CWI Syllabi

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).

CWI Syllabus 21

STATAL: statistical procedures in Algol 60, part 2

R. van der Horst, R.D. Gill (eds.)

Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

ISBN 90 6196 359 1

NUGI-code: 815

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 `CORMAT`, `MINICORMAT` 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 `PROMOCORCO`, `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.

TITLE: **Wilcoxons W1**

AUTHOR: R. Kaas

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION

For a sample of independent observations `X[LX], ···, X[UX]` 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" WILCOXONS W1 (X, LX, UX, MU, NONZERO, P2, SORTED);
"VALUE" LX, UX, MU, SORTED;
"INTEGER" LX, UX, NONZERO;
"REAL" MU, P2;
"ARRAY" X;
"CODE" 42400;
```

Formal parameters

`X`: <array identifier>, vector containing the sample `X[LX], ···, X[UX]`;

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

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

`MU`: <arithmic 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 `WILCOXONS 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:
Errornumber 2 (if LX ⩾ UX−1)

PROCEDURES USED

VECQSORT	STATAL 11020
WILCOXONS W	STATAL 40000
STATAL3 ERROR	STATAL 40100
PHI	STATAL 41500

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

[1] D. Wabeke & C. van Eeden: *Handleiding voor de toets van Wilcoxon,* Mathematical Centre Report SW 4/70, Mathematical Centre, Amsterdam, 1970.

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.6D")",
        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";
       N:= 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"
```

TITLE: **Kolmogorovs T**

AUTHOR: E. Opperdoes

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION

For a sample of independent observations X[LX],...,X[UX], 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" KOLMOGOROVS 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[LX],...,X[UX];

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 nul-hypothesis distribution funcition;

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 KOLMOGOROVS 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 $\geqslant$ UX−1)

PROCEDURES USED

VECQSORT	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 $ABS(F_E(X[I]) - F_0(X[I]))$, for $I=LX,...,UX$, where F_E is the emperical distribution function of the sample. The approximated right tail probability is computed using a formula in Smirnov (1948).

REFERENCES

[1] W.J. Conover: *Practical nonparametric statistics,* Chapter 6, John Wiley & Sons, 1971.

[2] N. Smirnov: Table for estimating goodness of fit of empirical distributions, *Ann. Math. Stat.* 19, (1948) p.279 - 281.

EXAMPLE OF USE

Program:

```
"BEGIN"
    "REAL" T, PROB, Y;
    "REAL" "ARRAY" SAMPLE[1:10];
    INARRAY(60, SAMPLE);
    T:= KOLMOGOROVS T(SAMPLE, 1, 10, Y, NORMAL(Y, 50, 25),
                                        PROB, "FALSE");
    OUTPUT(61, "("""("KOLMOGOROV'S T   = ")",+5ZD.6D,/,
        "("TAIL PROBABILITY = ")",+5ZD.6D")", T, PROB)
"END"
```

Input:

```
89  48  13  11  75  42  78   5  17  12
```

Output:

```
KOLMOGOROV'S T   =      +0.406582
TAIL PROBABILITY =      +0.073308
```

SOURCE TEXT

```
"CODE" 42405;
"REAL" "PROCEDURE" KOLMOGOROVS 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 * 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" L > U - 2
    "THEN" STATAL3 ERROR ("("KOLMOGOROVS 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); KOLMOGOROVS T:= D;
    P2:= KOLSMYRAS (D * SQRT (NOBS))
"END" KOLMOGOROVS T;
        "EOP"
```

TITLE: **Cramer von Mises W1**

AUTHOR: E. Opperdoes

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION

For a sample of independent observations `X[LX],...,X[UX]`, the procedure computes Cramer von Mises' test statistic `W1` 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 `W1`

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`: <array identifier>, vector containing the sample `X[LX],...,X[UX]`;

`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 nul-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 Cramer von Mises' test statistic `W1` 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:

Errornumber 2 (if LX ⩾ UX−1)

PROCEDURES USED

LIMIT	STATAL LIMIT
VECQSORT	STATAL 11020
STATAL3 ERROR	STATAL 40100
BESS KA01	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

$$\frac{1}{12N}\sum_{I=1}^{N}\{F_0(X(I))-(I-\frac{1}{2})/N\}^2 ,$$

where F_0 is the null-hypothesis distribution function, X(1),...,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

[1] T.W. Anderson & D.A. Darling: Asymptotic theory of certain "goodness-of-fit" criteria based on stochastic processes, *Ann. Math. Stat.* 23, (1952) p.193 - 212.

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 = ")",+5ZD.6D,/,
        "("TAIL PROBABILITY     = ")",+5ZD.6D")", W1, PROB)
"END"
```

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" STATAL3 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[LX],...,X[UX], the procedure computes Anderson Darling's test statistic for testing the hypothesis that the sample is from a normal distribution (with unknown mean and variable).

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:	<array identifier>, vector containing the sample X[LX],...,[UX];
LX:	<integer arithmetic expression>, smallest index of the sample;
UX:	<integer arithmetic expression>, largest index of the sample;
SORTED:	<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:

Errornumber 1	(if all observations in the sample are equal)
Errornumber 3	(if LX ⩾ UX)

PROCEDURES USED

VECQSORT	STATAL 11020
STATAL3 ERROR	STATAL 40100
PHI	STATAL 41500

LANGUAGE
Algol 60

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

$$-(1+4/N+25/N^2)(\sum_{I=1}^{N}(2I-1)(LN(Z(I))+LN(1-Z(N+1-I)))/N-N),$$

where `N=UX-LX+1,Z(I)=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).

significant level	0.15	0.10	0.05	0.025	0.01
critical value	0.576	0.656	0.787	0.918	1.092

REFERENCE

[1] M.A. Stephens: EDF Statistics for goodness of fit and some comparisons, *J. Am. Stat. Assoc.*, 69, p.730-737, (1974).

EXAMPLE OF USE

Program:

```
"BEGIN"
    "ARRAY" X[1:10];
    INARRAY(60, X);
    OUTPUT(61, "(""("ANDERSON-DARLING TEST STATISTIC")",
        4ZD.6D")", 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"
        STATAL3 ERROR("("ANDERSON-DARLING")", 3, U);
    "IF" "NOT" SORTED "THEN" VECQSORT(X, L, U);
    "IF" X[L] = X[U] "THEN"
    STATAL3 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]:= X[I] - 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: **Wilcoxons W2**

AUTHOR: R. Kaas

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION

For two samples of independent observations X[LX],...,X[UX] and Y[LY],...,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,...,X[UX]−SHIFT,Y[LY],...,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], ··· ,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], ··· ,[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 WILCOXONS W2, and the value of the two-sided tail probability to the output parameter P2. The following error messages may appear:

Errornumber 2	(if LX ⩾ UX−1)
Errornumber 5	(if LY ⩾ UY−1)

PROCEDURES USED

WILCOXONS 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

[1] D. Wabeke and C. van Eeden: *Handleiding voor de toets van Wilcoxon,* Mathematical Centre Report, SW 4170, Mathematical Centre, Amsterdam, 1970.

EXAMPLE OF USE

Program:

```
"BEGIN"
    "REAL" W2, PROB;
    "REAL" "ARRAY" SAMPLE1[1:8], SAMPLE2[1:12];
    INARRAY(60, SAMPLE1); INARRAY(60, SAMPLE2);
    W2:= WILCOXONS W2(SAMPLE1, 1, 8, SAMPLE2, 1, 12, 0,
                                    PROB, "FALSE");
    OUTPUT(61, "(""("WILCOXON'S W2    = ")",+5ZD,/,
        "("TAIL PROBABILITY = ")",+5ZD.6D")", W2, PROB)
"END"
```

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, P2, 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"
       X[I]:= X[I] - 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"
```

TITLE: **Ansari Bradleys W**

AUTHOR: R. Kaas

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION

For two samples of independent observations X[LX],...,X[UX] and Y[LY],...,Y[UY], the procedure computes Ansari-Bradley's test statistic W after shifting the first sample by a given distance 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 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],...,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],...,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:

Errornumber 2	(if LX ⩾ UX−1)
Errornumber 5	(if LY ⩾ UY−1)

PROCEDURES USED

VEC 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[LX]−SHIFT,...,X[UX]−SHIFT,Y[LY],...,Y[UY], and N and M are the sizes of the two samples. In case of ties midranks are used. The two-sided tail probability is computed using a normal approximation with continuity correction.

REFERENCE

[1] M. Hollander & D.A. Wolfe, *Nonparametric Statistical Methods*, John Wiley, New York, 1973.

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"
```

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, SIGWAB, 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" STATAL3ERROR("("ANSARI BRADLEY")", 2, XLOW);
    "IF" YLOW>YUPP-2
    "THEN" STATAL3ERROR("("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[I] - 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[I] - 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; NNU:= 1;
        "FOR" J:= J + 1 "STEP" 1 "UNTIL" YUPP "DO"
        "IF" Y[J] = MIN "THEN" NNU:= NNU + 1 "ELSE"
        "BEGIN" D2:= D2 +
                    RANK(TTOT + 1, TTOT + NNU) ** 2 * NNU;
            TTOT:= TTOT + NNU; NNU:= 1; MIN:= Y[J]
        "END";
        D2:= D2 + RANK(TTOT + 1, NT) ** 2 * NNU;
    "END";

    ANSARIBRADLEY:= WAB; "IF" NT // 2 * 2 = NT "THEN"
    "BEGIN" EWAB:= M * (NT + 2) / 4;
        SIGWAB:= N * M / 16 / NT / (NT - 1) *
                 (D2 * 16 - NT * (NT + 2) ** 2)
    "END" "ELSE"
    "BEGIN" EWAB:= M * (NT + 1) ** 2 / NT / 4;
        SIGWAB:= N * M / 16 / NT / NT / (NT - 1) *
                 (D2 * 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"
```

TITLE: **Kendalls Tau**

AUTHOR: R. Kaas

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION

For a sample of paired observations `(X[LXY],Y[LXY]),...,(X[UXY], Y[UXY])`, 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[LXY],...,X[UXY]` all permutations of `Y[LXY],...,Y[UXY]` 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" 42411;
```

Formal parameters

`X`: <array identifier>, vector containing the first components `X[LXY],...,X[UXY]`;

`Y`: <array identifier>, vector containing the second components `Y[LXY],...,Y[UXY]`;

`LXY`: <integer arithmetic expression>, smallest index of the sample;

`UXY`: <integer arithmetic expression>, largest index of the sample;

`KENDALLS S`: <real variable>, output parameter, which at exit contains the value of Kendall's `S`;

`P2`: <real variable>, 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` can not be computed the values `40/9=4.44...4` and 1 are assigned to `KENDALLSTAU` and `P2`.

The following error message may appear:

Errornumber 3 (if LXY ⩾ UXY − 1)

PROCEDURES USED

VECQSORT	STATAL 11020
VEC2QSORT	STATAL 11024
STATAL3 ERROR	STATAL 40100
PHI	STATAL 41500

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
Kendal'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.6D,/,
        "("KENDALL'S S      = ")",+5ZD.6D,/,
        "("TAIL PROBABILITY = ")",+5ZD.6D")",
        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
```

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"
            "BEGIN" L1:= L1 - 1; A [L1]:= A [U2];
                    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"
                "BEGIN" A [L1]:= X2; A [U2]:= X1 "END"
                "ELSE"
                "BEGIN" X:= X1; X1:= X2; X2:= X "END"
            "END";
            "IF" LEFT "THEN"
            "BEGIN" L1:= L1 - 2; U2:= U2 - 2;
                    A [L1]:= X1; A [U2]:= X2
            "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];
    VECQSORT (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[M]; 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[H];
    "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:= Y1[H];
    "FOR" J:= LOW1 + 1 "STEP" 1 "UNTIL" UPP1 "DO"
    "BEGIN" B:= Y1[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 + 1 - H; FF:= F * F; TY2:= TY2 + FF;
    TY3:= TY3 + FF * F;

    F:= UPP - LOW + 1; FF:= F * F;
    DENOM:= (FF - TX2) * (FF - TY2);
    "IF" DENOM > 0 "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: **Spearmans Rho**

AUTHOR: R. Kaas

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION

For a sample of paired observations `(X[LXY],Y[LXY]),...,(X[UXY],Y[UXY])`, 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[LXY],...,X[UXY]` all permutations of `Y[LXY],...,Y[UXY]` 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[LXY],...,X[UXY]`;

`Y`: <array identifier>, vector containing the second components `Y[LXY],...,Y[UXY]`;

`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` can not be computed or if `ABS(RHO)=1`, the values $4*10^{15}$ and 1 are assigned to `SPEARMANS T` and `P2`.

The following error message may appear:

Errornumber 3 (if LXY ⩾ UXY − 1)

PROCEDURES USED

VEC PERM	STATAL 11022
VEC RANKTIE	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*SQRT((UXY−LXY−1)/ (1−RHO²)) 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   = ")",+5ZD.6D,/,
        "("SPEARMAN'S T     = ")",+5ZD.6D,/,
        "("TAIL PROBABILITY = ")",+5ZD.6D")",
        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
```

SOURCE TEXT

```
"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"
STATAL3 ERROR("("SPEARMANS RHO")", 3, LOW)
"ELSE"
"BEGIN" "INTEGER" N, H;
    "REAL" N3, TIES2, TIES3, CTX, CTY, RHO, D;
    "INTEGER" "ARRAY" PERM[LOW : UPP];

    N:= UPP - LOW + 1; N3:= N * N * N;
    VECRANKTIE (Y, LOW, UPP, PERM, Y, TIES2, TIES3);
    CTY:= 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;
        "EOP"
```

TITLE: **Smirnovs D**

AUTHOR: E. Opperdoes

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION

For two samples of independent observations `X[LX],...,X[UX]` and `Y[LY],...,Y[UY]`, the procedure computes Smirnov's test statistic D after shifting the first sample by a given distance `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 `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 indentifier>, vector containing the first sample `X[LX],...,X[UX];`

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

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

`Y`: <array indentifier>, vector containing the second sample `Y[LY],...,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 massages may appear:
Errornumber 2 (if `LX≥UX−1`)
Errornumber 6 (if `LY≥UY−1`)

Procedures used

`VEC QSORT` `STATAL 11020`
`STATAL3 ERROR` `STATAL 40100`

Language

Algol 60

Method and performance

Smirnov's test statistic `D` equals the maximal value of `ABS` ($F_1(Z) - F_2(Z)$), for `Z = X[LX]−SHIFT,...,X[UX] −SHIFT,Y[LY],...,Y[UY]`, 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

[1] W.J. Conover, *Practical nonparametric statistic, chapter 6,* John Wiley & Sons, 1971.
[2] N. Smirnov, Tabel for testing goodness of fit of empirical distributions, *Ann. Math. Stat.,* 19, (1984) P.279-281.

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  58  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);
"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"
    STATAL3 ERROR ("("SMIRNOVS D")", 2, L1);
    "IF" L2 > U2 - 2 "THEN"
    STATAL3 ERROR ("("SMIRNOVS D")", 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" SMIRNOVS D:= T;
        P2:= KOLSMYRAS (T / SQRT (1 / NOBS1 + 1 / NOBS2))
    "END" "ELSE"
    "BEGIN" "REAL" H; H:= ABS (F1 - F2);
        "IF" T < H "THEN" T:= H;
        "GOTO" AGAIN
    "END"
"END" SMIRNOVS D;
        "EOP"
```

TITLE: **Cramer von Mises W2**

AUTHOR: E. Opperdoes

INSTITUTE: Mathematical Centre

RECEIVED: 760901

BRIEF DESCRIPTION

For two samples of independent observations `X[LX],...,X[UX]` and `Y[LY],...,Y[UY]`, the procedure computes Cramer-von Mises test statistic `W2` after shifting the first sample by a given distance `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 `SHIFT`. (This test is two-sided).

KEYWORDS

Cramer-von Mises' two-sample test statistic W2

CALLING SEQUENCE

Heading

```
"REAL" "PROCEDURE" CRAMER VON MISES 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 indentifier>, vector containing the first sample `X[LX],...,X[UX]`;

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

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

`Y`: <array indentifier>, vector containing the second sample; `Y[LY],...,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 Cramer-von Mises' test statistic W2 is assigned to the procedure indentifier CRAMER VON MISES W2, and the approximate value of the right tail probability to the output parameter PR.
The following error messages may appear:

Errornumber 2	(if LX≥UX−1)
Errornumber 6	(if LY≥UY−1)

PROCEDURES USED

LIMIT	STATAL LIMIT
VEC QSORT	STATAL 11020
STATAL3 ERROR	STATAL 40100
BESSKA01	NUMAL 35191

LANGUAGE

Algol 60

METHOD AND PERFORMANCE

Cramer-von Mises' test statistic W2 equals

$$\frac{MN}{(M+N)^2}\sum_{Z}(F_1(Z)-F_2(Z))^2,$$

where the sum is over Z=X[LX]−SHIFT,...,X[UX]−SHIFT, Y[LY],...,Y[UY], 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

[1] W.J. Conover, *Partical nonparametric statistics, chapter 6,* John Wiley & Sons, 1971.

[2] T.W. Anderson & D.A. Darling, Asymptotic theory of certain "goodness-of-fit" criteria based on stochastic processes, *Ann. Math. Stat.* 23, (1952), P. 193-212.

EXAMPLE OF USE

Program:

```
"BEGIN"
    "REAL" W2, PROB;
    "REAL" "ARRAY" SAMPLE1[1:8], SAMPLE2[1:12];
    INARRAY(60, SAMPLE1); INARRAY(60, SAMPLE2);
    W2:= CRAMER VON MISES W2(SAMPLE1, 1, 8, SAMPLE2, 1, 12, 0,
                                    PROB, "FALSE");
    OUTPUT(61, "(""("CRAMER VON MISES' W2 = ")",+5ZD.6D,/,
        "("TAIL PROBABILITY     = ")",+5ZD.6D")", 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"
    STATAL3 ERROR ("("CRAMER VON MISES W2")", 2, L1);
    "IF" L2 > U2 - 2 "THEN"
    STATAL3 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[LX],...,X[UX] 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 indentifier>, vector containing the sample X[LX],...,X[UX];

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 assingned to the procedure indentifier 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 T1 and its two-sided tail probability is straightforward.

EXAMPLE OF USE

Program:

```
"BEGIN"
    "REAL" T1, PROB;
    "REAL" "ARRAY" SAMPLE[1:10];
    INARRAY(60, SAMPLE);
    T1:= STUDENTS T1(SAMPLE, 1, 10, 50, PROB);
    OUTPUT(61, "(""("STUDENT'S T1      = ")",+5ZD.6D,/,
        "("TAIL PROBABILITY = ")",+5ZD.6D")", T1, PROB)
"END"
```

Input:

```
7  74  12  94  79  22  6  51  88  29
```

Output:

```
STUDENT'S T1      =      -0.342054
TAIL PROBABILITY =      +0.740162
```

SOURCE TEXT

```
"CODE" 42402;
"REAL" "PROCEDURE" STUDENTS T1(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("("STUDENTS T1")", 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("("STUDENTS T1")", 0, NOM) "ELSE"
    "BEGIN" STUDENTS T1:= T:=
                    (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[LX],...,X[UX]` and `Y[LY],...,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 `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`: <array indentifier>, vector containing the first sample `X[LX],...,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],...,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 value of the two-sided tail probability.

DATA AND RESULTS

The value of the test statistic is assigned to the procedure indentifier STUDENTS T2, 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 T2)
Errornumber 2	(if LX≥UX−1)
Errornumber 5	(if LX≥UY−1)

PROCEDURES USED

STATAL3 ERROR	STATAL 40100
STUDENT	STATAL 41530

LANGUAGE

Algol 60

METHOD AND PERFORMANCE

The computation of Student's T2 and its two-sided tail probability is straightforward.

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     = ")",+5ZD.6D,/,
        "("TAIL PROBABILITY = ")",+5ZD.6D")", T2, PROB)
"END"
```

Input:

```
34  84  33  62  47  82  87  17
 4  14  22  19  50  58  41  50  64  75  35  27
```

Output:

```
STUDENT'S T2      =       -2.977801
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" STATAL3 ERROR("("STUDENTS T2")", 2, L1);
    "IF" N <= 2
    "THEN" STATAL3 ERROR("("STUDENTS T2")", 5, L2);
    "FOR" I:= L1 "STEP" 1 "UNTIL" U1 "DO"
    "BEGIN" AI:= A1[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"
    STATAL3ERROR("("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[LXY],Y[LXY]),..., (X[UXY],Y[UXY]) the procedure computes the product-moment correlation coëfficient, the corvariance and the two-sided tail probability under the hypothesis of independence and normality of the observations.

KEYWORDS

product-moment correlation coëfficient

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: <array indentifier>, vector containing the first components of the sample X[LXY],...,X[UXY];

Y: <array indentifier>, vector containing the second components of the sample Y[LXY],...,Y[UXY];

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

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

COV: <real variable>, output parameter, which at exit contains the value of the covariance;

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

DATA AND RESULTS

The value of the product-moment correlation coëfficient 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...4 and 1 are assigned to PROMOCORCO and P2.

The following error message may appear:

Errornumber 3 (if LXY≥UXY−1)

PROCEDURES USED

STATAL3 ERROR	(STATAL 40100)
STUDENT	(STATAL 41530)

LANGUAGE

Algol 60

METHOD AND PERFORMANCE

The correlation coëfficient 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      = ")",+5ZD.6D,/,
        "("COVARIANCE       = ")",+5ZD.6D,/,
        "("TAIL PROBABILITY = ")",+5ZD.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" STATAL3 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;
       ALGMESS("("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"
```

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 coëfficient and prints its lower traingular 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 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.

CALLING SEQUENCE LABELS

Heading

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

Formal parameters

VARNR:	<integer array indentifier>, 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")"     "("-2ZV5D")"
       "("7ZB")"     "("-ZD.4D")"
 21    "("B4AB")"     "("-ZV4D")"
       "("5ZB")"     "("B-D.DD")"
```

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

The value 4.4444..... is printed if a correlation coëfficient cannot be computed.

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 culumns of the data matrix. Their correlation is computed as follows:

```
R(X,Y)=(SXY-SX*SY/N/
SQRT((SXX-SX*SX/N)*(SYY-SY*SY/N)),
```

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, 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 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, "("B3D")", VJ);
        J:= 2; INPUT(6, "("3BZD")", 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
```

```
        "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 3BZD AND THE
    OTHER ONES IN FORMAT -ZD. 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, "("5ZB")", "(" -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

          1     2     3     4     5     6     7     8

    1    10000
    2     8361 10000
    3     6632  2394 10000
    4     4888  1126  6750 10000
    5     6631  3487  8324  5429 10000
    6     1813 -0800  4068  5545  2612 10000
    7     8102  6111  8328  6383  7249  4323 10000
    8     2202  0946  3817 -1149  2137  5133  3482 10000

          1     2     3     4     5     6     7     8
```

SOURCE TEXT

```
"CODE" 44410;
"PROCEDURE" CORMAT(CHN, NROFCOL, FORMATVARLABELS,
    FORMATMATRIXELEMENTS, TEXT, ORDER, NRESP, SKIP,
    READRESP, LABELS);
    "VALUE" CHN, NROFCOL, ORDER, NRESP,SKIP; "REAL" SKIP;
    "INTEGER" CHN, NROFCOL, ORDER, NRESP;
    "STRING" FORMATVARLABELS, FORMATMATRIXELEMENTS, 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 1, ADDR, OLDPP;


    "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, "("*,N")", TEXT);
        SYSPARAM(CHN, 2, 71);
        OUTPUT(CHN, "(""("PAGE:")",3ZD/")", PAGE)
    "END";


    "PROCEDURE" HEADING;
    "BEGIN" "INTEGER" I; OUTPUT(CHN, "("/")");
        OUTPUT(CHN, FORMATVARLABELS, 0);
        OUTPUT(CHN, "("2B")");
        "FOR" I:= LOW "STEP" 1 "UNTIL" UPP "DO"
        OUTPUT(CHN, FORMATVARLABELS, VAR LABELS[I]);
    "END";

    "PROCEDURE" NEWPAGE;
```

```
    "IF" LPP60 "THEN"
    "BEGIN" HEADING; HEADLINE; HEADING;
            OUTPUT(CHN, "("/")") "END";
    "IF" CHN <= 0 "OR" CHN = 60 "THEN"
        STATAL3 ERROR("("CORMAT")", 1, CHN);
    SYSPARAM(CHN, 7, OLDPP); LPP60:= OLDPP ↑= 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[J]:= 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[1 : "IF" COR IN CORE
                        "THEN" CORSPACE "ELSE" 1];

    "IF" FEW RESP "THEN"
    "BEGIN" "ARRAY" DATA[1 : "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, DATA[I, J], 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"
"END" READ DATA;

"BEGIN" "REAL" XI, XJ, SI, SJ, SII, SIJ, SJJ, DENOM;
    "ARRAY" VARI, VARJ[1 : NRESP];
    "INTEGER" ADDR1, ADDR2;

    ADDR2:= MINADDR + 1;
    "IF" COR IN CORE "THEN" COR[1]:= 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:= SJ:= SII:= SIJ:= SJJ:=0;
            RESPCOUNTER:= 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:= VARI[K]; XJ:= VARJ[K];
                "IF" XI ≠ SKIP & XJ ≠ SKIP "THEN"
                "BEGIN" SI:= SI + XI; SJ:= SJ + XJ;
                    SII:= SII + XI * XI;
                    SIJ:= SIJ + XI * XJ;
```

```
                        SJJ:= SJJ + 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" SOM12, SOM21, KSOM12, KSOM21, COUNT[1 :
          "IF" SUMS IN CORE "THEN" SUMSPACE "ELSE" 1];

    "INTEGER" RESPONDENT, COR POINTER, POINTER,
              N NOT MISS, VAR1, VAR2, VAR11, ADDR1,
              ADDR2, AANT RESP;
    "REAL" SCORE1, SCORE2, S12, S21, DENOM;

    "IF" SUMS IN CORE "THEN"
    "BEGIN" "FOR" I:= 1 "STEP" 1 "UNTIL" SUMSPACE "DO"
        "BEGIN"
            COR[I]:= SOM12[I]:= SOM21[I]:=
            KSOM12[I]:= KSOM21[I]:= 0;
            COUNT[I]:= 0;
        "END";
        "FOR" I:= SUMSPACE + 1 "STEP" 1 "UNTIL" CORSPACE
        "DO" COR[I]:= 0
    "END" "ELSE"
    "BEGIN" "ARRAY" ZERO[1 : ORDER]; "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" RESPONDENT:= 1 "STEP" 1 "UNTIL" NRESP "DO"
        "BEGIN" READRESP(I, SCORES[I], ORDER, SKIP);
            N NOT MISS:= POINTER:= 0;
            "IF" SCORES[1] ≠ SKIP "THEN"
```

```
"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 : VAR1],
        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);

        "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" I:= 1 "STEP" 1
           "UNTIL" N NOT MISS "DO"
        "BEGIN" VAR2:= NOT MISSING[I];
            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[N 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(POINTER + 1,
                        POINTER + VAR11,
                         -POINTER, SOM12, SUM12);
                    DUPVEC(POINTER + 1,
                        POINTER + VAR11,
                         -POINTER, SOM21, SUM21);
                    DUPVEC(POINTER + 1,
                        POINTER + VAR11,
                         -POINTER, KSOM12, KSUM12);
                    DUPVEC(POINTER + 1,
                        POINTER + VAR11,
                         -POINTER, KSOM21, KSUM21);
                    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";


            "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" VAR1:= 2 "STEP" 1 "UNTIL" ORDER "DO"
"BEGIN" "ARRAY" CORREL[1 : VAR1],
 SUM12, SUM21, KSUM12, KSUM21, RCOUNT[1 : VAR1 - 1];

    VAR11:= VAR1 - 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[VAR1]:= 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, KSUM12);
    DUPVEC(POINTER + 1, POINTER + VAR11,
           -POINTER, KSOM21, KSUM21);

    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, "("N")", 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"
                   DUPVEC(LOW, MAX, J * (J-1) / 2, CORREL,
                                                       COR)
                  "ELSE"
                   "BEGIN"
                       ADDR:= (J - 1) * J / 2 + LOW +
                                                  MINADDR;
                       FETCH ARRAY(FILENR, ADDR, CORREL);
                   "END";
                   "FOR" I:= LOW "STEP" 1 "UNTIL" MAX "DO"
                   OUTPUT(CHN, FORMATMATRIXELEMENTS,
                   "IF" ABS(CORREL[I]) < 1.00005
                   "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(FILENR);
    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 coëfficient 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 indentifier>, 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:

Errornumber 1 (if CHN ≤0 or CHN = 60)

PROCEDURES USED

TAMMAT	NUMAL 34014
STATAL3 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)=(SXY-SX*SY/NRESP)/
SQRT((SXX-SX*SX/NRESP)*(SYY-SY*SY/NRESP)),
```

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 computed 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"
```

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 1389.000
  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

        1       2       3       4
```

SOURCE TEXT

```
"CODE" 44411;
"PROCEDURE" MINICORMAT(CHN, ORDER, NRESP, READRESP,
                                          CORMAT TO FILE);
    "VALUE" CHN, ORDER, NRESP, CORMAT TO FILE;
    "INTEGER" CHN, ORDER, NRESP; "BOOLEAN" CORMAT 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, "("/,4B")");
    "FOR" I:= LOW "STEP" 1 "UNTIL" UPP "DO"
    OUTPUT(CHN, "("5ZDBB")", I)
"END" HEADING;

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

"FOR" J:= 1 "STEP" 1 "UNTIL" ORDER "DO"
"BEGIN" SUMJ:= MINJ:= MAXJ:= DATA[1, 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]:= SUMJ;
    DEVJ:= TAMMAT(1, NRESP, J, J, DATA, DATA) * NRESP -
                    SUMJ * SUMJ;
    DATADEV[J]:= DEVJ:= "IF" DEVJ <= 0
                        "THEN" 0 "ELSE" SQRT(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" DATADEV[I] = 0 "THEN" 2 "ELSE"
    (TAMMAT(1, NRESP, I, J, DATA, DATA) * NRESP -
    DATASUM[I] * SUMJ) / (DATADEV[I] * DEVJ);
    OUTPUT(CHN, "("/,2ZD,-5ZD.3D,3(-3ZD.3D)")",
        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")",10B,
       "("(CORRELATIONS MULTIPLIED BY 100000)")"")");
    SYSPARAM(CHN, 4, K // 10 + K + 1);
    HEADING; OUTPUT(CHN, "("/")");
    "FOR" J:= LOW "STEP" 1 "UNTIL" ORDER "DO"
    "BEGIN" OUTPUT( CHN, "("/,2ZDBB")", 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, "("/,"("1")"")");
    "IF" CORMAT TO FILE
    "THEN" OUTPUT(65, "("(-D.5DBB)")", CORREL)
"END" MINI CORMAT;
        "EOP"
```

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 factorscore coefficients are computed.

KEYWORDS

Joreskog factor analysis

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, READRESP 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 indentifier>, to contain integer labels for the variables;
NVAR: <integer arithmetic expression>, number of variables;
RESPNR: <integer array indentifier>, 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 "NTIL" 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 corvariance 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 negative, the analysis of the covariance martix 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) factors 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 $\leqslant$ number of latent roots and vectors, and $\geqslant$ 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 a 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,
<N> 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 R^2.
12: Matrix S*, its trace and determinant. (see Joreskog, 1963, p25).
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 ≥ 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 factorloadings.
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 factorloadings 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.
16.2: factorscores, computed using standardized data. (see Joreskog, 1963 p.40).
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
DUPVECROW	NUMAL 31031
DUPROWVEC	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

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 of 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 ocurring in practice, at least T200 is required.

REFERENCES
[1] K. G. Joreskog,
Statistical estimation in factor analysis,
Almqvist and Wiksell, Uppsala, 1963.
[2] Algol 60 reference manual, version 5,
Control Data Corporation, 1979.
[3] H.H. Harman,
Modern factor analysis,
University of Chicago Press, Chicago, 1970.
[4] J.G. Bethlehem, H. Elffers, R.D. Gill & J. Rijvordt,
Methoden, Voetangels en klemmen in de factoranalyse,
Mathematical Centre report SN 7/77,
Mathematical Centre, Amsterdam, 1977.

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;  ;

    JORESKOG(.05, READRESP, LABELS)
"END"
```

Input:

```
CHANNEL,62=60
CHANNEL,63=60
CHANNEL,64=61
CHANNEL,1=A,A
CHANNEL,2=B,A
CHANNEL,3=C,A
CHANNEL,4=D,A
CHANNEL,5=E,A
CHANNEL,6=F,A
CHANNEL,7=G,A
CHANNEL,8=H,A
CHANNEL,END
100  11  3  8  0  1  2
1.0000
 .7949  1.0000
 .7432   .7964  1.0000
 .7317   .7449   .6490  1.0000
 .7512   .6653   .5305   .6009  1.0000
 .7676   .4492   .4046   .4562   .5114  1.0000
 .8556   .5658   .5252   .5219   .7259   .7316  1.0000
 .4445   .1955   .1601   .2037   .2619   .6392   .3956  1.0000
 .3514   .1878   .1756   .1690   .1829   .4133   .2884   .7154  1.0000
 .4163   .2010   .1719   .2163   .2471   .5491   .3843   .6571   .4733  1.0000
 .3867   .1810   .1902   .2362   .1636   .5084   .3255   .6638   .5586   .7229  1.0000
```

Output:

```
CHANNEL,62=60
CHANNEL,63=60
CHANNEL,64=61
CHANNEL,1=A,A
CHANNEL,2=B,A
CHANNEL,3=C,A
CHANNEL,4=D,A
CHANNEL,5=E,A
CHANNEL,6=F,A
CHANNEL,7=G,A
CHANNEL,8=H,A
CHANNEL,END
```

```
JORESKOG FACTOR ANALYSIS
========================

NUMBER OF RESPONDENTS:                                  100
NUMBER OF VARIABLES:                                     11
CODE FOR ENTERED MATRIX:                                  3
NUMBER OF LATENT ROOTS AND VECTORS TO BE COMPUTED:        8
NUMBER OF UNROTATED FACTORS:                              0
NUMBER OF ANALYSES TO BE PERFORMED:                       1
NUMBERS OF FACTORS:                                       2
```

CORRELATION MATRIX

	1	2	3	4	5	6	7	8	9	10	11
1	1.0000										
2	.7949	1.0000									
3	.7432	.7964	1.0000								
4	.7317	.7449	.6490	1.0000							
5	.7512	.6653	.5305	.6009	1.0000						
6	.7676	.4492	.4046	.4562	.5114	1.0000					
7	.8556	.5658	.5252	.5219	.7259	.7316	1.0000				
8	.4445	.1955	.1601	.2037	.2619	.6392	.3956	1.0000			
9	.3514	.1878	.1756	.1690	.1829	.4133	.2884	.7154	1.0000		
10	.4163	.2010	.1719	.2163	.2471	.5491	.3843	.6571	.4733	1.0000	
11	.3867	.1810	.1902	.2362	.1636	.5084	.3255	.6638	.5586	.7229	1.0000
	1	2	3	4	5	6	7	8	9	10	11

	DIAG INV	MULT R	MULT R*R
1	+.1477"+02	+.9656"+00	+.9323"+00
2	+.4776"+01	+.8892"+00	+.7906"+00
3	+.3371"+01	+.8387"+00	+.7034"+00
4	+.2739"+01	+.7968"+00	+.6349"+00
5	+.2884"+01	+.8082"+00	+.6532"+00
6	+.4114"+01	+.8700"+00	+.7569"+00
7	+.5284"+01	+.9004"+00	+.8108"+00
8	+.3624"+01	+.8509"+00	+.7241"+00
9	+.2202"+01	+.7388"+00	+.5458"+00
10	+.2486"+01	+.7732"+00	+.5978"+00
11	+.2608"+01	+.7852"+00	+.6166"+00

MATRIX S*

	1	2	3	4	5	6	7	8	9	10	11
1	+.1477"+02										
2	+.6677"+01	+.4776"+01									
3	+.5245"+01	+.3196"+01	+.3371"+01								
4	+.4654"+01	+.2694"+01	+.1972"+01	+.2739"+01							
5	+.4903"+01	+.2469"+01	+.1654"+01	+.1689"+01	+.2884"+01						
6	+.5984"+01	+.1991"+01	+.1507"+01	+.1531"+01	+.1761"+01	+.4114"+01					
7	+.7559"+01	+.2842"+01	+.2217"+01	+.1986"+01	+.2834"+01	+.3411"+01	+.5284"+01				
8	+.3252"+01	+.8134"+00	+.5597"+00	+.6418"+00	+.8467"+00	+.2468"+01	+.1731"+01	+.3624"+01			
9	+.2004"+01	+.6090"+00	+.4784"+00	+.4150"+00	+.4609"+00	+.1244"+01	+.9837"+00	+.2021"+01	+.2202"+01		
10	+.2523"+01	+.6926"+00	+.4977"+00	+.5645"+00	+.6616"+00	+.1756"+01	+.1393"+01	+.1972"+01	+.1107"+01	+.2486"+01	
11	+.2400"+01	+.6388"+00	+.5640"+00	+.6313"+00	+.4487"+00	+.1665"+01	+.1208"+01	+.2041"+01	+.1339"+01	+.1841"+01	+.2608"+01
	1	2	3	4	5	6	7	8	9	10	11

```
TRACE:          4.886"+01
DETERMINANT:    1.368"+02
```

LATENT ROOTS AND VECTORS OF S*

	+.3171"+02	+.7174"+01	+.2865"+01	+.1469"+01	+.1308"+01	+.1136"+01	+.9054"+00	+.6494"+00
1	-.6725"+00	+.1211"+00	-.9463"-01	-.2400"-01	-.1839"+00	-.4217"-01	+.1387"+00	+.1218"+00
2	-.3120"+00	+.3196"+00	+.4251"+00	+.7032"-01	+.1048"+00	+.8703"-01	-.3822"+00	+.5562"+00
3	-.2420"+00	+.2545"+00	+.3792"+00	-.4978"-01	-.3306"+00	-.4388"+00	-.1344"+00	-.5842"+00
4	-.2179"+00	+.1815"+00	+.2562"+00	-.1320"+00	+.1827"+00	+.5601"+00	+.5575"+00	-.2164"+00
5	-.2326"+00	+.1401"+00	-.1849"+00	+.2305"+00	+.6781"+00	+.9433"-01	-.2473"+00	-.3376"+00
6	-.2900"+00	-.2627"+00	-.2835"+00	-.1438"+00	-.4543"+00	+.4384"+00	-.2154"+00	+.1809"-01
7	-.3542"+00	-.4854"-02	-.5609"+00	+.2834"-01	+.1240"+00	-.3791"+00	+.1923"+00	+.6383"-01
8	-.1784"+00	-.5276"+00	+.1511"+00	+.3822"+00	-.3488"-01	+.2056"+00	-.2494"+00	-.3179"+00
9	-.1102"+00	-.3338"+00	+.2514"+00	+.5959"+00	+.1052"-01	-.2052"+00	+.3185"+00	+.2509"+00
10	-.1380"+00	-.3786"+00	+.1281"+00	-.4814"+00	+.3176"+00	-.1216"+00	-.3148"+00	+.6789"-01
11	-.1329"+00	-.4047"+00	+.2633"+00	-.4087"+00	+.1696"+00	-.2010"+00	+.3140"+00	+.2589"-01

2 - FACTOR SOLUTION

BASED ON CORRELATION MATRIX:

CANONICAL FACTORLOADINGS

	+.5427"+01	+.1971"+01
1	-.9679"+00	+.7757"-01
2	-.7897"+00	+.3601"+00
3	-.7291"+00	+.3413"+00
4	-.7281"+00	+.2700"+00
5	-.7576"+00	+.2032"+00
6	-.7910"+00	-.3189"+00
7	-.8523"+00	-.5200"-02
8	-.5182"+00	-.6825"+00
9	-.4108"+00	-.5540"+00
10	-.4840"+00	-.5913"+00
11	-.4554"+00	-.6172"+00

GUTTMAN-CRITERION

1	+.9300"+00
1	+.6908"+00

TRANSFORMATION MATRIX

	1	2
1	+.8815"+00	+.4721"+00
1	-.4721"+00	+.8815"+00

NUMBER OF ITERATIONS: 2

```
      VARIMAX ROTATED FACTORLOADINGS

      +.4285"+01  +.3113"+01

 1    -.8898"+00  -.3886"+00
 2    -.8662"+00  -.5536"-01
 3    -.8038"+00  -.4333"-01
 4    -.7693"+00  -.1057"+00
 5    -.7637"+00  -.1786"+00
 6    -.5467"+00  -.6546"+00
 7    -.7489"+00  -.4070"+00
 8    -.1346"+00  -.8463"+00
 9    -.1006"+00  -.6823"+00
10    -.1475"+00  -.7498"+00

11    -.1100"+00  -.7591"+00

      NUMBER OF UNROTATED FACTORS:  0


      GUTTMAN-CRITERION AFTER ROTATION

 1    +.8767"+00
 1    +.7441"+00


      PERCENTAGE OF VARIANCE EXPLAINED:  67.26


      COMMUNALITIES

 1    +.9428"+00
 2    +.7533"+00
 3    +.6480"+00
 4    +.6030"+00
 5    +.6152"+00
 6    +.7274"+00
 7    +.7264"+00
 8    +.7343"+00
 9    +.4757"+00
10    +.5839"+00

11    +.5883"+00


    RESIDUAL CORRELATION MATRIX

          1          2          3          4          5          6          7          8          9         10         11

 1   -.1788"-01
 2   +.2647"-02 +.1449"-01
 3   +.1108"-01 +.9774"-01 +.2301"-01
 4   +.6044"-02 +.7268"-01 +.2600"-01 -.7917"-02
 5   +.2209"-02 -.6105"-02 -.9116"-01 -.5542"-02 +.2303"-03
 6   +.2677"-01 -.6057"-01 -.6322"-01 -.3360"-01 -.2303"-01 +.3022"-02
 7   +.3110"-01 -.1054"+00 -.9439"-01 -.9724"-01 +.8130"-01 +.5581"-01 +.6371"-01
 8   -.4141"-02 +.3205"-01 +.1523"-01 +.1066"-01 +.7959"-02 +.1163"-01 -.4962"-01 -.4034"-01
 9   -.3263"-02 +.6289"-01 +.6517"-01 +.1946"-01 -.1578"-01 -.8835"-01 -.6462"-01 +.1244"+00 +.2054"-01
10   -.6293"-02 +.3174"-01 +.2085"-01 +.2356"-01 +.5584"-03 -.2233"-01 -.3128"-01 +.2724"-02 -.5313"-01 -.2999"-01

11   -.6159"-02 +.4369"-01 +.6887"-01 +.7130"-01 -.5598"-01 -.4862"-01 -.6580"-01 +.6597"-02 +.2959"-01 +.1376"+00 -.1351"-01

          1          2          3          4          5          6          7          8          9         10         11


TEST OF FIT:
============

MEAN OF THE   9 LOWEST LATENT ROOTS OF S-STAR:  1.109"+00
U-STATISTIC:                                    1.426"+02
DEGREES OF FREEDOM:                            44



END OF JORESKOG
```

SOURCE TEXT

```
"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, COVM, 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, SINFI;
        "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 [J, I]:= 0
        "END";
        COUNTER:= 0; NFACTORMIN1:= NFACTOR - 1;
    ROTATION: 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;
```

```
            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" SINFI:= - SINFI;
            ROTCOL (1, NTEST, J, K, F, COSFI, SINFI);
            "IF" ROTATIONMATRIX
            "THEN" ROTCOL (NFACTOR1, NFACTOR, J, K,
                                      T, COSFI, SINFI)
        "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";
    VARIMAXROTATION:= COUNTER
"END" VARIMAX ROTATION;

"PROCEDURE" STANDARD SCORES (COVM, SCORES, STANDARD,
                                          C, K, NRESP);
"VALUE" COVM, SCORES, STANDARD, K, NRESP;
"INTEGER" COVM, SCORES, STANDARD, K, NRESP; "ARRAY" C;
"BEGIN" "INTEGER" I, J; "REAL" SQRTSJ;
    "ARRAY" S, G, BUF [1 : 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" 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" SQRTSJ:= SQRT (S [J]);
```

```
            BUF [J]:= "IF" SQRTSJ < "-8 "THEN" .9999"9
            "ELSE" (BUF [J] - G [J]) / SQRTSJ
        "END";
        PUT ARRAY (STANDARD, BUF)
    "END"
"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 (1, 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";
```

```
            "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 ROW VEC (HL, HU, I, X, BUF)
                    "END";
                    PAGE:= RECTANGLE (OC, HEADING, X, VL,
                      VU, HL, HU, VARNR, 2, RESPNR, PAGE);
                    "IF" HU ^= K "THEN"
                    "FOR" I:= 1 "STEP" 1 "UNTIL" 50
                    "DO" BACKSPACE (FILE);
                    HU:= HU + AA; "IF" HU > 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, 1 : 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, 2, 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, I1, 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]; I1:= I + 1; IJ:= II + I;
        "FOR" J:= I1 "STEP" 1 "UNTIL" N "DO"
        "BEGIN" U [J]:= A [IJ]; IJ:= IJ + J "END";
        "FOR" J:= N "STEP" - 1 "UNTIL" I1 "DO"
        "BEGIN" JJ:= IJ - I; IJ:= IJ - J;
            A [IJ]:= - (VECVEC (I1, J, JJ - J, U, A) +
             SEQVEC (J + 1, N, JJ + J, 0, A, U)) * R
        "END";
        A [II]:= (R - SEQVEC (I1, N, II + I, 0, A, U))
                 * R;
        II:= II - I
    "END"
"END" INVSYM1;

"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;
        "IF" A [KK] > NORM "THEN" NORM:= A [KK]
    "END";
    EPSNORM:= AUX [0] * NORM; AUX [1]:= 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]:= A [KJ] -
                    VECVEC (LOW, UP, KJ - KK, A, A);
                    R:= ABS (R);
                    "IF" R > M "THEN" M:= R;
                    KJ:= KJ + J
                "END"
            "END";
            "GOTO" END
```

```
        "END";
        KK:= KK + K; LOW:= KK - K + 1; UP:= KK - 1;
        P [K]:= 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,
                                                    A);
            ICHSEQ (PK + 1, N, PP + K, PK - K, A);
            A [PP]:= A [KK]
        "END";
        A [KK]:= R:= SQRT (MAX); KJ:= KK + K; JJ:= KK;
        "FOR" J:= K + 1 "STEP" 1 "UNTIL" N "DO"
        "BEGIN" W:= A [KJ]:= (A [KJ] -
             VECVEC (LOW, UP, KJ - KK, A, A)) / R;
            JJ:= JJ + J; A [JJ]:= A [JJ] - W * W;
            KJ:= KJ + J
        "END"
    "END";
    K:= N + 1;
END: AUX [2]:= M; AUX [3]:= D;
    RNKSYM10:= "IF" M <= 2 * EPSNORM
                "THEN" K - 1 "ELSE" - K
"END" RNKSYM10;

"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 ^= I "THEN"
        "BEGIN" PP:= (PI + 1) * PI // 2;
            ICHVEC (II - I + 1, II - 1,
                                PP - PI - II + I, A);
            ICHSEQVEC (I + 1, PI - 1,
                                II + I, PP - PI, A);
            ICHSEQ (PI + 1, N, PP + I, PI - I, A);
            R:= A [II]; A [II]:= A [PP]; A [PP]:= R
        "END";
        II:= II - I
    "END"
"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")", "("LFN")", "("CINFILE")",
              "("P")", 80);
"IF" "NOT" CHEXIST(IN) "THEN"
```

```
CHANNEL(IN, "("C")", "("LFN")", "("INFILE")",
            "("P")", 80);
"IF" "NOT" CHEXIST(OC) "THEN"
CHANNEL(OC, "("C")", "("LFN")", "("OUTFILE")",
            "("P")", 136, "("PP")", 60);
"FOR" CHN:= SCORES, COVM, CORRM, INV, STANDARD,
            REDCOV, REDCORR, FSCORES "DO"
"BEGIN" "IF" "NOT" CHEXIST(CHN) "THEN"
    CHANNEL(CHN, "("B")");
    OPEN(CHN, "("IO")");
"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:= -"7;
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, RE1, TRACE, DI, DIAGI, DET, VV;
    "INTEGER" "ARRAY" ICC, VARNR, H [1 : K],
                                  VALUE [1 : NVAL];
    "ARRAY" C [1 : T], EM [0 : 9], AUX [0 : 3], D, CC,
        DIAG [1 : K], KOM [1 : K, 1 : 2];
    "BOOLEAN" "ARRAY" BCC [1 : 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" ININTARRAY(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:                              ")",
              6ZD,/,
  "("NUMBER OF VARIABLES:                                ")",
              6ZD,/,
  "("CODE FOR ENTERED MATRIX:                            ")",
              -5ZD,/,
  "("NUMBER OF LATENT ROOTS AND VECTORS TO BE COMPUTED:")",
              6ZD,/,
  "("NUMBER OF UNROTATED FACTORS:                        ")",
              6ZD,/,
  "("NUMBER OF ANALYSES TO BE PERFORMED:                 ")",
              6ZD,/,
  "("NUMBERS OF FACTORS:                                 ")",
              2B")",
       NRESP, K, "IF" MATRIX = -3 "THEN" 3 "ELSE" MATRIX,
       P, P1 - 1, NVAL);
      "FOR" I:= 1 "STEP" 1 "UNTIL" NVAL
      "DO" OUTPUT (OC, "("B3ZDB")", ABS (VALUE [I]));
      OUTPUT (OC, "("/")");
      "IF" MATRIX < 0 "AND" MATRIX ≠ - 3
      "THEN" OUTPUT (OC, "("/,
             "("THE ANALYSIS OF THE COVARIANCE MATRIX ")",
                    "("WILL BE SUPPRESSED")",//")");
      "IF" SKIP ≠ - "7
      "THEN" OUTPUT (OC, "("//,48("("-")"),/,
"("THE MISSING OBSERVATIONS OF A VARIABLE, CODED BY")",/,
      -D.3D"+DD,"(", 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, VEC
             [1 : K, 1 : ("IF" P > 2 "THEN" P "ELSE" 3)],
              HELPVEC [1 : K], ROTMAT [1 : P, 1 : P], VAL,
              KOM2 [1 : P];
          "INTEGER" "ARRAY" RR[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" K "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 + I]:= (TAMMAT (1,
                   NRESP, I, J, X, X) * NRESP -
                   S [I] * SJ) / (NRESP - 1) / NRESP
            "END";
            "FOR" I:= 1 "STEP" 1 "UNTIL" NRESP "DO"
            "BEGIN" DUP VEC ROW (1, 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 [J], 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 [J];
                    S [J]:= S [J] + XJ;
                    "FOR" I:= J "STEP" - 1 "UNTIL" 1
                    "DO" C [PJ + I]:= C [PJ + I] +
                         X [I] * XJ
                "END"
            "END";
            "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 [I] * SJ) /
                     (NRESP - 1) / NRESP
            "END"
        "END";
        "GOTO" OUTP;

    SYSTEM3:
        "BEGIN" "ARRAY" BUF, S, M [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 [J]:= S [J] + 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 [J] / (NRESP -
                                OWVAR [J]);
            S [J]:= 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 [I];
                "FOR" II:= 1 "STEP" 1
                            "UNTIL" A
```

```
                                "DO"
                                "BEGIN"
                                   "IF" X [II, I] = SKIP
                                   "THEN" X [II, I]:= MI;
                                   "IF" X [II, J] = SKIP
                                   "THEN" X [II, J]:= MJ;
                                   C [PJ + I]:= C [PJ + I] +
                                   X [II, I] * X [II, J]
                                "END"
                             "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 [I] * SJ)
                                    / (NRESP - 1) / NRESP
                          "END"
                       "END";
                       REWIND (INV)
                    "END";

              OUTP:
                    "IF" SKIP ^= - "7 "THEN"
                    "BEGIN" OUTPUT (OC, "("5/,
   "("FREQUENCY OF MISSING OBSERVATIONS FOR EACH VARIABLE:")",
                              //,64("("-")"),/,
                              "("SECOND DIGIT I")",10(4BD),/,
                         "("FIRST DIGIT  I")",/,13B,"("I")",/,
                                64("("-")"),/,13B,"("I")",/,
                                 5BD,7B,"("I")",6B")",
                              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,
                                "("/,5ZD,7B,"("I")",B")", J);
                          J:= OWVAR [I];
                          "IF" J = 0
                          "THEN" OUTPUT (OC, "(""("   - ")"")")
                          "ELSE" OUTPUT (OC, "("B,ZZD,B")", J);
                       "END";
                       OUTPUT (OC, "("/,64("("-")"),6/,
```

```
     "("NUMBER OF RESPONDENTS, A[J], WITH J ")",
               "("MISSING VALUES:")",/,
               51("=")"),//,3B,
     "("J  A[J]")",/,2B,8("("-")"),4/")");
        "FOR" I:= 1 "STEP" 1 "UNTIL" K "DO"
        "BEGIN" J:= OWRESP [I];
            "IF" J ≠ 0
            "THEN" OUTPUT (OC,
                   "("BZZD,B,3ZD,/")", 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"
"END"
"ELSE"
"BEGIN" REWIND (COVCORR);
        GET ARRAY (COVCORR, C) "END";
AUX [0]:= "-11; 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 [1 : RANK], B [1 : 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]:=
                    SQRT (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" .9999"9
            "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:= REDCORR;
    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;
    T:= B [I]; B [I]:= B [J]; B [J]:= T;
    M [I]:= N [I]:= I;
"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 [NI]:= M [BI -
                               VARNR [BI]];
        N [J]:= NI
    "END";
    T:= 0;
    "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 = COVM
               "THEN" CORRM "ELSE"
     "IF" COVCORR = REDCOV "THEN" REDCORR
     "ELSE" 77,
     C);
    REWIND (COVCORR); GET ARRAY (COVCORR, C);
"END"
"ELSE"
TRIANGLE (OC, "("CORRELATION MATRIX")", C,
```

```
                                                -K, VARNR);
    TRACE:= 0; COUNTER2:= 0; DET:= AUX [3];
    "FOR" I:= 1 "STEP" 1 "UNTIL" K "DO"
    "BEGIN" DI:= D [I];
        RE:= 1 - 1 / C [COUNTER2 + I] / DI / DI;
        "IF" RE < 0 "THEN" RE:= 0; VEC [I, 3]:= 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, 1]:= DI:= DI * DI; DET:= DET * DI;
        VEC [I, 2]:= SQRT (RE)
    "END";
    RECTANGLE (OC,
     "("DIAG INV    MULT R      MULT R*R")",
     VEC, 1, K, 1, 3, VAL, 0, VARNR, 1);
    DUP VEC COL (1, K, 1, HELPVEC, VEC);
    TRIANGLE (OC, "("MATRIX S*")", C, K, VARNR);
    OUTPUT (OC,
     "("3/,8B,"("TRACE:")",8B,-D.3D"+DD,/,
           8B,"("DETERMINANT:")",2B,-D.3D"+DD,/")",
     TRACE, DET);
    "IF" P > K "THEN" P:= K;
    "IF" P = 0 "THEN" "GOTO" NEXTPROB;
    EM [0]:= "-12; EM [2]:= "-10; EM [4]:= "-3;
    EM [6]:= "-8; EM [8]:= 5;
    EIGSYM1 (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("("=")"), 2/")", P);
    RE:= (TRACE - SUM (I, 1, P, VAL [I])) / (K - P);
    "FOR" I:= 1 "STEP" 1 "UNTIL" K "DO"
    "BEGIN" DI:= D [I];
        "FOR" J:= 1 "STEP" 1 "UNTIL" P
        "DO" L [I, J]:= LR [I, J]:= VEC [I, J] *
             SQRT (VAL [J] - RE) / DI
    "END";
    "FOR" I:= 1 "STEP" 1 "UNTIL" P
    "DO" KOM2 [I]:= TAMMAT (1, K, I, I, L, L);
    VV:= SUM (I, 1, P, KOM2 [I]) * 100 /
```

```
("IF" MATRIX = - 3 "THEN" K
 "ELSE" VECVEC (1, K, 0, DIAG, DIAG));
"IF" MATRIX > 0 "OR" MATRIX = - 3 "THEN"
"BEGIN" "IF" MATRIX = -3 "THEN"
    "BEGIN"
        OUTPUT (OC, "(""("BASED ON ")",
              "("CORRELATION MATRIX:")",/,
              28("("-")"),//")");
        RECTANGLE (OC,
              "("CANONICAL FACTORLOADINGS")",
              L, 1, K, 1, P, KOM2, 1, VARNR, 1);
    "END" "ELSE"
    "BEGIN"
        OUTPUT (OC, "(""("BASED ON ")",
             "("COVARIANCE MATRIX:")",/,
             27("("-")"),//")");
        RECTANGLE (OC,
             "("CANONICAL FACTORLOADINGS")",
             L, 1, K, 1, P, KOM2, 1, VARNR, 1)
    "END";
        "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);
    "IF" P - P1 < 1 "THEN"
    OUTPUT (OC, "("4/,8B,"("PERCENTAGE OF ")",
      "("VARIANCE EXPLAINED:")",3ZD.DD/")", VV)
"END";
"IF" P - P1 < 1 "THEN" "GOTO" NOR1;
HI:= VARIMAXROTATION (LR, K, P1, P, "TRUE",
     ROTMAT, DI, EPS);
"IF" MATRIX > 0 "OR" MATRIX = -3 "THEN"
"BEGIN"
    RECTANGLE(OC, "("TRANSFORMATION MATRIX")",
     ROTMAT, 1, P, 1, P, DUPFACNR, 2, FACNR, 1);
    OUTPUT(OC, "("//,8B,
     "("NUMBER OF ITERATIONS:")",2ZD, /")", HI);
"END";
"FOR" J:= 1 "STEP" 1 "UNTIL" P
"DO" KOM2 [J]:= TAMMAT (1, K, J, J, LR, LR);
"IF" MATRIX > 0 "OR" MATRIX = - 3 "THEN"
"BEGIN"
    RECTANGLE (OC,
       "("VARIMAX ROTATED FACTORLOADINGS")",
       LR, 1, K, 1, P, KOM2, 1, VARNR, 1);
    OUTPUT(OC,"("//,8B,
       "("NUMBER OF UNROTATED FACTORS:")",
        2ZD,/")", P1 - 1);
    "BEGIN" "REAL" "ARRAY" G [1 : P];
        DUP VEC COL (1, P, 2, G, 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, "("4/, 8B,
            "("PERCENTAGE OF VARIANCE EXPLAINED:")",
            3ZD.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);
                COUNTER2:= COUNTER2 + 1;
                C [COUNTER2]:= C [COUNTER2] - DI
            "END";
            C [COUNTER2]:= C [COUNTER2] - RE / D [I] ** 2;
            KOM [I, 1]:= DI
        "END";
        "IF" MATRIX = - 3 "THEN" "GOTO" CONTINUE;
        "IF" MATRIX > 0 "THEN"
        "BEGIN" RECTANGLE (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" K "DO"
        "BEGIN" DIAGI:= DIAG [I];
            "FOR" J:= 1 "STEP" 1 "UNTIL" P "DO"
            "BEGIN" L [I, J]:= L [I, J] / DIAGI;
                LR [I, J]:= LR [I, J] / DIAGI
            "END"
        "END";
        "FOR" I:= 1 "STEP" 1 "UNTIL" P
        "DO" KOM2 [I]:= TAMMAT (1, K, I, I, L, L);
        SYSPARAM (OC, 3, I);
        "IF" MATRIX > 0 "OR" MATRIX = -3
        "THEN" OUTPUT (OC, "("*")")
        "ELSE" "IF" I > 3 "THEN" OUTPUT (OC, "("3/")");
        OUTPUT (OC,
             "(""("BASED ON CORRELATION MATRIX:")",/,
             28("("-")"),//")");
        RECTANGLE (OC, "("CANONICAL 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]:= TAMMAT (1, K, J, J, LR, LR);
        VV:= SUM (I, 1, P, KOM2 [I]) * 100 / K;
        "IF" P - P1 < 1 "THEN"
        "BEGIN"
             OUTPUT(OC, "("4/,8B,"("PERCENTAGE OF ")",
```

```
            "("VARIANCE EXPLAINED:")",3ZD.DD/")", VV);
        "GOTO" NOR2
    "END";
    RECTANGLE(OC,"("TRANSFORMATION MATRIX")",
        ROTMAT, 1, P, 1, P, DUPFACNR, 2, FACNR, 1);
    OUTPUT(OC, "("//,8B,
        "("NUMBER OF ITERATIONS:")", 2ZD, /")", HI);
    RECTANGLE (OC,
        "("VARIMAX ROTATED FACTORLOADINGS")", LR, 1,
          K, 1, P, KOM2, 1, VARNR, 1);
    OUTPUT(OC, "("//,8B,
         "("NUMBER OF UNROTATED FACTORS:")",
        2ZD,/")", P1 - 1);
    "BEGIN" "REAL" "ARRAY" G [1 : P];
        DUP VEC COL (1, P, 2, G, 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, "("4/,
8B,"("PERCENTAGE OF VARIANCE EXPLAINED:")",3ZD.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);
    RE1:= VAL [1];
    "FOR" I:= 2 "STEP" 1 "UNTIL" P
    "DO" RE1:= RE1 * VAL [I];
    RE1:= DET / RE1;
    "IF" PR FSCORES "THEN"
    "BEGIN" "REAL" X; "INTEGER" COUNTER3;
        "REAL" "ARRAY" FSM [1 : HK, 1 : P];

        REWIND(FSCORES);
        "FOR" I:= 1 "STEP" 1 "UNTIL" K "DO"
        ROW CST (1, P, I, L,
                HELPVEC [I] * 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 ")",2ZD,
                   "(" LOWEST LATENT ROOTS ")",
                   "("OF S-STAR:")",B,-D.3D"+DD,/,
                   "("U-STATISTIC:")",35B,-D.3D"+DD,/,
                   "("DEGREES OF FREEDOM:")",26B,3ZD,/")",
                   K - P, RE, ((K - P) * LN (RE) - LN (RE1))
                   * (NRESP - 1), (K - P - 1) *
                   (K - P + 2) // 2);
            "GOTO" NEXTP;
        "END"
    "END";
DUMP: OUTPUT (OC, "("*,10/,
         20B,"("*** EOF / EOR ENCOUNTERED ON CHANNEL ")",
         ZD,"(" ***")", /,20B,"("*** ")",4ZDB,
         "("RESPONDENTS HAVE BEEN READ    ***")"")",
         IN, RESP);
    EXIT;
NEXTPROB: OUTPUT (OC, "("6/,"("END OF JORESKOG")",*")")

"END" JORESKOG;
        "EOP"
```

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 succes.

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 indentifier `BIN LOW BOUND`.
The following error messages may appear:

Errornumber 0	(if no lower bound is found)
Errornumber 1	(if `X` is not an integer $\geqslant 0$, or `X` $>$ `N`)
Errornumber 2	(if `N` is not an integer >0)
Errornumber 3	(if `ALPHA` $\leqslant 0$ or `ALPHA` $\geqslant 1$)

PROCEDURES USED

`STATAL3 ERROR`	`STATAL 40100`
`BOUND`	`STATAL BOUND`
`ZEROINDER`	`NUMAL 34453`

LANGUAGE

Algol 60

METHOD AND PERFORMANCE

The lower bound is computed by finding a solution P of the equation

$$\sum_{K=X}^{N} \binom{N}{K} P^K (1-P)^{N-K} = \text{ALPHA}.$$

The precision is 10^{-10}.

EXAMPLE OF USE

Program:

```
"BEGIN"
    OUTPUT(61, "("3(Z.6D,/)")",
        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"
    STATAL3 ERROR("("BIN LOW BOUND")", 2, NN);
    T:= ENTIER(X);
    "IF" T < 0 "OR" T > N "OR" T < X "THEN"
    STATAL3 ERROR("("BIN LOW BOUND")", 1, X);
    "IF" ALPHA <= 0 "OR" ALPHA >= 1 "THEN"
    STATAL3 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"
```

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 indentifier `BIN UPP BOUND`.
The following error messages may appear:

Errornumber 1 (if `X` is not an integer $\geqslant 0$, or `X` $>$`N`)
Errornumber 2 (if `N` is not an integer > 0)
Errornumber 3 (if `ALPHA` $\leqslant 0$ or `ALPHA` $\geqslant 1$)
Errornumber 0 (if no upper bound is found)

PROCEDURES USED

`STATAL3 ERROR`	`STATAL 40100`
`BOUND`	`STATAL BOUND`
`ZEROINDER`	`NUMAL 34453`

LANGUAGE
Algol 60

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

$$\sum_{K=0}^{X} \binom{N}{X} P^K (1-P)^{N-K} = \text{ALPHA}.$$

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
 .511984
 .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"
```

TITLE: **Sample Des**

AUTHOR: A. Nonymous

INSTITUTE: Mathematical Centre

RECEIVED: 1981/1982

BRIEF DESCRIPTION

For a sample of independent observations X[LX],...,X[UX], the procedure computes the following 12 descriptive statistics: mean, variance, standard deviation, standard error of the mean, third and fourth central moments, skewness, kurtosis, minimun, maximum, median and range.

KEYWORDS

Discriptive 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[LX],...,X[UX];
LX:	<integer arithmetic expression>, smallest index of the sample;
UX:	<integer arithmetic expression>, largest index of the sample;
SORTED:	<boolean expression>, indicating whether the sample is sorted in non-decreasing order or not;
STATISTICS:	< array indentifier>, output parameter, array of dimension [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	STATAL 11020
STATAL3 ERROR	STATAL 40100

LANGUAGE

Algol 60

METHOD AND PERFORMANCE

The computation of the statistics is straightforward. (The sample variance is given by $N^{-1}\sum_{I}(X[I]-\text{mean})^2$, where N=UX−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 DES(X, 1, 10, SORTED, STAT);
    OUTPUT(61, "(""("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")", STAT)
"END"
```

Input:

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

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 DES(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 DES")", 3, U);

    "COMMENT" CONDITIONAL SORTING;
    "IF" "NOT" SORTED "THEN"
    "BEGIN" VECQ SORT(SAMPLE, L, U); SORTED:= "TRUE" "END";
    N:= U - L + 1; MINIMUM:= SAMPLE[L]; MAXIMUM:= SAMPLE[U];
    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" L PLUS U:= L + U; I:= L PLUS U // 2;
        MEDIAN:= "IF" L PLUS U = I * 2
                 "THEN" SAMPLE[I]
                 "ELSE" (SAMPLE[I] + SAMPLE[I + 1]) / 2;

        "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" Y:= (SAMPLE[I] - MEDIAN) / RANGE;
            S1:= S1 + Y; Y1:= Y * Y; S2:= S2 + Y1;
            Y1:= 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;
        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" 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"
```

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

Discriptive 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 indentifier>, vector containing the (different) observed values VALUE[L],...,VALUE[U];
NUMBER:	<integer array indentifier>, 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;
STATISTICS:	<array indentifier>, output parameter, array of dimension [1:14] which at exit contains the statistics.

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:
Errornumber 4 (if L > U)
Errornumber 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 $N^{-1}\sum_{I}$NUMBER[I](VALUE[I]−mean)^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(60, X);
    ININTARRAY(60, NUMBER);
    FREQTAB DES(X, NUMBER, 1, 10, SORTED, STAT);
    OUTPUT(61, "("""("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"
```

Input:

```
 .1  .2  .3  .4  .5  .6  .7  .8  .9  1.0
  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 2,
               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[P]:= V2[Q]; V2[Q]:= 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"
                "BEGIN" Y:= X; V1[P]:= X:= Z; V1[Q]:= Z:= Y;
                    Y:= V2[P]; V2[P]:= V2[Q]; V2[Q]:= 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" 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[IZ]:= Z;
                    Y:= V2[Q]; V2[Q]:= V2[IZ]; V2[IZ]:= Y
                "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" VEC2SORT;
                "IF" 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;
"IF" 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"
    STATAL3ERROR("("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[N OF STATISTICS]:= 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";
        "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 intems 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),...,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,...,N,

(ii) E(I) is 'less than' E(J) when there exists an index P, K≤P≤N, such that X(I,H)=X(J,H) for H=K,...,P−1 and X(I,P)<X(J,P),

(iii) the list E(L),...,E(U) is sorted ('lexicographically') if for every pair of indices (I,J), L≤I≤J≤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),...,E(U) in non-decreasing (lexicographical) order.
2. The type 'INDQSORT': the list E(L),...,E(U) remains unaltered but the contents of an array IND[LI:UI] is rearranged such that E(IND[LI]),...,E(IND[UI]) is in non-decreasing (lexicographical) order.
3. The type 'PERM': the list E(L),...,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),...,E(U) is delivered in an array RNK[L:U], while leaving E(L),...,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),...,E(U) these 20

procedures can also be divided according to five prefixes:

1. If the list is stored as `V[LV],...,V[UV]`, (a segment of) a one-dimensional array, the procedure identifiers contain the prefix '`VEC`'.
2. If the list is stored as `M[R,LC],...,M[R,UC]`, (a segment of) a row of a two-dimensional array, the procedure identifiers contain the prefix '`ROW`'.
3. If the list is stored as `M[LR,C],...,M[UR,C]`, (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 columms are to be reordered lexicographically, the procedure identifiers contain the prefix '`CMAT`'.

In addition there is one more sorting procedure, `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 `STATAL` procedures, the sorting and ranking procedures do not have any error messages. It is adviced 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,
Communications of the ACM, 13, (1970), 9, p.563-567.

[2] M.H. van Emden,
Algorithm 402, increasing the efficiency of quicksort,
Communications of the ACM, 13, (1970), 11, p.693.

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 indentifier ('`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[I]`, `V2[I]`) into non-decreasing order of the first components (i.e. `V1[LV]` $\leqslant$ `V1[LV+1]` $\leqslant$ .... $\leqslant$ `V1[UV]`).

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 indentifier>, 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` $\leqslant$ `LV` $\leqslant$ `UV` $\leqslant$ `U`.

DATA AND RESULTS
After a call of `VEC QSORT (V, LV, UV)`, `V[LV]` 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)`.

EXAMPLE OF USE

Program:

```
"BEGIN" "ARRAY" X[1..10];
    INARRAY(60, X);
    VEC QSORT(X, 4, 8);
    OUTPUT(61, "("10(+ZD2B)")", 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" 11020;
"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]:= 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:= 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[P]:= XX; V[IX]:= X "END";
```

```
            "IF" Q < IZ "AND" Z > ZZ "THEN"
            "BEGIN" V[Q]:= ZZ; 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 VECQSORT;
         "EOP"
```

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 indentifier>, 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 condition `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 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 on successive elements of a matrix row. The number of operations is of the order (UC−LC+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" "ARRAY" Z[1:3, 1:9];
    INARRAY(60, Z);
    ROW QSORT(Z, 2, 2, 9);
    OUTPUT(61, "("3(9(+ZD2B),/)")", Z)
"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  +25  +25  +25  +26  +27  +27  +28  +30
 +3  +19  +18  +17  -16  +15  +14  -13  +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" M[R, P]:= XX; M[R, IX]:= X "END";
            "IF" Q < IZ "AND" Z > ZZ "THEN"
            "BEGIN" M[R, 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"
```

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 indentifier>, a two-dimensional array `M[L1:U1, 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 $L2 \leqslant C \leqslant U2$;

`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 $L1 \leqslant LR \leqslant UR \leqslant U1$.

DATA AND RESULTS
After a call of `COL QSORT (M, C, LR, UR)`, `M[LR, C]` up to and including `M[UR, C]` are rearranged into non-decreasing order. All other 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 on successive elements of a matrix column. The number of operations is of the order (UR−LR+1)*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(+ZD2B),/)")", 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" COLQSORT(M, C, LR, UR);
"VALUE" C, LR, UR; "INTEGER" C, LR, UR; "ARRAY" M;
"BEGIN" "INTEGER" P, Q, IX, IZ; "REAL" X, XX, Y, ZZ, 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;
```

```
            M[Q, C]:= 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[P, C];
                "IF" X >= XX "THEN" "GOTO" RIGHT
            "END";
            P:= Q - 1; "GOTO" OUT;

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

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

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

Heading

```
"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 indentifier>, 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 $L1 \leqslant LR \leqslant UR \leqslant U1$;

`LC, UC`: <integer arithmetic expression>, smallest and largest index of the columns involved. `LC` and `UC` should satisfy the conditions $L2 \leqslant LC \leqslant UC \leqslant U2$.

DATA AND RESULTS

After a call of `RMAT QSORT (M, LR, UR, LC, UC)` the (segments of) rows (`M[LR, LC],...,M[LR, 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

`COL INDQSORT`	`STATAL 11041`
`RMAT INDQSORT`	`STATAL 11051`
`RMAT PERM`	`STATAL 11052`

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) + (UC−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(+ZD2B),/)")", 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
```

SOURCE TEXT

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

    "FOR" K:= LR "STEP" 1 "UNTIL" UR "DO" IND[K]:= K;
    RMATINDQSORT(M, IND, LR, UR, LC, UC);
    RMATPERM(IND, LR, UR, LC, UC, M);
"END" RMATQSORT;
        "EOP"
```

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 indentifier>, a two-dimensional array `M[L1: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 INDQSORT	STATAL 11031
CMAT INDQSORT	STATAL 11061
CMAT PERM	STATAL 11062

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*(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(+ZD2B),/)")", Z)
"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  +11  +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
```

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[K]:= K;
    CMATINDQSORT(M, IND, LR, UR, LC, UC);
    CMATPERM(IND, LR, UR, LC, UC, M);
"END" CMATQSORT;
        "EOP"
```

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 indentifier>, a one-dimensional array `V1[L1:U1]`;

`V2`: <array indentifier>, 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 $L1 \leqslant LV \leqslant UV \leqslant U1$ and $L2 \leqslant LV \leqslant UV \leqslant U2$.

DATA AND RESULTS
After a call of `VEC2 QSORT (V1, V2, LV, UV)`, the elements of `V1` and `V2` are rearranged simultaneously such that $V1[LV] \leqslant \cdots \leqslant V1[UV]$. 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 `(UV-LV+1) * LN(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" "ARRAY" X, Y[1:10];
    INARRAY(60, X); INARRAY(60, Y);
    VEC2 QSORT(X, Y, 4, 8);
    OUTPUT(61, "("2(10(+ZD2B),/)")", 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" VEC2QSORT(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" 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[P]:= V2[Q]; V2[Q]:= 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"
```

```
            "BEGIN" Y:= X; V1[P]:= X:= Z; V1[Q]:= Z:= Y;
                Y:= V2[P]; V2[P]:= V2[Q]; V2[Q]:= 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" 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[IZ]:= Z;
                Y:= V2[Q]; V2[Q]:= V2[IZ]; V2[IZ]:= Y
            "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" 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 'INDQSORT'), 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[LI] up to and including IND[UI] 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[LI] up to and including IND[UI] are permuted such that E(IND[LI]) ⩽ E(IND[LI+1]⩽ ··· ⩽E(IND[UI]).
The prefixes ('VEC', 'ROW', 'COL', 'RMAT' or 'CMAT') of the procedure indentifiers indicate the nature of the items to be processed.

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

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

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

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

CMAT INDQSORT 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 indentifier>, a one-dimensional array `V[L:U]`;

`IND`: <integer array indentifier>, a one-dimensional integer array `IND[LI:UI]` 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 $LI \leqslant LVI \leqslant UVI \leqslant UI$.

DATA AND RESULTS

Before calling the procedure, `IND[LVI]` 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[LVI]` up to and including `IND[UVI]` are permuted in such a way that $V[IND[LVI]] \leqslant \cdots \leqslant V[IND[UVI]]$. All other elements of `IND` and all elements of `V` 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 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)*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 INDQSORT(X, P, 4, 8);
    OUTPUT(61, "("2(10(+ZD2B),/)")", 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" VECINDQSORT(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
```

```
        "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:= V[N];
                "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;
                IND[P]:= H:= N; X:= Z; IND[Q]:= N:= K; Z:= Y
            "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[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" LVI:= L; UVI:= P - 1; L:= Q + 1 "END"
            "ELSE"
            "BEGIN" UVI:= U; LVI:= Q + 1; U:= P - 1 "END";
            "IF" UVI > LVI "THEN" VECINDSORT;
            "IF" U > L "THEN" "GOTO" PART
        "END" U - L > 1
    "END" VECINDSORT;

    "IF" UVI > LVI "THEN" VECINDSORT
"END" VECINDQSORT;
        "EOP"
```

TITLE: **Row 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 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

Heading

```
"PROCEDURE" ROW INDQSORT (M, R, IND, LCI, UCI);
"VALUE" R, LCI, UCI;
"INTEGER" R, LCI, UCI;
"ARRAY" M;
"INTEGER" "ARRAY" IND;
"CODE" 11031;
```

Formal parameters

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

`R`: <integer arithmetic expression>, the index of the row of `M` whose non-decreasing order has to be recorded in the array `IND`. `R` should satisfy the condition $L1 \leqslant R \leqslant U1$;

`IND`: <integer array indentifier>, a one-dimensional integer array `IND[LI:UI]` which contains in the positions `LCI` up to and including `UCI` distinct indices within the range (`L2`, `U2`);

`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 $LI \leqslant LCI \leqslant UCI \leqslant UI$.

DATA AND RESULTS

Before calling the procedure, `IND[LCI]` 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 `ROW INDQSORT (M, R, IND, LCI, UCI)`, `IND[LCI]` up to and including `IND[UCI]` are permuted in such a way that $M[R, IND[LCI]] \leqslant \cdots \leqslant M[R, IND[UCI]]$. 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 row `R` of the array `M`.
The number of operations is of the order `(UCI-LCI+1)*LN(UCI-LCI+1)`. No auxiliary arrays are declared. The recursion depth is at most `LN(UCI-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 INDQSORT(Z, 2, P, 2, 9);
    OUTPUT(61, "("3(9(+ZD2B),/),/,9(+ZD2B)")", 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
```

SOURCE TEXT

```
"CODE" 11031;
"PROCEDURE" ROWINDQSORT(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" ROWINDSORT;
    "BEGIN" "INTEGER" L, U; L:= LCI; U:= UCI;

 PART:  P:= L; H:= IND[P]; X:= M[R, H];
        Q:= U; N:= IND[Q]; Z:= M[R, N]; "IF" X > Z "THEN"
        "BEGIN" K:= H; Y:= X;
            IND[P]:= H:= N; X:= Z; IND[Q]:= N:= K; Z:= Y
        "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[R, 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:= M[R, N];
                "IF" Z <= ZZ "THEN" "GOTO" DIST
            "END";
            Q:= P; N:= H; Z:= X;
            P:= P - 1; H:= IND[P]; X:= M[R, H];

 DIST:      "IF" X > Z "THEN"
            "BEGIN" K:= H; Y:= X;
                IND[P]:= H:= N; X:= Z; IND[Q]:= N:= K; Z:= Y
            "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[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" LCI:= L; UCI:= P - 1; L:= Q + 1 "END"
            "ELSE"
            "BEGIN" UCI:= U; LCI:= Q + 1; U:= P - 1 "END";
            "IF" UCI > LCI "THEN" ROWINDSORT;
            "IF" U > L "THEN" "GOTO" PART
        "END" U - L > 1
    "END" ROWINDSORT;
```

```
    "IF" UCI > LCI "THEN" ROWINDSORT
"END" ROWINDQSORT;
        "EOP"
```

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 $L2 \leqslant C \leqslant U2$;

`IND`: <integer array indentifier>, a one-dimensional integer array `IND[LI:UI]` 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 $LI \leqslant LRI \leqslant URI \leqslant UI$.

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]]` $\leqslant \cdots \leqslant$ `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 `M`.
The number of operations is of the order `(URI-LRI+1) * LN (URI-LRI+1)`. No auxiliary arrays are declared. The recursion depth is at most `LN (URI-LRI+1) / LN(2)`.

EXAMPLE OF USE

Program:

```
"BEGIN" "INTEGER" I, J; "ARRAY" Z[1:9, 1:3];
    "INTEGER" "ARRAY" P[1:9];
    INARRAY(60, 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" 3 "DO"
        OUTPUT(61, "("+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" COLINDSORT;
    "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;
            IND[P]:= H:= N; X:= Z; IND[Q]:= N:= K; Z:= Y
        "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;
                IND[P]:= H:= N; X:= Z; IND[Q]:= N:= K; Z:= Y
            "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[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"
```

TITLE: **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 indentifier>, a two-dimensional array `M[L1:U1, L2:U2]`;

`IND`: <integer array identifier>, a one-dimensional integer array `IND[LI:UI]` 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 $LI \leqslant LRI \leqslant URI \leqslant UI$;

`LC, UC`: <integer arithmetic expression>, smallest and largest index of the columns involved. `LC` and `UC` should satisfy the condition $L2 \leqslant LC \leqslant UC \leqslant U2$.

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 `RMAT INDQSORT (M, IND, LRI, URI, LC, UC)`, `IND[LRI]` 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 $LRI \leqslant I \leqslant URI$. All other elements of `IND` and all elements of `M` remain unaltered.

PROCEDURES USED
COL INDQSORT STATAL 11041

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
According to the involved elements of column LC of the matrix, the contents of IND[LRI] up to and including IND[URI] are rearranged by COL INDQSORT. For any tie, (M[IND[LT], LC]= ··· =M[IND[UT], LC]),
COL INDQSORT rearranges IND[LT] up to and including IND[UT] with regard to those elements of column LC+1 whose row indices belong to the set (IND[LT], IND[LT+1],...,IND[UT]). 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")", Z[I, 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  +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        +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
```

SOURCE TEXT

```
"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"
```

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 INDQSORT (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 indentifier>, a two-dimensional array `M[L1:U1, L2:L2]`;

`IND`: <integer array indentifier>, a one-dimensional integer array `IND[LCI:UCI]` 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 statisfy the condition $L1 \leqslant LR \leqslant UR \leqslant U1$;

`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 $LI \leqslant LCI \leqslant UCI \leqslant UI$.

DATA AND RESULTS

Before calling the procedure, `IND[LCI]` 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 INDQSORT (M, IND, LR, UR, LCI, UCI)`, `IND[LCI]` up to and including `IND[UCI]` are permuted in such a way that they refer to the lexicographical order of the column-segments `(M[LR, IND[I]],...,M[UR, IND[I]])` with $LCI \leqslant I \leqslant UCI$. All other elements of `IND` and all elements of `M` remain unaltered.

PROCEDURES USED
ROW INDQSORT STATAL 11031

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
According to the involved elements of row LR of the matrix, the contents of IND[LCI] up to and including IND[UCI] are rearranged by ROW INDQSORT. For any tie, (M[LR, IND[LT]]= ··· =M[LR, IND[UT]], ROW INDQSORT rearranges IND[LT] up to and including IND[UT] with regard to those element of row LR+1 whose column indices belong to the set (IND[LT], IND [LT+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" Z[1:5, 1:9];
    "INTEGER" "ARRAY" P[1:9];
    INARRAY(60, Z);
    "FOR" I:=1 "STEP" 1 "UNTIL" 9 "DO" P[I]:=I;
    CMAT INDQSORT(Z, P, 2, 5, 2, 9);
    OUTPUT(61, "("5(9(+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  +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
```

SOURCE TEXT

```
"CODE" 11061;
"PROCEDURE" CMATINDQSORT(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 ;
        ROWINDQSORT(M, LR, IND, LCI, U) ;
        X:= M[ LR, 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 ;
                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" CMATINDQSORT;
        "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 IND (i.e. the orginal contents of E[IND[I]] is assigned to E[I], for every L ≤ I ≤ U).
In the case that those indices are obtained from E(L),...,E(U) by a call of a procedure of the 'INDQSORT'-type, the result is in non-decreasing (lexicographical) order.
The prefixes ('VEC', 'ROW', 'COL', 'RMAT' or 'CMAT') of the procedure indentifiers 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.

TITLE: **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 IND.

KEYWORDS

Rearrangement of vector elements via permutation of indices

CALLING SEQUENCE

Heading

```
"PROCEDURE" VEC PERM (IND, LV, UV, V);
"VALUE" LV, UV;
"INTEGER" LV, UV;
"ARRAY" V;
"INTEGER" "ARRAY" IND;
"CODE" 11022;
```

Formal parameters

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

LV, UV: <integer arithmetic expression>, smallest and largest index of the segment of V which has to be permuted and of the segment of IND which has to be used. LV and UV should satisfy the conditions L ⩽ LV ⩽ UV ⩽ U and LI ⩽ LV ⩽ UV ⩽ UI;

V: <array indentifier>, a one-dimensional array V[L:U].

DATA AND RESULTS

Before calling the procedure, IND[LV] up to and including IND[UV] should contain user supplied indices which are distinct and within the range LV up to and including UV. After a call of VEC PERM(IND, LV, UV, V), V[LV] up to and including V[UV] are rearranged according to the permutation of indices contained in (the segment of) IND. All other elements of V remain unaltered. VEC 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 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−LV+1). One auxiliary boolean array of (UV−LV+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(+ZD2B),/)")", 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:= V[K];
```

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

"END" VECPERM;
         "EOP"
```

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 segement 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 indentifier>, a one-dimensional integer array `IND[LI:UI]` 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 sement 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 `LI ≤ LC ≤ UC ≤UI`;

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

DATA AND RESULTS

Before calling the procedure, `IND[LC]` up to and including `IND[UC]` 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, LC]` up to and including `M[R, 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`.

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(+ZD2B),/),/,9(+ZD2B)")", 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   +5   +4
```

Output:

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

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

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 indentifier>, a one-dimensional integer array IND[LI:UI] 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 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 COL PERM(IND, C, LR, UR, M), M[LR, C] 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, "("5B,+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  +25  +12        +4
```

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 ^= 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"
```

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:UI] 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 $L1 \leqslant LR \leqslant UR \leqslant U1$ and $LI \leqslant LR \leqslant UR \leqslant UI$;

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

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(-segment)s 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, Z);
    "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"
```

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 indentifier>, a one-dimensional integer array IND[LI:UI] 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 $L1 \leqslant LR \leqslant UR \leqslant U1$;

LC, UC: <integer arithmetic expression>, smallest and largest index indicating the columns which have to be permuted, and of the segment of IND wich has to be used. LC and UC should satisfy the conditions $L2 \leqslant LC \leqslant UC \leqslant U2$ and $LI \leqslant LC \leqslant UC \leqslant UI$;

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

DATA AND RESULTS

Before calling the procedure, IND[LC] up to and including IND[UC] should contain 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)s (M[LR, IND[LC]], ...,M[UR, IND[LC]]) up to and including (M[LR, IND[UC]], ...,M[UR, IND[UC]]) are rearranged according to the permutation 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 `IND`. 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−LC+1) * (UR−LR+1)`. One auxiliary boolean array of `(UC−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(60, Z); ININTARRAY(60, P);
    CMAT PERM(P, 2, 5, 2, 9, Z);
    OUTPUT(61, "("5(9(+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

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

Output:

```
 +1   +2   +3   +4   +5   +6   +7   +8   +9
 +2  +11  +11  +11  +11  +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
```

SOURCE TEXT

```
"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, K]:= COL[H];
        TODO[K]:= "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. Futhermore, the same permutation of indices as in the type 'INDQSORT' 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+{the number of elements E(J) with E(J)<E(K)}, and T (K)={the number of elements E(H) with E(H) ≤ 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 ≤ RNK[K] ≤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 indentifiers 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, RKN, 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[LI:UI]` 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[RL:RU]` 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.

DATA AND RESULTS

After a call of `VEC RANKTIE (V, LV, UV, IND, RS, T2, T3)`, `RS[I]` 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[LV]` up to and including `IND[UV]` are such that `V[IND[LV]]`$\leq \cdots \leq$`V[IND[UV]]`. All other elements of `IND` and `RS` and all elements of `V` remain unaltered.

PROCEDURES USED

`VEC INDQSORT` `STATAL 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, IND, V, T2, T3)`. The number of operations is of the order `(UV-LV+1) * (1+LN(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(+ZD2B),/,10(-ZD.D),/,10(+ZD2B),/,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, RNK, TIES2, TIES3);
"VALUE" LV, UV; "INTEGER" LV, UV; "REAL" TIES2, TIES3;
"INTEGER" "ARRAY" IND; "REAL" "ARRAY" V, RNK;
"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"
            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 + UV) / 2 - LV + 1;
    "FOR" J:= P "STEP" 1 "UNTIL" UV "DO" RNK[IND[J]]:= Z;
    F:= UV - P + 1; F2:= F * F; TIES2:= TIES2 + F2;
    TIES3:= TIES3 + F2 * F
"END" VECRANKTIE;
        "EOP"
```

TITLE: **Row Ranktie**

AUTHOR: A.C. IJsselstein

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 indentifier>, 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 $L1 \leqslant R \leqslant U1$;

`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 $LI \leqslant LC \leqslant UC \leqslant UI$ and $L2 \leqslant LC \leqslant UC \leqslant U2$;

`IND`: <integer array identifier>, output parameter, a one-dimensional integer array `IND[LI:UI]` 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 indentifier>, output parameter, a one-dimensional real array `RNK[RL:RU]` with $RL \leqslant LC \leqslant UC \leqslant RU$, 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)`, `RS[I]` has as value the (average) rank of `M[R, I]`, `I=LC,...,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 INDQSORT` STATAL 11031

LANGUAGE

Algol 60

METHOD AND PERFORMANCE

The permutation of indices which indicates the non-decreasing order is generated by `ROW INDQSORT`. 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(Z, 2, 2, 9, P, RN, T2, T3);
    OUTPUT(61,
       "("3(9(+ZD2B),/),/,9(-ZD.D),/,9(+ZD2B),/,2(-5ZD)")",
        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
```

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;
    ROWINDQSORT(M, R, IND, LC, UC); TIES2:= TIES3:= 0;
    X:= M[R, IND[LC]]; 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) / 2 - LC + 1;
            "FOR" J:= P "STEP" 1 "UNTIL" Q "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 + UC) / 2 - LC + 1;
    "FOR" J:= P "STEP" 1 "UNTIL" UC "DO" RNK[IND[J]]:= Z;
    F:= UC - P + 1; F2:= F * F; TIES2:= TIES2 + F2;
    TIES3:= TIES3 + F2 * F
"END" ROWRANKTIE;
        "EOP"
```

TITLE: **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[L1: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 `LI ≤ LR ≤ UR ≤ UI` and `L1 ≤ LR ≤ UR ≤U1`;

`IND`: <integer array indentifier>, output parameter, a one-dimensional integer array `IND[LI:UI]` 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[RL: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.

DATA AND RESULTS
After a call of `COL RANKTIE (M, C, LR, UR, IND, RS, T2, T3)`, `RS[I]` has as value the (average) rank of `M[I, C]`, `I=LR,...,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], C]` $\leqslant$`M[IND[LR+1], C]` $\leqslant \cdots \leqslant$ `M[IND[UR], C]`. All other elements of `IND` and `RS` and all elements of `M` remain unaltered.

PROCEDURES USED
`COL INDQSORT` `STATAL 11041`

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
The permutation of indices which indicates the non-decreasing order is generated via `COL INDQSORT`. Using this information the ranks and the sizes of the ties are computed. The number of operations is of the order `(UR-LR+1) * (1+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" "INTEGER" I, J; "REAL" T2, T3;
    "INTEGER" "ARRAY" P[1:9]; "ARRAY" Z[1:9, 1:3], 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, 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, "("5B, -ZD.D")", RN[I]);
        OUTPUT(61, "("5B, +ZD2B, /")", P[I])
    "END";
    OUTPUT(61, "("/, -5ZD, -5ZD")", T2, T3)
"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        0.0      +0
+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;
    COLINDQSORT(M, C, IND, LR, UR); TIES2:= TIES3:= 0;
    X:= M[IND[LR], C]; P:= LR;

    "FOR" H:= LR + 1 "STEP" 1 "UNTIL" UR "DO"
    "BEGIN" Y:= M[IND[H], C]; "IF" X < Y "THEN"
        "BEGIN" Q:= H - 1; Z:= (P + Q) / 2 - LR + 1;
            "FOR" J:= P "STEP" 1 "UNTIL" Q "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) / 2 - LR + 1;
    "FOR" J:= P "STEP" 1 "UNTIL" UR "DO" RNK[IND[J]]:= Z;
    F:= UR - P + 1; F2:= F * F; TIES2:= TIES2 + F2;
    TIES3:= TIES3 + F2 * F
"END" COLRANKTIE;
        "EOP"
```

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

Heading

```
"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 indentifier>, a two-dimensional array `M[L1: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 ≤ UI` 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 indentifier>, output parameter, a one-dimensional integer array `IND[LI:UI]` 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 indentifier>, 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[I], LC],..., M[IND[I], UC])` with $LR \leqslant I \leqslant UR$. All other elements of `IND` and `RS` and all elements of `M` remain unaltered.

PROCEDURES USED

`COL INDQSORT` `STATAL 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.D")", RN[I]);
        OUTPUT(61, "("5B, +ZD2B, /")", P[I])
    "END";
    OUTPUT(61, "("/, -5ZD, -5ZD")", 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"
```

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 indentifier>, 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 `LI ⩽ LC ⩽ UC ⩽ UI` and `L2 ⩽ LC ⩽ UC ⩽U2`;

`IND`: <integer array identifier>, output parameter, a one-dimensional integer array `IND[LI:UI]` 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 indentifier>, output parameter, a one-dimensional real array `RNK[RL:RU]` 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[I] has as value the (average) rank of the column-segment (M[LR, I],...,M[UR, I]), 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-segements (M[LR, IND[I]],..., M[UR, IND[I]]) with LC ⩽ I ⩽ UC. All other elements of IND and RS and all elements of M remain unaltered.

PROCEDURES USED

ROW INDQSORT 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 INDQSORT (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, "("5(9(+ZD2B),/),/,9(-ZD.D),/,9(+ZD2B),/,
        2(-5ZD)")", 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[H]]; "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,...,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 $\prod_{I=1}^{N} F(I)!$ times in the set of all possible arrangements of the M items.
The procedure PERMUTATION assumes that all items in the set are represented by different integers (thus F(I)=1 for I=1,...,N), while the procedure GENERAL PERM allows identical values for items (thus F(I) ⩾ 0, I=1,...,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:

1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

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:

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
2	3	2	1
3	1	2	2
3	2	1	2
3	2	2	1

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,...,N); while the procedure GENERAL COMB allows identical values of items (thus F(I) ⩾ 0 for I=1,...,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 subseqence. This number of arrangements equals

$$R! \prod_{I=1}^{N} \left[\begin{matrix} F(I) \\ G(I) \end{matrix} \right],$$

where G(I) is the number of items in the particular subsequence which have value I.
For example, let N=5, and F(1)=···=F(5)=1, thus the distinct non-decreasing subsequences of size R=3 are:

1	2	3
1	2	4
1	2	5
1	3	4
1	3	5
1	4	5
2	3	4
2	3	5
2	4	5
3	4	5

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):

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

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,...,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.

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,...,N`. The value `I` appears exactly once (`I=1,2,...,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` $\leqslant$ `1` and `U` $\geqslant$ `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:
Errornumber 2 (if `N` is not an integer $\geqslant$ `1`)

PROCEDURES USED

`STATAL3 ERROR` `STATAL 40100`

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(D2B),/")", 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" ENTIER(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:
    "IF" A[W] < A[W - 1] "THEN"
    "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"
        "BEGIN" A[W - 1]:= A[I]; A[I]:= U;
```

```
            "GOTO" NEXT STEP
        "END"
    "END";

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

    FIRST:= "TRUE";
    "FOR" I:= 2 "STEP" 1 "UNTIL" N "DO"
    "IF" A[I - 1] < A[I] "THEN"
    "BEGIN" FIRST:= "FALSE"; "GOTO" EXIT PROC "END";

EXIT PROC:
"END" PERMUTATION;
          "EOP"
```

TITLE: **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;
"BOOLEAN" FIRST;
"CODE" 40502;
```

Formal parameters

V: <integer array indentifier>, output parameter, one-dimensional integer array V[L:U], where L ⩽ 1 and U ⩾ M, which at exit contains the result of the permutation;

M: <arithmetic expression>, number of items, M=∑F[I] where the sum is over I=1,...,N;

F: <integer array indentifier>, 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:
Errornumber 3 (if one of the F[I]'s is not an integer $\geqslant 0$, or M is not equal to $\sum$F[I])
Errornumber 4 (if N is not an integer $\geqslant 1$)

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[1]:= 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, "("4(D2B),5ZD,/")", 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
```

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:= F[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:= F[I];
            "FOR" J:= 1 "STEP" 1 "UNTIL" FI "DO"
            "BEGIN" TEL:= TEL + 1;
                A[TEL]:= I; NUM:= NUM * J
            "END";
        "END"; "GOTO" CHECK LAST
    "END";
    W:= M;

CHECK ORDER:
    "IF" W = 1 "THEN" "GOTO" CHECK LAST "ELSE"
    "IF" A[W] <= A[W - 1] "THEN"
    "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"
        "BEGIN" A[W - 1]:= A[I]; A[I]:= U;
            "GOTO" NEXT STEP
        "END"
    "END";

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

CHECK LAST:
    FIRST:= "TRUE";
```

```
    "FOR" I:= 2 "STEP" 1 "UNTIL" M "DO"
    "IF" A[I - 1] < A[I] "THEN"
    "BEGIN" FIRST:= "FALSE"; "GOTO" EXIT PROC "END";

EXIT PROC:
"END" GENERAL PERM;
        "EOP"
```

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,...,N. The value I appears exactly once in the sequence (I= 1, 2,...,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[L: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:
Errornumber 2 (if R is not an integer ⩾ 1, or R > N)
Errornumber 3 (if N is not an integer ⩾ 1)

PROCEDURES USED
STATAL3 ERROR STATAL 40100

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(D2B),5ZD,/")", 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"
    STATAL3 ERROR("("COMBINATION")", 3, N) "ELSE"
    "IF" R < 1 "OR" R > N "OR" ENTIER(R) < R "THEN"
    STATAL3 ERROR("("COMBINATION")", 2, R);
```

```
    "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"
    "IF" A[R] < N "THEN" A[R]:= A[R] + 1 "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"
            A[J]:= A[I] + J - I;
            "GOTO" CHECK LAST;
        "END"
    "END";
    CHECK LAST:
    "IF" A[1] = N - R + 1 "THEN" FIRST:= "TRUE"
"END" COMBINATION;
        "EOP"
```

TITLE: **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,...,N. The value I appears F(I) times in the sequence I= 1, 2,...,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 indentifier>, 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 indentifier>, 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 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:

Errornumber 2 (if R is not an integer ⩾ 1, or R > ΣF[I])

Errornumber 3 (if one of the F[I]'s is not an integer ⩾ 0)

Errornumber 4 (if N is not an integer ⩾ 1)

PROCEDURES USED
STATAL3 ERROR STATAL 40100

LANGUAGE
Algol 60

METHOD AND PERFORMANCE
The subsequences of size 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 V.

EXAMPLE OF USE

Program:

```
"BEGIN"  "INTEGER" "ARRAY" W, FW[1:3];
    "INTEGER" I, NM; "BOOLEAN" F;
    F:="TRUE";
    FW[1]:= 2; FW[2]:= 3; FW[3]:= 2;
    "FOR" I:= 1, I + 1 "WHILE" "NOT" F "DO"
    "BEGIN" GENERAL COMB(W, 3, FW, 3, NM, F);
        OUTPUT(61, "("3(D2B),5ZD,/")", W, NM)
    "END"
"END"
```

Output:

```
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
```

SOURCE TEXT

```
"CODE" 40504;
"PROCEDURE" GENERAL COMB(A, R, F, N, NUM, FIRST);
"VALUE" R, N;
"INTEGER" "ARRAY" A, F; "REAL" R, N; "INTEGER" NUM;
"BOOLEAN" FIRST;
"BEGIN" "INTEGER" I, J, STAP, REST, PTR, IND, NEW, UP;
    "REAL" FI, S; "INTEGER" "ARRAY" FR[1:N];

    "IF" N < 1 "OR" ENTIER(N) < N "THEN"
    STATAL3 ERROR("("GENERAL COMB")", 4, N);
    S:= 0; "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
```

```
    "BEGIN" FI:= F[I];
        "IF" FI < 0 "OR" ENTIER(FI) < FI "THEN"
        STATAL3 ERROR("("GENERAL COMB")", 3, FI) "ELSE"
        S:= S + FI
    "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" FR[I]:= 0;
        "FOR" I:= 1 "STEP" 1 "UNTIL" R "DO"
        FR[A[I]]:= FR[A[I]] + 1;
    "END";

L1: PTR:= PTR - STAP; IND:= A[PTR]; STAP:= FR[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" FR[NEW] < F[NEW] "THEN" "GOTO" L2; "GOTO" L1;

L2: PTR:= REST:= PTR - 1;

L3: STAP:= F[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[A[I]]:= FR[A[I]] + 1 "END";
    "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
    "BEGIN" UP:= F[I]; FI:= FR[I]; "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

1 Vacantiecursus 1984: *Hewet - plus wiskunde.* 1984.

2 E.M. de Jager, H.G.J. Pijls (eds.). *Proceedings Seminar 1981-1982. Mathematical structures in field theories.* 1984.

3 W.C.M. Kallenberg, et al. *Testing statistical hypotheses: worked solutions.* 1984.

4 J.G. Verwer (ed.). *Colloquium topics in applied numerical analysis, volume 1.* 1984.

5 J.G. Verwer (ed.). *Colloquium topics in applied numerical analysis, volume 2.* 1984.

6 P.J.M. Bongaarts, J.N. Buur, E.A. de Kerf, R. Martini, H.G.J. Pijls, J.W. de Roever. *Proceedings Seminar 1982-1983. Mathematical structures in field theories.* 1985.

7 Vacantiecurus 1985: *Variatierekening.* 1985.

8 G.M. Tuynman. *Proceedings Seminar 1983-1985. Mathematical structures in field theories, Vol.1 Geometric quantization.* 1985.

9 J. van Leeuwen, J.K. Lenstra (eds.). *Parallel computers and computations.* 1985.

10 Vacantiecursus 1986: *Matrices.* 1986.

11 P.W.H. Lemmens. *Discrete wiskunde: tellen, grafen, spelen en codes.* 1986.

12 J. van de Lune. *An introduction to Tauberian theory: from Tauber to Wiener.* 1986.

13 G.M. Tuynman, M.J. Bergvelt, A.P.E. ten Kroode. *Proceedings Seminar 1983-1985. Mathematical structures in field theories, Vol.2.* 1987.

14 Vacantiecursus 1987: *De personal computer en de wiskunde op school.* 1987.

15 Vacantiecursus 1983: *Complexe getallen.* 1987.

16 P.J.M. Bongaarts, E.A. de Kerf, P.H.M. Kersten. *Proceedings Seminar 1984-1986. Mathematical structures in field theories, Vol.1.* 1988.

17 F. den Hollander, H. Maassen (eds.). *Mark Kac seminar on probability and physics. Syllabus 1985-1987.* 1988.

18 Vacantiecursus 1988. *Differentierekening.* 1988.

19 R. de Bruin, C.G. van der Laan, J.R. Luyten, H.F. Vogt. *Publiceren met LATEX.* 1988.

20 R. van der Horst, R.D. Gill (eds.). *STATAL: statistical procedures in Algol 60, part 1.* 1988.

21 R. van der Horst, R.D. Gill (eds.). *STATAL: statistical procedures in Algol 60, part 2.* 1988.

22 R. van der Horst, R.D. Gill (eds.). *STATAL: statistical procedures in Algol 60, part 3.* 1988.

MC SYLLABI

1.1 F. Göbel, J. van de Lune. *Leergang besliskunde, deel 1: wiskundige basiskennis.* 1965.

1.2 J. Hemelrijk, J. Kriens. *Leergang besliskunde, deel 2: kansberekening.* 1965.

1.3 J. Hemelrijk, J. Kriens. *Leergang besliskunde, deel 3: statistiek.* 1966.

1.4 G. de Leve, W. Molenaar. *Leergang besliskunde, deel 4: Markovketens en wachttijden.* 1966.

1.5 J. Kriens, G. de Leve. *Leergang besliskunde, deel 5: inleiding tot de mathematische besliskunde.* 1966.

1.6a B. Dorhout, J. Kriens. *Leergang besliskunde, deel 6a: wiskundige programmering 1.* 1968.

1.6b B. Dorhout, J. Kriens, J.Th. van Lieshout. *Leergang besliskunde, deel 6b: wiskundige programmering 2.* 1977.

1.7a G. de Leve. *Leergang besliskunde, deel 7a: dynamische programmering 1.* 1968.

1.7b G. de Leve, H.C. Tijms. *Leergang besliskunde, deel 7b: dynamische programmering 2.* 1970.

1.7c G. de Leve, H.C. Tijms. *Leergang besliskunde, deel 7c: dynamische programmering 3.* 1971.

1.8 J. Kriens, F. Göbel, W. Molenaar. *Leergang besliskunde, deel 8: minimaxmethode, netwerkplanning, simulatie.* 1968.

2.1 G.J.R. Förch, P.J. van der Houwen, R.P. van de Riet. *Colloquium stabiliteit van differentieschema's, deel 1.* 1967.

2.2 L. Dekker, T.J. Dekker, P.J. van der Houwen, M.N. Spijker. *Colloquium stabiliteit van differentieschema's, deel 2.* 1968.

3.1 H.A. Lauwerier. *Randwaardeproblemen, deel 1.* 1967.

3.2 H.A. Lauwerier. *Randwaardeproblemen, deel 2.* 1968.

3.3 H.A. Lauwerier. *Randwaardeproblemen, deel 3.* 1968.

4 H.A. Lauwerier. *Representaties van groepen.* 1968.

5 J.H. van Lint, J.J. Seidel, P.C. Baayen. *Colloquium discrete wiskunde.* 1968.

6 K.K. Koksma. *Cursus ALGOL 60.* 1969.

7.1 *Colloquium moderne rekenmachines, deel 1.* 1969.

7.2 *Colloquium moderne rekenmachines, deel 2.* 1969.

8 H. Bavinck, J. Grasman. *Relaxatietrillingen.* 1969.

9.1 T.M.T. Coolen, G.J.R. Förch, E.M. de Jager, H.G.J. Pijls. *Colloquium elliptische differentiaalvergelijkingen, deel 1.* 1970.

9.2 W.P. van den Brink, T.M.T. Coolen, B. Dijkhuis, P.P.N. de Groen, P.J. van der Houwen, E.M. de Jager, N.M. Temme, R.J. de Vogelaere. *Colloquium elliptische differentiaalvergelijkingen, deel 2.* 1970.

10 J. Fabius, W.R. van Zwet. *Grondbegrippen van de waarschijnlijkheidsrekening.* 1970.

11 H. Bart, M.A. Kaashoek. H.G.J. Pijls, W.J. de Schipper, J. de Vries. *Colloquium halfalgebra's en positieve operatoren.* 1971.

12 T.J. Dekker. *Numerieke algebra.* 1971.

13 F.E.J. Kruseman Aretz. *Programmeren voor rekenautomaten; de MC ALGOL 60 vertaler voor de EL X8.* 1971.

14 H. Bavinck, W. Gautschi, G.M. Willems. *Colloquium approximatietheorie.* 1971.

15.1 T.J. Dekker, P.W. Hemker, P.J. van der Houwen. *Colloquium stijve differentiaalvergelijkingen, deel 1.* 1972.

15.2 P.A. Beentjes, K. Dekker, H.C. Hemker, S.P.N. van Kampen, G.M. Willems. *Colloquium stijve differentiaalvergelijkingen, deel 2.* 1973.

15.3 P.A. Beentjes, K. Dekker, P.W. Hemker, M. van Veldhuizen. *Colloquium stijve differentiaalvergelijkingen, deel 3.* 1975.

16.1 L. Geurts. *Cursus programmeren, deel 1: de elementen van het programmeren.* 1973.

16.2 L. Geurts. *Cursus programmeren, deel 2: de programmeertaal ALGOL 60.* 1973.

17.1 P.S. Stobbe. *Lineaire algebra, deel 1.* 1973.

17.2 P.S. Stobbe. *Lineaire algebra, deel 2.* 1973.

17.3 N.M. Temme. *Lineaire algebra, deel 3.* 1976.

18 F. van der Blij, H. Freudenthal, J.J. de Iongh, J.J. Seidel, A. van Wijngaarden. *Een kwart eeuw wiskunde 1946-1971, syllabus van de vakantiecursus 1971.* 1973.

19 A. Hordijk, R. Potharst, J.Th. Runnenburg. *Optimaal stoppen van Markovketens.* 1973.

20 T.M.T. Coolen, P.W. Hemker, P.J. van der Houwen, E. Slagt. *ALGOL 60 procedures voor begin- en randwaardeproblemen.* 1976.

21 J.W. de Bakker (red.). *Colloquium programmacorrectheid.* 1975.

22 R. Helmers, J. Oosterhoff, F.H. Ruymgaart, M.C.A. van Zuylen. *Asymptotische methoden in de toetsingstheorie; toepassingen van naburigheid.* 1976.

23.1 J.W. de Roever (red.). *Colloquium onderwerpen uit de biomathematica, deel 1.* 1976.

23.2 J.W. de Roever (red.). *Colloquium onderwerpen uit de biomathematica, deel 2.* 1977.

24.1 P.J. van der Houwen. *Numerieke integratie van differentiaalvergelijkingen, deel 1: eenstapsmethoden.* 1974.

25 *Colloquium structuur van programmeertalen.* 1976.

26.1 N.M. Temme (ed.). *Nonlinear analysis, volume 1.* 1976.

26.2 N.M. Temme (ed.). *Nonlinear analysis, volume 2.* 1976.

27 M. Bakker, P.W. Hemker, P.J. van der Houwen, S.J. Polak, M. van Veldhuizen. *Colloquium discretiseringsmethoden.* 1976.

28 O. Diekmann, N.M. Temme (eds.). *Nonlinear diffusion problems.* 1976.

29.1 J.C.P. Bus (red.). *Colloquium numerieke programmatuur, deel 1A, deel 1B.* 1976.

29.2 H.J.J. te Riele (red.). *Colloquium numerieke programmatuur, deel 2.* 1977.

30 J. Heering, P. Klint (red.). *Colloquium programmeeromgevingen.* 1983.

31 J.H. van Lint (red.). *Inleiding in de coderingstheorie.* 1976.

32 L. Geurts (red.). *Colloquium bedrijfsystemen.* 1976.

33 P.J. van der Houwen. *Berekening van waterstanden in zeeën en rivieren.* 1977.

34 J. Hemelrijk. *Oriënterende cursus mathematische statistiek.* 1977.

35 P.J.W. ten Hagen (red.). *Colloquium computer graphics.* 1978.

36 J.M. Aarts, J. de Vries. *Colloquium topologische dynamische systemen.* 1977.

37 J.C. van Vliet (red.). *Colloquium capita datastructuren.* 1978.

38.1 T.H. Koornwinder (ed.). *Representations of locally compact groups with applications, part I.* 1979.

38.2 T.H. Koornwinder (ed.). *Representations of locally compact groups with applications, part II.* 1979.

39 O.J. Vrieze, G.L. Wanrooy. *Colloquium stochastische spelen.* 1978.

40 J. van Tiel. *Convexe analyse.* 1979.

41 H.J.J. te Riele (ed.). *Colloquium numerical treatment of integral equations.* 1979.

42 J.C. van Vliet (red.). *Colloquium capita implementatie van programmeertalen.* 1980.

43 A.M. Cohen, H.A. Wilbrink. *Eindige groepen (een inleidende cursus).* 1980.

44 J.G. Verwer (ed.). *Colloquium numerical solution of partial differential equations.* 1980.

45 P. Klint (red.). *Colloquium hogere programmeertalen en computerarchitectuur.* 1980.

46.1 P.M.G. Apers (red.). *Colloquium databankorganisatie, deel 1.* 1981.

46.2 P.G.M. Apers (red.). *Colloquium databankorganisatie, deel 2.* 1981.

47.1 P.W. Hemker (ed.). *NUMAL, numerical procedures in ALGOL 60: general information and indices.* 1981.

47.2 P.W. Hemker (ed.). *NUMAL, numerical procedures in ALGOL 60, vol. 1: elementary procedures; vol. 2: algebraic evaluations.* 1981.

47.3 P.W. Hemker (ed.). *NUMAL, numerical procedures in ALGOL 60, vol. 3A: linear algebra, part I.* 1981.

47.4 P.W. Hemker (ed.). *NUMAL, numerical procedures in ALGOL 60, vol. 3B: linear algebra, part II.* 1981.

47.5 P.W. Hemker (ed.). *NUMAL, numerical procedures in ALGOL 60, vol. 4: analytical evaluations; vol. 5A: analytical problems, part I.* 1981.

47.6 P.W. Hemker (ed.). *NUMAL, numerical procedures in ALGOL 60, vol. 5B: analytical problems, part II.* 1981.

47.7 P.W. Hemker (ed.). *NUMAL, numerical procedures in ALGOL 60, vol. 6: special functions and constants; vol. 7: interpolation and approximation.* 1981.

48.1 P.M.B. Vitányi, J. van Leeuwen. P. van Emde Boas (red.). *Colloquium complexiteit en algoritmen, deel 1.* 1982.

48.2 P.M.B. Vitányi, J. van Leeuwen, P. van Emde Boas (red.). *Colloquium complexiteit en algoritmen, deel 2.* 1982.

49 T.H. Koornwinder (ed.). *The structure of real semisimple Lie groups.* 1982.

50 H. Nijmeijer. *Inleiding systeemtheorie.* 1982.

51 P.J. Hoogendoorn (red.). *Cursus cryptografie.* 1983.