEDURES USED}
\begin{verbatim}
STATA3 ERROR

LANGUAGE
Algol 60

\textbf{METHOD AND PERFORMANCE}

The subsequences of size \texttt{r} are generated in a lexicographical order according to the item values. Each call of the procedure, except the first one, must have the previous subsequence in the array \texttt{v}.

\textbf{EXAMPLE OF USE}

\textit{Program:}

\begin{verbatim}
"BEGIN" "INTEGER" "ARRAY" \texttt{w}, \texttt{fw}[1:3];
"INTEGER" \texttt{i}, \texttt{nm}; "BOOLEAN" \texttt{f};
\texttt{f}:="TRUE";
\texttt{fw}[1]:= 2; \texttt{fw}[2]:= 3; \texttt{fw}[3]:= 2;
"FOR" \texttt{i}:= 1, \texttt{i} + 1 "WHILE" "NOT" \texttt{f} "DO"
"BEGIN" GENERAL \texttt{COMB}(\texttt{w}, 3, \texttt{fw}, 3, \texttt{nm}, \texttt{f});
\hspace{1em}OUTPUT(61, "\{\texttt{3(020),5ZD,} /"\}, \texttt{w}, \texttt{nm})
"END"

"END"
\end{verbatim}

\textit{Output:}

\begin{verbatim}
 1 1 2  18
 1 1 3  12
 1 2 2  36
 1 2 3  72
 1 3 3  12
 2 2 2  6
 2 2 3  36
 2 3 3  18
\end{verbatim}

\textbf{SOURCE TEXT}

\begin{verbatim}
"CODE" 40504;
"PROCEDURE" GENERAL \texttt{COMB}(\texttt{a}, \texttt{r}, \texttt{f}, \texttt{N}, \texttt{NUM}, \texttt{FIRST});
"VALUE" \texttt{r}, \texttt{N};
"INTEGER" "ARRAY" \texttt{a}, \texttt{f}; "REAL" \texttt{r}, \texttt{N}; "INTEGER" \texttt{NUM};
"BOOLEAN" \texttt{FIRST};
"BEGIN" "INTEGER" \texttt{i}, \texttt{j}, \texttt{stap}, \texttt{rest}, \texttt{ptr}, \texttt{ind}, \texttt{new}, \texttt{up};
"REAL" \texttt{fi}, \texttt{s}; "INTEGER" "ARRAY" \texttt{fr}[1:N];

  "IF" \texttt{N} < 1 "OR" \texttt{ENTIER}(\texttt{N}) < \texttt{N} "THEN"
STATA3 ERROR("GENERAL \texttt{COMB}"); 4, \texttt{N}):
\hspace{1em}\texttt{S}:= 0; "FOR" \texttt{i}:= 1 "STEP" 1 "UNTIL" \texttt{N} "DO"
\end{verbatim}
BEGIN F[i]:= F[i];
  IF F[i] < 0 "OR" ENTIER(F[i]) < F[i] "THEN"
  STATAL3 ERROR("(GENERAL COMB)"), 3, F[i] "ELSE"
  S:= S + F[i]
END;
IF R < 1 "OR" R > S "OR" ENTIER(R) < R "THEN"
STATAL3 ERROR("(GENERAL COMB)"), 2, R;
PTR:= R; STAP:= REST:= 0;
IF" FIRST "THEN"
BEGIN" FIRST:= "FALSE"; PTR:= 0; NEW:= 1;
  "GOTO" L3
END" ELSE"
BEGIN" "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO" FRI[i]:= 0;
  "FOR" I:= 1 "STEP" 1 "UNTIL" R "DO"
  FRI[i]:= FRI[i] + 1;
END;
L1: PTR:= PTR - STAP; IND:= A[PTR]; STAP:= FRI[IND];
  "IF" IND = N "THEN" "GOTO" L1;
  UP:= "IF" PTR < R "THEN" A[PTR + 1] "ELSE" N;
  "FOR" NEW:= IND + 1 "STEP" 1 "UNTIL" UP "DO"
  "IF" FRNEW < F[NEW] "THEN" "GOTO" L2; "GOTO" L1;
L2: PTR:= REST:= PTR - 1;
L3: STAP:= FRI[NEW]; REST:= REST + STAP;
IF" REST > R "THEN" STAP:= STAP - REST + R;
"FOR" I:= 1 "STEP" 1 "UNTIL" STAP "DO"
BEGIN" PTR:= PTR + 1; A[PTR]=: NEW "END"
NEW:= NEW + 1;
IF" REST < R "THEN" "GOTO" L3;
PTR:= R + 1; FIRST:= "TRUE";
"FOR" I:= N "STEP" -1 "UNTIL" 1 "DO"
BEGIN" FI:= F[i];
  "FOR" J:= 1 "STEP" 1 "UNTIL" FI "DO"
  "BEGIN" PTR:= PTR - 1;
  "IF" PTR < 1 "THEN" "GOTO" L4 "ELSE"
  "IF" A[PTR]=: I "THEN"
BEGIN" FIRST:= "FALSE"; "GOTO" L4 "END"
END" "END";
L4: NUM:= 1;
"FOR" I:= 1 "STEP" 1 "UNTIL" N "DO" FR[I]:= 0;
"FOR" I:= 1 "STEP" 1 "UNTIL" R "DO"
BEGIN" NUM:= NUM + I; FR[i][j]:= FR[i][j] + 1 "END";
"FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
BEGIN" UP:= FI[j]; FI:= FR[I][j]; "IF" FI > 0 "THEN"
BEGIN" "FOR" J:= FI + 1 "STEP" 1 "UNTIL" UP "DO"
  NUM:= NUM + J / (J - FI)
END" "END";
"END" GENERAL COMB;
"EOP"
CWI SYLLABI

MC SYLLABUS