Algorithms

J. G. HERRIOT, Editor

ALGORITHM 320
HARMONIC ANALYSIS FOR SYMMETRICALLY DISTRIBUTED DATA [C6]

D. B. HUNTER (Recd. 1 June 1965, 4 Jan. 1966, and 26 June 1967)
Department of Mathematics, University of Bradford, Yorkshire, England

KEY WORDS AND PHRASES: harmonic analysis, cosine series, sine series, function approximation, curve fitting, trigonometric series
CR CATEGORIES: 5.13

procedure *trigfit* (*index*, *n*, *m*, *h*, *e*, *x*, *f*, *mt*, *a*);
value *index*, *n*, *m*, *h*, *e*; **integer** *index*, *n*, *m*, *mt*; **real** *h*, *e*; **array** *x*, *f*, *a*;
comment Approximates a function y of x by a half-range cosine or sine series of period $2h$ from values specified at discrete points, not necessarily equally-spaced, in the range $(0, h)$. The input parameters are:

index—if $index = 0$, a cosine series is fitted, if $index = 1$, a sine series. No other value is permitted.
n—number of function-values given.
m—order of the highest harmonic required.
h—half-period of the fitted series.
e—used to terminate the process if rounding errors start to accumulate excessively (see note below).
x—the given values of x are stored on $x[1], x[2], \cdots, x[n]$.
f—the value of y corresponding to $x = x[i]$ is stored on $f[i]$ $(i=1, 2, \cdots, n)$.

The procedure then calculates the coefficients $a[r]$ in the approximation

$$S(x) = \begin{cases} \frac{1}{2}a[0] + \sum_{r=1}^{mt} a[r] \cos (r\pi x/h) & \text{if } index = 0, \\ \sum_{r=1}^{mt} a[r] \sin (r\pi x/h) & \text{if } index = 1. \end{cases}$$

Here normally $mt = m$, but provision is included to calculate fewer harmonics if rounding errors begin to accumulate excessively (see note below).

Method of calculation. The coefficients $a[r]$ are calculated so as to minimize the sum

$$\sum_{i=1}^{n} w_i(f[i]-S(x[i]))^2, \quad w_i = \begin{cases} \frac{1}{2} \text{ if } x[i] = 0 \text{ or } h, \\ 1 \text{ otherwise.} \end{cases}$$

The method used is similar to that of [1]. First $S(x)$ is expanded in the form

$$S(x) = \sum_{i=index}^{mt} b_i p_i(x)$$

where

$$p_i(x) = \begin{cases} \frac{1}{2}a_{i0} + \sum_{j=1}^{i} a_{ij} \cos (j\pi x/h) & \text{if } index = 0, \\ \sum_{j=1}^{i} a_{ij} \sin (j\pi x/h) & \text{if } index = 1. \end{cases}$$

Then

$$a[r] = \sum_{i=r}^{mt} b_i a_{ir} .$$

The polynomials $p_j(x)$ are chosen so as to be orthogonal w.r.t. summation over $x = x[i]$, with weights w_i. This implies that

$$b_i = \sum_{j=1}^{n} w_j f[j] p_i(x[j]) / \sum_{j=1}^{n} w_j[p_i(x_j)]^2.$$

The $p_i(x)$ are generated by a recurrence relation

$$p_{i+1}(x) = (2 \cos (\pi x/h) - \alpha_i) p_i(x) - \beta_i p_{i-1}(x)$$

where

$$\alpha_i = \frac{2 \sum_{j=1}^{n} w_j \cos (\pi x[j]/h) \cdot [p_i(x[j])]^2}{\sum_{j=1}^{n} w_j[p_i(x[j])]^2} \quad (i \geq index),$$

$$\beta_i = \frac{\sum_{j=1}^{n} w_j[p_i(x[j])]^2}{\sum_{j=1}^{n} w_j[p_{i-1}(x[j])]^2} \quad (i > index).$$

The initial forms are

$p_0(x) = \frac{1}{2}$ if $index = 0$

or $p_1(x) = \sin (\pi x/h)$ if $index = 1$.

Thus if the $x[i]$ are equally spaced, i.e. if $x[i] = (i-1)h/(n-1)$, it follows that

$p_k(x) = \cos (k\pi x/h)$ or $\sin (k\pi x/h)$ according as $index = 0$ or 1.

The values of the $p_i(x)$ are calculated by the method of [2].

Note. If the $x[i]$ are verp irregular in their distribution serious rounding errors may accumulate, and it is recommended that the points be as nearly as possible equally spaced. However the procedure includes provision, under control of parameter e, to reduce the number of harmonics calculated, mt, if rounding errors do start to build up.

Rounding error is controlled by estimating the error which would occur in the analysis of a standard function $q(x)$ for the given points, where

$$q(x) = \begin{cases} 1 & \text{if } index = 0, \\ n \sin (\pi x/h) / \sum_{j=1}^{n} | \sin (\pi x[j]/h) | & \text{if } index = 1. \end{cases}$$

The estimate used for the rounding error in the rth harmonic is

$$e_r = \sum_{i=index+1}^{r} c_i \times d_i,$$

where

$$c_i = \max | a_{ij} | \text{ for } index \leq j \leq i,$$

$$d_i = | \sum_{j=1}^{n} w_j q(x[j]) p_i(x[j]) / \sum_{j=1}^{n} w_j [p_i(x[j])]^2 |.$$

If for any r, $e_r > e$, the procedure is terminated with $mt = r - 1$.

REFERENCES:

1. CLENSHAW, C. W. Curve-fitting with a digital computer, *Comput. J.* 2, 170-173.
2. WATT, J. M. A note on the evaluation of trigonometric series. *Comput. J.* 1, 162;


```
begin
  integer i, j;  real s1, s2, s3, alpha, beta, c, d, u, v, w, g, s, mean,
    p, coeff, er, cer;
  array c1[0:m], c2[0:m+1];
  g := 3.1415926536/h;
  if index = 0 then mean := 1 else
  begin mean := 0;
    for i := 1 step 1 until n do
      mean := mean + abs(sin(g×x[i]));
    mean := n/mean
  end;
  for i := index step 1 until m do a[i] := 0;
  c2[m+1] := alpha := cer := 0;
  for i := 0 step 1 until m do c1[i] := c2[i] := 0;
  c1[index] := −1;
  beta := s3 := 1;  mt := index;
loop:  coeff := 0;  for i := index step 1 until mt do
  begin
    d := (if i=0 then c2[1] else c1[i−1]) + c2[i+1] − beta ×
      c1[i] − alpha × c2[i];
    c1[i] := c2[i];  c2[i] := d;  d := abs(d);
  if d > coeff then coeff := d
  end;
  s1 := s2 := d := er := 0;
  for i := 1 step 1 until n do
  begin
    c := 2 × cos(g×x[i]);
    if mt = 0 then begin p := 0.5;  go to sum end;
    u := v := 0;
    for j := mt step − 1 until 1 do
    begin
      w := c × u − v + c2[j];  v := u;  u := w
    end;
    if index = 0 then
    begin
      s := 1;  p := 0.5 × (u×c+c2[0]) − v
    end
    else
    begin
      s := sin(g×x[i]);  p := u × s
    end;
sum:  w := if x[i] = 0 ∨ x[i] = h then 0.5 else 1;
      d := d + w × p × f[i];
      if mt > index then er := er + w × p × s × mean;
      p := w × p ↑ 2;  s1 := s1 + c × p;  s2 := s2 + p
  end;
  cer := cer + coeff × abs(er)/s2;
  if cer > e then go to exit;  alpha := s1/s2;
  beta := s2/s3;  d := d/s2;  s3 := s2;
  for i := index step 1 until mt do
    a[i] := a[i] + d × c2[i];
  mt := mt + 1;  if mt ≤ m then go to loop;
exit:  mt := mt − 1
end trigfit;
procedure harmanalsymm (n, m, h, e, x, ypos, yneg, mc, ms, a, b);
  value n, m, h, e;  integer n, m, mc, ms;  real h, e;  array x,
    ypos, yneg, a, b;
comment Approximates a function y of x by a finite trigonometric series of period 2h from values specified at discrete points in the range (−h, h). Those points need not be equally spaced, but must be symmetrically distributed about the value x = 0. Thus only the values of x in the range 0 ≤ x ≤ h need be given.
```

The input parameters are:

- n—number of values of x in the range $0 \leq x \leq h$.
- m—order of the highest harmonic required.
- h—half-period of the fitted series.
- e—used to terminate the process if rounding errors start to accumulate excessively (see note on *trigfit*).
- x—the given values of x in the range $(0, h)$ are stored on $x[1], x[2], \cdots, x[n]$.
- *ypos*—the value of y corresponding to $x = +x[i]$ is stored on $ypos[i]$ $(i=1, 2, \cdots, n)$.
- *yneg*—the value of y corresponding to $x = -x[i]$ is stored on $yneg[i]$ $(i=1, 2, \cdots, n)$.

The procedure then calculates the coefficients $a[r]$ and $b[r]$ in the approximation

$$S(x) = \tfrac{1}{2}a[0] + \sum_{r=1}^{mc} a[r] \cos(r\pi x/h) + \sum_{r=1}^{ms} b[r] \sin(r\pi x/h).$$

Here normally $mc = ms = m$, but provision is included to calculate fewer harmonics if rounding errors begin to accumulate excessively (see note on *trigfit*), or if m exceeds its maximum permissible value. For the cosine terms this maximum value is $n - 1$. For the sine terms it is n, this figure being reduced by 1 for each $x[i]$ equal to 0 or h. The cosine and sine series are calculated separately by *trigfit*, with

$$f[i] = \begin{cases} 0.5 \times (ypos[i] + yneg[i]) & \text{for cosine series,} \\ 0.5 \times (ypos[i] - yneg[i]) & \text{for sine series;} \end{cases}$$

```
begin
  integer i, md;  array f[1:n];  procedure trigfit;
  for i := 1 step 1 until n do
    f[i] := 0.5 × (ypos[i] + yneg[i]);
  trigfit (0, n, if m ≥ n then n − 1 else m, h, e, x, f, mc, a);
  md := n;
  for i := 1 step 1 until n do
  begin
    f[i] := 0.5 × (ypos[i] − yneg[i]);
    if x[i] = 0 ∨ x[i] = h then md := md − 1
    end;
  trigfit (1, n, if md ≥ m then m else md, h, e, x, f, ms, b)
end harmanalsymm
```

ALGORITHM 321
t-TEST PROBABILITIES [S14]

JOHN MORRIS (Recd. 6 Jan. 1967, 18 July 1967, and 10 Oct. 1967)

Computer Institute for Social Science Research, Michigan State University, East Lansing, Michigan

KEY WORDS AND PHRASES: T-test, Student's t-statistic, distribution function

CR CATEGORIES: 5.5


```
real procedure ttest (x, df, maxn, gauss, error);
  value x, df, maxn; real x;  integer df, maxn;  real procedure
    gauss; label error;
comment This procedure gives the probability that t will be greater in absolute value than the absolute value of x, where t is the Student t-statistic, as defined and tabled by R. A. Fisher [2], evaluated at df degrees of freedom: that is, 2 times the integral of the distribution function of t, evaluated from abs(x) to infinity. The procedure may also be used, e.g., to estimate the two-tailed probability of a simple correlation, r, where N = the
```

number of pairs of observations, $df = N - 2$, and $t = r \times sqrt(df/(1.0 - r \uparrow 2))$ (cf. e.g. [5]).

For reasonably small df, Student's cosine formula is used [3, 4]:

$$ttest = 1.0 - coef \int_0^\theta \cos^{df-1}\theta \, d\theta$$

where $\theta = \arctan(t/sqrt(df))$ and

$$coef = (df-1)/(df-2) \times (df-3)/(df-4) \cdots \begin{cases} (\frac{4}{3}) \times (2/\pi) & \text{for odd } df, \\ (\frac{5}{4}) \times (\frac{3}{2}) \times (\frac{1}{2}) & \text{for even } df. \end{cases}$$

Integrated in series, this gives results which appear to be correct to very nearly the full single precision accuracy of the machine (in terms of the number of digits after the decimal point, not necessarily significant digits).

An approximation due to R. A. Fisher [1] gives results accurate to within $\pm 3 \times 10^{-7}$ when *maxn* has been set at 30. The tradeoff on time is also optimal at about this point. The **real procedure** *gauss* computes the area under the left-hand portion of the normal curve. Algorithm 209 [6] may be used for this purpose.

Thanks to the referee for many helpful suggestions, most of which have been incorporated, and to David F. Foster, who wrote an early version of part of the program.

References:

1. Fisher, R. A. *Metron 5* (1925), 109–112.
2. ——. *Statistical Methods for Research Workers*. Oliver and Boyd, Edinburgh, 1965.
3. Gosset, W. S. (Student). The probable error of a mean. *Biometrika 6* (1908), 1.
4. ——. New tables for testing the significance of observations. *Metron 5* (1925), 105.
5. Guilford, J. P. *Fundamental Statistics in Psychology and Education*. McGraw-Hill, New York, 1956, pp. 219–221.
6. Ibbetson, D. Algorithm 209, Gauss. *Comm. ACM, 6* (Oct. 1963), 616.

```
begin
  if df < 1 then go to error;
  if x = 0 then ttest := 1.0 else
  begin real t;
    t := abs (x);
    if df < maxn then
    begin integer i, nh;  real cth, sth, cthsq, xi, coef, z;
      z := t/sqrt(df);
      cth := 1.0/sqrt(z↑2+1.0);
      sth := z × cth;
      cthsq := cth ↑ 2;
      nh := (df−1) ÷ 2;
      if df = 2 × (df÷2) then
      begin
        t := sth;
        if nh = 0 then go to g;
        cth := cthsq;  xi := 1.0;
        coef := 0.5 × sth
      end else
      begin
        t := 0.6366197724 × arctan(z);
        comment 0.63661977236758134307553510··· = 2/π;
        if nh = 0 then go to g;
        xi := 0;  coef := 0.6366197724 × sth
      end;
      for i := 1 step 1 until nh do
      begin
        t := t + coef × cth;  cth := cth × cthsq;
        xi := xi + 2.0;
        coef := coef × xi/(xi+1.0)
      end;
    g: t := 1.0 − t
    end else
      if t > 6.0 then t := 0 else
      if df < ⏨6 then
    begin real f, t2, t4, t6, t8, t10, t12, t14, t16, t18;
      f := df;  t2 := t × t;  t4 := t2 × t2;  t6 := t4 × t2;
      t8 := t6 × t2;  t10 := t8 × t2;  t12 := t10 × t2;
      t14 := t12 × t2;  t16 := t14 × t2;  t18 := t16 × t2;
      comment 0.39894228040143267793994610··· = 1/sqrt (2×π);
      t := 2.0 × (gauss(−t)+t×0.3989422804 ×exp(−0.5×t2)×
        ((t2+1.0)/(4.0×f)+(3.0×t6−7.0×t4−5.0×t2−3.0)/
        (96.0×f×f)+(t10−11.0×t8+14.0×t6+6.0×t4−3.0×t2−
        15.0)/(384.0×f↑3)+(15.0×t14−375.0×t12+2225.0×t10−
        2141.0×t8−939.0×t6−213.0×t4−915.0×t2+945.0)/
        (92160.0×f↑4)+(3.0×t18−133.0×t16+1764.0×t14−
        7516.0×t12+5994.0×t10+2490.0×t8+1140.0×t6+180.0×
        t4+5355.0×t2+17955.0)/(368640.0×f↑5)))
    end else t := 2.0 × gauss(−t);
    ttest := if t <0 then 0 else t
  end
end ttest
```

ALGORITHM 322
F-DISTRIBUTION [S14]

Egon Dorrer (Recd. 25 Jan. 1967, 3 July 1967, and 17 Oct. 1967)

Institut für Photogrammetrie und Kartographie, Technische Hochschule München, W. Germany; now: Department of Surveying Engineering, University of New Brunswick, Fredericton, N.B., Canada

KEY WORDS AND PHRASES: Fisher's F-distribution, Student's t-distribution
CR CATEGORIES: 5.5

```
real procedure Fisher (m, n, x);
  value m, n, x;  integer m, n;  real x;
comment Fisher's F-distribution with m and n degrees of
  freedom. Computation of the probability
```

$$Pr(F < x) = \frac{\Gamma\left(\frac{m+n}{2}\right)}{\Gamma\left(\frac{m}{2}\right) \cdot \Gamma\left(\frac{n}{2}\right)} \cdot \int_0^w \frac{\xi^{m/2-1}}{(\xi+1)^{(m+n)/2}} \, d\xi,$$

where $w = (m/n)x$ and $F = (\sum_{i=1}^m x_i^2/m)/(\sum_{j=1}^n y_j^2/n)$. The solution results recursively from the basic integrals

$$Fisher\ (1,1,x) = 2\cdot\arctan\sqrt{w}/\pi, \quad Fisher\ (1,2,x) = (w/(w+1))^{\frac{1}{2}},$$

$$Fisher\ (2,1,x) = 1 - 1/(w+1)^{\frac{1}{2}}, \quad Fisher\ (2,2,x) = w/(w+1).$$

π is introduced by $0.3183098862 = 1/\pi$. By calling *Fisher* $(1, n, t\uparrow 2)$, Student's t-distribution will be obtained;

```
begin integer a, b, i, j;  real w, y, z, d, p;
  a := 2 × (m÷2) − m + 2;  b := 2 × (n÷2) − n+ 2;
    w := x × m/n;  z := 1/(1+w);
  if a = 1 then
  begin
    if b = 1 then
    begin
      p := sqrt(w);  y := 0.3183098862;
      d := y × z/p;  p := 2 × y × arctan(p)
    end else
    begin
      p := sqrt(w×z);  d := 0.5×p × z/w
    end
  end else
```

```
  if b = 1 then
  begin
    p := sqrt(z);  d := 0.5 × z × p;  p := 1 − p
  end else
  begin
    d := z × z;  p := w × z
  end;
  y := 2 × w/z;
  for j := b + 2 step 2 until n do
  begin
    d := (1 + a/(j−2)) × d × z;
    p := if a = 1 then p + d × y/(j−1) else (p+w) × z
  end j;
  y := w × z;  z := 2/z;  b := n − 2;
  for i := a + 2 step 2 until m do
  begin
    j:= i + b;  d := y × d × j/(i−2);  p := p − z × d/j
  end i;
  Fisher := p
end Fisher
```

ALGORITHM 323
GENERATION OF PERMUTATIONS IN LEXICOGRAPHIC ORDER [G6]

R. J. Ord-Smith (Recd. 27 Apr. 1967 and 26 July 1967)
Computing Laboratory, University of Bradford, Bradford, Yorkshire, England

KEY WORDS AND PHRASES: permutations, lexicographic order, lexicographic generation, permutation generation
CR CATEGORIES: 5.39

Author's Remark. Lexicographic generation involves more than the minimum of $n!$ transpositions for generation of the complete set of $n!$ permutations of n objects. The actual number of transpositions required can be shown to tend asymptotically to (cosh 1) $n! \doteqdot 1.53n!$ However, lexicographic generation can be described by an algorithm requiring very simple book-keeping. The author is indebted to Professor H. F. Trotter for suggesting an improvement to an original algorithm, which now results in a process more than twice as fast as the previously fastest lexicographic Algorithm 202 [*Comm. ACM 6* (Sept. 1963), 517]. Tabulated results below show *BESTLEX* to be only 9.3 percent slower than the transposition Algorithm 115 [*Comm. ACM 5* (Aug. 1962), 434] when $n = 8$.

The usual practice is adopted of using a nonlocal Boolean variable called *first* which may be assigned the value *true* to initialize generation. On procedure call this is set *false* and remains so until it is again set *true* when complete generation of permutations has been achieved. Table I gives results obtained for *BESTLEX*. The times given in seconds are for an I.C.T. 1905 computer. t_n is the time for complete generation of $n!$ permutations. r_n has the usual definition $r_n = t_n/(n \cdot t_{n-1})$.

TABLE I

Algorithm	t_7	t_8	r_8	*Number of transpositions*
BESTLEX	6	47	0.98	$\rightarrow 1.53n!$
202	12.4	100	1.00	?
115	5.6	43	0.98	$n!$

```
procedure BESTLEX (x, n);  value n;  integer n;  array x;
begin own integer array q[2:n];  integer k, m;  real t;
comment own dynamic arrays are not often implemented. The
  upper bound will then have to be given explicitly;
  if first then
  begin first := false;
    for m := 2 step 1 until n do q[m] := 1
  end of initialization process;
  if q[2] = 1 then
  begin q[2] := 2;
    t := x[1];  x[1] := x[2];  x[2] := t;
    go to finish
  end;
  for k := 2 step 1 until n do
    if q[k] = k then q[k] := 1  else go to trstart;
first := true;  k := n;  go to trinit;
trstart:  m := q[k];  t := x[m];  x[m] := x[k];  x[k] := t;
  q[k] := m + 1;  k := k − 1;
trinit:  m := 1;
transpose:  t := x[m];  x[m] := x[k];  x[k] := t;
  m := m + 1;  k := k − 1;
  if m < k then go to transpose;
finish:
end of procedure BESTLEX
```

ALGORITHM 324
MAXFLOW [H]

G. Bayer (Recd. 31 July 1967)
Technische Hochschule, Braunschweig, Germany

KEY WORDS AND PHRASES: network, linear programming, maximum flow
CR CATEGORIES: 5.41

```
procedure maxflow (from, to, cap, flow, v, n, mflow, source, sink,
inf, eps);
  value v, n, source, sink, inf;
  integer v, n, source, sink;  real inf, eps, mflow;
  integer array from, to;  array cap, flow;
comment The nodes of the network are numbered from 1 to sn.
  It is not necessary but reasonable that each number represent a
  node. The data of the network are given by arrays from, to, cap
  in the following manner. There is a maximum possible flow of
  cap[i], nonnegative, leading from from[i] to to[i], i = 1, ··· , v.
    Compute the maximum flow mflow from source to sink,
  (source and sink given by their node numbers). inf represents
  the greatest positive real number within machine capacity.
  flow[i] gives the actual flow from from[i] to to[i]. Flows abso-
  lutely less than eps are considered to be zero. Literature: G.
  Hadley, Linear Programming, Addison-Wesley, Reading (Mass.)
  and London, 1962, pp. 337–344.
    Multiple solutions are left out of account;
begin integer l, j, k, r, lk, ek, u, s;  real gjk, d;
  integer array low, up, klist, labj[1:n], ind[1:v];  real array
  labf[1:n];
comment Note structure of data lists in up and low;
  l := 1;
  for j := 1 step 1 until n do
  begin low[j] := l;
    for r := 1 step 1 until v do
    begin if from[r] = j then
      begin ind[l] := r;
        flow[l] := cap[l];  l := l + 1
      end
```

```
      end;
      up[j] := l - 1
    end;
    mflow := 0.0;
lab:;
    comment Prepare lists for new labeling;
    for j := 1 step 1 until n do
    begin labj[j] := klist[j] := 0;
      labf[j] := 0.0
    end;
    labf [source] := inf;
    comment labeling;
    j := source;  lk := ek := 0;
path:
    u := up[j];
    for s := low[j] step 1 until u do
    begin l := ind[s];
      k := to[l];  gjk := flow[l];
      if labj[k] ≠ 0 ∨ abs(gjk) < eps
        then go to end;
      labj[k] := j;
      labf[k] := if gjk < labf[j] then gjk else labf[j];
      if k = sink then go to reached;
      lk := lk + 1;  klist[lk] := k;
end:
    end;
    ek := ek + 1;  j := klist[ek];
    if j ≠ 0 then go to path else go to max;
    comment sink is labeled, find path and possible
      flow, reduce excess capacities along path;
reached:
    j := sink;  d := labf[j];  mflow := mflow + d;
look:  k := labj[j];  u := up[k];
    for s := low[k] step 1 until u do
    begin l := ind[s];
      if to[l] = j then flow[l] := flow[l] - d
    end;
    u := up[j];
    for s := low[j] step 1 until u do
    begin l := ind[s];
      if to[l] = k then flow[l] := flow[l] + d
    end;
    j := k;  if j ≠ source then go to look;
    go to lab;
max:;  comment maximal flow found;
    for l := 1 step 1 until v do
      flow[l] := cap[l] - flow[l]
end
```

ALGORITHM 325
ADJUSTMENT OF THE INVERSE OF A SYMMETRIC MATRIX WHEN TWO SYMMETRIC ELEMENTS ARE CHANGED [F1]

GERHARD ZIELKE (Recd. 24 Aug. 1967)
Institut für Numerische Mathematik der Martin Luther Universität Halle-Wittenberg, German Democratic Republic

KEY WORDS AND PHRASES: symmetric matrix, matrix inverse, matrix perturbation, matrix modification
CR CATEGORIES: 5.14

```
procedure INVSYM 2 (n, i, j, c, a, b);
  value n, i, j, c;  integer n, i, j;  real c;  array a, b;
```

comment *INVSYM* 2 computes the inverse $A^{-1} = a$ of a nonsingular symmetric nth order matrix $A = B + c(e_i e_j' + e_j e_i')$ which arises from a symmetric matrix B by a change c in two elements B_{ij} and $B_{ji} = B_{ij}$ $(i \neq j)$. The inverse matrix $B^{-1} = b$ is assumed to be known. The calculation with the new formula

$$a = b - \frac{c}{d}\,[b_{.i}(h_1 b_{j.} + h_2 b_{i.}) + b_{.j}(h_3 b_{j.} + h_1 b_{i.})]$$

where

$$h_1 = 1 + cb_{ij}\ ,\quad h_2 = -cb_{jj}\ ,\quad h_3 = -cb_{ii}\ ,\quad d = h_1{}^2 - h_2 h_3$$

requires $n^2 + O(n)$ multiplications, therefore only about the same number of operations as if the well-known Sherman-Morrison formula for a change in one element (see Algorithm 51 [*Comm. ACM 4* (Apr. 1961), 180]) is used. In these equations e_i denotes the ith column and e_i' the ith row of the unit matrix, $b_{.i} = be_i$ denotes the ith column and $b_{i.} = e_i' b$ the ith row of the matrix b;

```
begin integer k, l;  real h1, h2, h3, d;
  array r, s[1:n];
  h1 := 1 + c × b[i, j];  h2 := -c × b[j, j];
  h3 := -c × b[i, i];  d := h1 ↑ 2 - h2 × h3;  d := c/d;
  h1 := h1 × d;  h2 := h2 × d;  h3 := h3 × d;
  for k := 1 step 1 until n do
  begin
    r[k] := h1 × b[j, k] + h2 × b[i, k];
    s[k] := h3 × b[j, k] + h1 × b[i, k]
  end;
  for k := 1 step 1 until n do
  for l := 1 step 1 until k do
    a[k, l] := a[l, k] := b[k, l] - b[k, i] × r[l] - b[k, j] × s[l]
end INVSYM 2
```

MODIFIED SHARE CLASSIFICATIONS

[*Designations follow algorithm titles.*]

- A1 Real Arithmetic, Number Theory
- A2 Complex Arithmetic
- B1 Trig and Inverse Trig Functions
- B2 Hyperbolic Functions
- B3 Exponential and Logarithmic Functions
- B4 Roots and Powers
- C1 Operations on Polynomials and Power Series
- C2 Zeros of Polynomials
- C5 Zeros of One or More Transcendental Equations
- C6 Summation of Series, Convergence Acceleration
- D1 Quadrature
- D2 Ordinary Differential Equations
- D3 Partial Differential Equations
- D4 Differentiation
- E1 Interpolation
- E2 Curve and Surface Fitting
- E3 Smoothing
- E4 Minimizing or Maximizing a Function
- F1 Matrix Operations, Including Inversion
- F2 Eigenvalues and Eigenvectors of Matrices
- F3 Determinants
- F4 Simultaneous Linear Equations
- F5 Orthogonalization
- G1 Simple Calculations on Statistical Data
- G2 Correlation and Regression Analysis
- G5 Random Number Generators
- G6 Permutations and Combinations
- G7 Subset Generators and Classifications
- H Operations Research, Graph Structures
- I5 Input—Composite
- J6 Plotting
- K2 Relocation
- M1 Sorting
- M2 Data Conversion and Scaling
- O2 Simulation of Computing Structure
- S Approximation of Special Functions... Functions are Classified S01 to S22, Following Fletcher-Miller-Rosenhead, Index of Math. Tables
- Z All Others