

Algorithms

J. H. WEGSTEIN, Editor

ALGORITHM 125 WEIGHTCOEFF

H. RUTISHAUSER

Eidg. Technische Hochschule, Zurich, Switzerland

procedure *weightcoeff* (*n,q,e,eps,w,x*); **value** *n*; **real** *eps*;
integer *n*; **array** *q,e,w,x*;
comment Computes abscissae x_i and weight coefficients w_i for a Gaussian quadrature method $\int_0^b w(x)f(x) dx \approx \sum_{i=1}^n w_i f(x_i)$, where $\int_0^b w(x) dx = 1$ and $w(x) \geq 0$. The method requires the order n , a tolerance *eps* and the $2n-1$ first coefficients of the continued fraction

$$\int_0^b \frac{w(x)}{z-x} dx = \frac{1}{z} - \frac{q_1}{1} - \frac{e_1}{z} - \frac{q_2}{1} - \frac{e_2}{z} - \dots$$

to be given, the latter as two arrays $q[1:n]$ and $e[1:n-1]$ all components of which are automatically positive by virtue of the condition $w(x) \geq 0$. The method works as well if the upper bound b is actually infinity (note that b does not appear directly as parameter!) or if the density $w(x) dx$ is replaced by $d\alpha(x)$ with a monotonically increasing $\alpha(x)$ with at least n points of variation. The tolerance *eps* should be given in accordance to the machine accuracy, e.g. as 10^{-10} for a computer with a ten-digit mantissa. The result is delivered as two arrays $w[1:n]$ (the weight coefficients) and $x[1:n]$ (the abscissae). For a description of the method see H. Rutishauser, "On a modification of the QD-algorithm with Graeffe-type convergence" [Proceedings of the IFIPS Congress, Munich, 1962].;

begin

integer *k*;

Boolean *test*;

real *m, p*;

array *g[1:n]*;

procedure *red* (*a,f,n*); **value** *n*; **integer** *n*; **array** *a,f*;

comment subprocedure *red* reduces a heptadiagonal matrix a to tridiagonal form as described in the paper loc. cit. Since the bulk of the computing time of the whole method is spent in this subprocedure, it would pay to write it in machine code.;

begin

real *c*; **integer** *j,k*;

for *k* := 1 **step** 1 **until** $n-1$ **do**

begin

for *j* := *k* **step** 1 **until** $n-1$ **do**

begin

$c := -f[j] \times a[j,7]/a[j,2]$;

$a[j,7] := 0$;

$a[j+1,2] := a[j+1,2] + c \times a[j,5]$;

$a[j,1] := a[j,1] - c \times f[j] \times a[j,4]$;

$a[j,6] := a[j,6] - c \times a[j+1,1]$;

$a[j+1,3] := a[j+1,3] - c \times a[j+1,6]$;

end *j*;

for *j* := *k* **step** 1 **until** $n-1$ **do**

begin

$c := -f[j] \times a[j,4]/a[j,1]$;

$a[j,4] := 0$;

$a[j+1,1] := a[j+1,1] + c \times a[j,6]$;

$a[j+1,6] := a[j+1,6] + c \times a[j+1,3]$;

$a[j,5] := a[j,5] - c \times a[j+1,2]$;

$a[j+1,0] := a[j+1,0] - c \times a[j+1,5]$;

end *j*;

for *j* := $k+1$ **step** 1 **until** $n-1$ **do**

begin

$c := -a[j,3]/a[j-1,6]$;

$a[j,3] := 0$;

$a[j,6] := a[j,6] + c \times a[j,1]$;

$a[j-1,5] := a[j-1,5] - c \times f[j] \times f[j] \times a[j,0]$;

$a[j,2] := a[j,2] - c \times f[j] \times f[j] \times a[j,5]$;

$a[j,7] := a[j,7] - c \times f[j] \times a[j+1,2]$;

end *j*;

for *j* := $k+1$ **step** 1 **until** $n-1$ **do**

begin

$c := -a[j,0]/a[j-1,5]$;

$a[j,0] := 0$;

$a[j+1,2] := a[j+1,2] + c \times f[j] \times a[j,7]$;

$a[j,5] := a[j,5] + c \times a[j,2]$;

$a[j,1] := a[j,1] - c \times f[j] \times f[j] \times a[j,6]$;

$a[j,4] := a[j,4] - c \times f[j] \times a[j+1,1]$;

end *j*;

end *k*;

end *red*;

procedure *qdgraeffe* (*n,h,g,f*); **value** *n*;

integer *n*; **array** *h,g,f*;

comment Subprocedure *qdgraeffe* computes for a given finite continued fraction

$$f(z) = \frac{1}{z} - \frac{q_1}{1} - \frac{e_1}{z} - \frac{q_2}{1} - \dots - \frac{q_n}{1}$$

another one, the poles of which are the squares of the poles of $f(z)$. However *qdgraeffe* uses not the coefficients q_1, \dots, q_n and e_1, \dots, e_{n-1} of $f(z)$, but the quotients

$$\begin{cases} f_k = q_{k+1}/q_k \\ g_k = e_k/q_{k+1} \end{cases} \quad (k := 1, 2, \dots, n-1)$$

and the $h_k = \ln(\text{abs}(q_k))$ ($k := 1, 2, \dots, n$), and the results are delivered in the same form. Procedure *qdgraeffe* can be used independently, but requires subprocedure *red* above;

begin

integer *k*; **array** *a[0:n,0:7]*;

$g[n] := f[n] := 0$;

for *k* := 1 **step** 1 **until** n **do**

begin

$a[k-1,4] := a[k-1,5] := 1$;

$a[k,1] := a[k,2] := 1 + g[k] \times f[k]$;

$a[k,6] := a[k,7] := g[k]$;

$a[k,0] := a[k,3] := 0$;

comment The array a represents the heptadiagonal matrix Q of the paper loc. cit., but with the modifications needed to avoid the large numbers and with a peculiar

```

    arrangement.;
  end k;
  a[n,5] := 0;
  red(a,f,n);
  for k := 1 step 1 until n do
    h[k] := 2 × h[k] + ln(abs(a[k,1] × a[k,2]));
    comment A saving might be achieved by economizing the
    log-computation in the range .8 ≤ x ≤ 1.2;
  for k := 1 step 1 until n-1 do
  begin
    f[k] := f[k] × f[k] × a[k+1,2] × a[k+1,1]/(a[k,1] × a[k,2]);
    g[k] := a[k,5] × a[k,6]/(a[k+1,1] × a[k+1,2])
  end k;
end qdgraeffe;
L1: x[1] := q[1] + e[1];
  for k := 2 step 1 until n do
  begin
    g[k-1] := e[k-1] × q[k]/x[k-1];
    x[k] := q[k] + (if k=n then 0 else e[k] - g[k-1]);
    g[k-1] := g[k-1]/x[k];
    w[k-1] := x[k]/x[k-1];
    x[k-1] := ln(x[k-1]);
  end k;
  x[n] := ln(x[n]);
L2: p := 1;
L25: begin
  test := true;
  for k := 1 step 1 until n-1 do
    test := test ∧ abs(g[k] × w[k]) < eps;
  if test then go to L3;
  qdgraeffe (n,x,g,w);
end;
p := 2 × p;
go to L25;
comment What follows is a peculiar method to compute
the wk from given ratios gk = wk+1/wk such that  $\sum_{k=1}^n w_k = 1$ ,
but the straightforward formulae to do this might well
produce overflow of exponent.;
L3: w[1] := m := 0;
  for k := 1 step 1 until n-1 do
  begin
    w[k+1] := w[k] + ln(g[k]);
    if w[k] > m then m := w[k];
  end k;
  for k := 1 step 1 until n do w[k] := exp(w[k]-m);
  m := 0;
  for k := 1 step 1 until n do m := m + w[k];
  for k := 1 step 1 until n do begin w[k] := w[k]/m;
    x[k] := exp(x[k]/p) end;
end weightcoeff

```

ALGORITHM 126 GAUSS' METHOD

JAY W. COUNTS

University of Missouri, Columbia, Mo.

```

procedure gauss (u,a,y);
real array a,y; integer u;
comment This procedure is for solving a system of linear equa-
tions by successive elimination of the unknowns. The augmented
matrix is a and u is the number of unknowns. The solution vector
is y. If the system hasn't any solution or many solutions, this is
indicated by the go to error where error is a label outside the
procedure.;
begin
  integer i,j,k,m,n;
  n := 0;

```

```

ck0: n := n + 1;
  for k := n step 1 until u do if a[k,n] ≠ 0 then go to ck1;
  go to error;
ck1: if k = n then go to ck2;
  for m := n step 1 until u+1 do
  begin
    temp := a[n,m]; a[n,m] := a[k,m]; a[k,m] := temp
  end;
ck2: for j := u + 1 step -1 until n do a[n,j] := a[n,j]/a[n,n];
  for i := k + 1 step 1 until u do
  for j := n + 1 step 1 until u + 1 do
    a[i,j] := a[i,j] - a[i,n] × a[n,j];
  if n≠u then go to ck0;
  for i := u step -1 until 1 do
  begin
    y[i] := a[i,u + 1]/a[i,i];
    for k := i - 1 step -1 until 1 do
      a[k,u + 1] := a[k,u + 1] - a[k,i] × y[i]
    end end;

```

ALGORITHM 127

ORTHO

PHILIP J. WALSH

National Bureau of Standards, Washington, D. C.

```

procedure ORTHO(W,Y,Z,n,fn,m,p,r,ai,au1,mu1,ze1,X,DEV,
  COF,STD,CV,VCV,gmdt,Q,Q2,E,EP,A,GF,ENF);
value n,m,p,r,ai,au1,mu1,ze1;
real fn,gmdt;
array W,Y,Z,X,DEV,COF,STD,CV,VCV,Q,Q2,E,EP,A,GF,ENF;
integer n,m,p,r,ai,au1,ze1,mu1;
switch at := at1,at2; switch ze := ze1, ze2;
switch au := au1,au2; switch mu := mu1,mu2, mu3;
comment ORTHO is a general purpose procedure which is
capable of solving a wide variety of problems. For a detailed
discussion of the applications listed below and other applica-
tions, see (1) Philip Davis and Philip Rabinowitz, "A Multiple
Purpose Orthonormalizing Code and Its Uses," J. ACM 1
(1954), 183-191, (2) Philip Davis, "Orthonormalizing Codes in
Numerical Analysis," in J. Todd (Ed.), A Survey of Numerical
Analysis, Ch. 10 (McGraw-Hill, 1962), (3) Philip Davis and
Philip Rabinowitz, "Advances in Orthonormalizing Computa-
tion," in F. L. Alt (Ed.), Advances in Computers, Vol. 2, pp. 55-
133 (Academic Press, 1961), (4) Philip J. Walsh and Emilie V.
Haynsworth, General Purpose Orthonormalizing Code, SHARE
Abstr. #850. APPLICATIONS: (a) orthonormalizing a set of
vectors with respect to a general inner product, (b) least squares
approximation to given functions by polynomial approximations
or any linear combination of powers, rational functions, trans-
cendental functions and special functions, such as those defined
numerically by a set of values, (c) curve fitting of empirical data
in two or more dimensions, (d) finding the best solution in the
l.s.s. to a system of m linear equations in n unknowns ( $n \leq m$ ),
(e) matrix inversion and solution of linear systems of equations,
(f) expansion of functions in a series of orthogonal functions,
such as a series of Legendre or Chebyshev polynomials.
The following information must be supplied to the procedure.
(We are considering here the approximation feature of the pro-
cedure.)
n the number of components per vector (excluding augmenta-
tion)
m the number of vectors used in the approximation. For a
polynomial fit of degree t, set  $m=t+1$ .
p the number of augmented components per vector. A feature
of this procedure is that once the approximating vectors

```

have been orthonormalized, they may be used in approximating r functions without repeating the orthonormalization procedure on the original approximating vectors.

- r the number of functions to be approximated.
- ai a switch control concerning the approximating vectors. With $ai=1$, the procedure selects the first n components of the first row of $[Z]$, supplied by user. The i powers of these values are computed and stored into working location $[X]$, $i=0(1)m-1$. This is the usual set up for a polynomial fit. With $ai=2$, the procedure selects the first n components of the first m rows of $[Z]$ supplied by user and stores them into working location $[X]$.
- awi a switch control concerning augmentation on the approximating vectors. If $p=0$, this switch is ignored. With $awi=1$, regular augmentation is applied to the vectors in $[X]$. p zeros are stored after the n th component of the first m rows of $[X]$. The $(n+i)$ th component is replaced by 1.0, $i=1(1)m$. With $awi=2$, special augmentation is applied to the vectors in $[X]$. The p components located after the n th component of the first m rows of $[Z]$ supplied by the user augment $[X]$.
- zei a switch control concerning augmentation on the functions to be approximated. If $r=0$, this switch is ignored. With $zei=1$, regular augmentation is applied to the functions during the calculation. The n components of the first r rows of $[Y]$ supplied by user will be augmented by p zeros when moving $[Y]$ to $[X]$. With $zei=2$, special augmentation is applied. The first n components of the first r rows of $[Y]$ are the functional values supplied by user. The next p components of the first r rows of $[Y]$ are special values also supplied by user.
- mui a switch control concerning weights. $[W]$ is an $n \times n$ real, positive definite, symmetric matrix of weights. It is generally diagonal and often the Identity matrix. $mui=1$ when $[W]=I_n$, the matrix $[W]$ need not be supplied. $mui=2$ when $[W]$ is diagonal, but not I_n . The procedure is supplied the n diagonal elements of $[W]$, but stored in the first row of matrix $[W]$. $mui=3$ when the full weighting matrix is supplied to the procedure.

The following list of matrix arrays is given to aid the user in determining the number of components and vectors in the input and results. $W[1:n,1:n]$, $Y[1:r,1:n+p]$, $Z[1:m,1:n+p]$, $X[1:m+1,1:n+p]$, $DEV[1:r,1:n]$, $COF[1:r,1:p]$, $STD[1:r]$, $CV[1:p+1,1:p]$, $VCV[1:r,1:p+1,1:p]$, $Q[1:r,1:m+1]$, $Q2$, E , $EP[1:r,1:m]$, $A[1:m,1:p]$, $GF[1:m+r]$, $ENF[1:m]$.

The results of the procedure are stored in the following locations. The user must be sufficiently familiar with the theory to know which results are relevant to his application of the procedure. All vectors are stored row-wise in the matrices listed below.

- X orthonormal vectors
- DEV deviations
- COF coefficients
- STD standard deviations
- CV covariance matrix, stored in upper triangular form. The $(p+1)$ st row contains the square root of the diagonal elements of the matrix.
- VCV variance-covariance matrices, stored in upper triangular form with the $(p+1)$ st rows containing the square root of the diagonal elements. There are r such matrices, the first subscript running over the r values.
- $gmdt$ Gram determinant value
- Q Fourier coefficients
- $Q2$ squared Fourier coefficients
- E sum of the squared residuals
- EP residuals
- A a lower triangular matrix used to calculate the covariance matrix. $CV = A'A$.

GF Gram factors

ENF norms of the approximating vectors;

begin

integer $npp, npm, m1, n2, m2, r1, rbar, p2, bei, rhi, i18, gai, sii, i, j, dei, mui, elz1, elz2, k, thi, ali, omi, nui$;

array $PK, XP[1:n+p], QK[1:m+1]$;

real $denom, sum, dk2, dk, fi, ss, ssq$;

switch $be := be1, be2$; **switch** $rh := rh1, rh2$; **switch** $ga := ga1, ga2$;

switch $si := si1, si2$; **switch** $de := de1, de2$; **switch** $nu := nu1, nu2$;

switch $th := th1, th2, th3$; **switch** $al := al1, al2$;

switch $om := om1, om2$;

$npp := n+p$; $npm := n+m$; $m1 := m-1$; $n2 := n+1$; $m2 := m+1$;

$r1 := 0$; $rbar := r$; $p2 := p+1$; $denom :=$ **if** $n=m$ **then** 1.0 **else** $\sqrt{n-m}$; $bei := rhi := i18 := 1$;

if $(p \neq 0)$ **then** $gai := sii := 2$ **else** $gai := sii := 1$;

$box1$: **go to** $at[ai]$;

$at1$: **for** $j := 1$ **step** 1 **until** n **do begin**
 $X(2,j) := Z(1,j)$; $X[1,j] := 1.0$ **end**;

for $i := 2$ **step** 1 **until** $m1$ **do begin**
for $j := 1$ **step** 1 **until** n **do**
 $X[i+1,j] := X[i,j] \times X[2,j]$ **end**; **go to** $box2$;

$at2$: **for** $i := 1$ **step** 1 **until** m **do begin**
for $j := 1$ **step** 1 **until** n **do**
 $X[i,j] := Z[i,j]$ **end**;

$box2$: **if** $p=0$ **then go to** $box3$ **else go to** $au[awi]$;

$au1$: **for** $i := 1$ **step** 1 **until** m **do begin**
for $j := n2$ **step** 1 **until** npp **do**
 $X[i,j] := 0.0$; $X[i, n+i] := 1.0$ **end**; **go to** $box3$;

$au2$: **for** $i := 1$ **step** 1 **until** m **do begin**
for $j := n2$ **step** 1 **until** npp **do**
 $X[i,j] := Z[i,j]$ **end**;

$box3$: $dei := nui := elz1 := elz2 := k := 1$;

$box4$: $thi := 1$;

$box5$: $ali := omi := 1$; **if** $p=0$ **then go to** $box6$ **else**
for $j := 1$ **step** 1 **until** p **do** $PK[n+j] := 0.0$;

go to $mu[mui]$;

$mu1$: **for** $i := 1$ **step** 1 **until** n **do** $PK[i] := X[k,i]$;

go to $box7$;

$mu2$: **for** $i := 1$ **step** 1 **until** n **do**
 $PK[i] := X[k,i] \times W[1,i]$; **go to** $box7$;

$mu3$: **for** $i := 1$ **step** 1 **until** n **do begin** $sum := 0.0$;

for $j := 1$ **step** 1 **until** n **do** $sum := sum + X[k,j] \times W[i,j]$;

$PK[i] := sum$ **end**;

$box7$: **go to** $om[omi]$;

$om1$: **for** $i := 1$ **step** 1 **until** k **do begin** $sum := 0.0$;

for $j := 1$ **step** 1 **until** npp **do**
 $sum := sum + PK[j] \times X[i,j]$; $QK[i] := sum$ **end**;

go to $box8$;

$om2$: $dk2 := 0.0$; **for** $i := 1$ **step** 1 **until** npp **do**
 $dk2 := dk2 + PK[i] \times X[k,i]$;

$dk := \sqrt{dk2}$;

$GF[i18] := dk$; $i18 := i18 + 1$;

for $i := 1$ **step** 1 **until** npp **do**
 $X[k,i] := X[k,i]/dk$;

$omi := 1$; **go to** $box6$;

$box8$: **go to** $de[dei]$;

$de1$: $elz1 := -elz1$; **if** $elz1 < 0$ **then go to** $box8b$ **else**
go to $box8a$;

$box8a$: **for** $i := 1$ **step** 1 **until** $k-1$ **do**
 $QK[i] := -QK[i]$; $QK[k] := 1.0$;

for $i := 1$ **step** 1 **until** npp **do begin**
 $sum := 0.0$; **for** $j := 1$ **step** 1 **until** k **do**
 $sum := sum + X[j,i] \times QK[j]$;

$XP[i] := sum$ **end**; **go to** $box9$;

$box8b$: $ENF[i18] := \sqrt{QK[k]}$; **go to** $box8a$;

$de2$: $elz2 := -elz2$; **if** $elz2 < 0$ **then go to** $box8c$ **else**

```

go to box8a;
box8c: for i := 1 step 1 until m do begin
  Q[r1,i] := QK[i]; Q2[r1,i] := QK[i] × QK[i] end;
  Q[r1,m2] := QK[m2]; E[r1,1] := Q[r1,m2] - Q2[r1,1];
  for j := 2 step 1 until m do
    E[r1,j] := E[r1,j-1] - Q2[r1,j];
  fi := 1.0;
  for i := 1 step 1 until m do begin
    if (fn-fi) > 0.0 then begin if E[r1,i] < 0.0 then begin
      EP[r1,i] := -sqrt(abs(E[r1,i])/(fn-fi)); go to box8d;
    end
    else EP[r1,i] := sqrt(E[r1,i]/(fn-fi));
    go to box8d; end else E[r1,i] := -1.0;
box8d: fi := fi+1.0; end go to box8a;
box9: go to th[thi];
th1: for i := 1 step 1 until npp do
  X[k,i] := XP[i]; go to box10;
th2: for i := 1 step 1 until n do
  DEV[r1,i] := XP[i];
  for i := 1 step 1 until p do
    COF[r1,i] := -XP[n+i]; thi := 3; go to th1;
th3: go to box11;
box10: go to al[ali];
al1: omi := ali := 2; go to box6;
al2: if k < m then begin k := k+1; go to box4; end
  else go to box12;
box11: go to nu[nui];
nu1: nui := 2; go to box14;
nu2: ss := dk/denom; ssq := ss × ss;
  STD[r1] := ss; go to box14;
box12: go to be[bei];
be1: for i := 1 step 1 until m do begin
  for j := 1 step 1 until p do
    A[i,j] := X[i,n+j] end;
  gmdt := 1.0; for i := 1 step 1 until m do
    gmdt := gmdt × (GF[i]/ENF[i]);
  gmdt := gmdt × gmdt; dei := bei := thi := 2;
  k := k + 1; go to box13;
be2: go to box11;
box13: go to ga[gai];
ga1: go to box11;
ga2: for i := 1 step 1 until p do begin
  for j := i step 1 until p do begin
    sum := 0.0;
    for nii := 1 step 1 until m do
      sum := sum + A[nii,i] × A[nii,j];
    CV[i,j] := sum end end;
  for i := 1 step 1 until p do
    CV[p2,i] := sqrt(CV[i,i]); gai := 1; go to box11;
box14: go to rh[rhi];
rh1: if rbar = 0 then go to final else rbar := rbar - 1;
  r1 := r1 + 1; thi := rhi := 2; go to ze[zei];
ze1: for i := 1 step 1 until n do
  X[m2,i] := Y[r1,i];
  for i := 1 step 1 until p do
    X[m2,n+i] := 0.0; go to box5;
ze2: for i := 1 step 1 until npp do
  X[m2,i] := Y[r1,i]; go to box5;
rh2: go to si[sii];
si1: go to rh1;
si2: for i := 1 step 1 until p do begin
  for j := i step 1 until p do
    VCV[r1,i,j] := ssq × CV[i,j] end;
  for i := 1 step 1 until p do
    VCV[r1,p2,i] := ss × CV[p2,i]; go to rh1;
final: end ortho

```

ALGORITHM 128
SUMMATION OF FOURIER SERIES
M. WELLS
University of Leeds, Leeds 2, England*

* Currently with Burroughs Corp., Pasadena, Calif.

```

procedure Fourier (X, r, w, n, A, B);
value n; real X, w, A, B; integer r, n;
comment Fourier sums a one-dimensional Fourier series,
  using a recurrence relation described by Watt [Computer
  J. 1, 4 (1959) 162]. The parameters are the coefficients X, which
  are selected by r, w, the argument and n the total number of
  terms in the series. On exit A =  $\sum_{r=0}^{n-1} X_r \cos(rw)$  and
  B =  $\sum_{r=0}^{n-1} X_r \sin(rw)$ . Fourier is particularly efficient
  where  $X_r = 0$  for all  $r >$  some  $r_1$  and  $X_r \neq 0$  for all  $r \leq r_1$ ;
begin real t, tr, tr1, cosw2;
  tr1 := 0; cosw2 := 2 × cos(w);
  for r := n-1 step -1 until 0 do
    begin if X ≠ 0 then go to term end search for nonzero term;
    tr := 0; go to all zeroes;
term: tr := X; for r := r-1 step -1 until 0 do
  begin t := tr × cosw2 + X - tr1; tr1 := tr; tr := t end
  recurrence;
all zeros: A := tr - tr1 × cosw2/2; B := tr1 × sin(w)
end Fourier series

```

CERTIFICATION OF ALGORITHM 40
CRITICAL PATH SCHEDULING [B. Leavenworth,
Comm. ACM (Mar. 1961)]

LARS HELMBERG
Facit Electronics AB, Solna, Sweden.

The Critical Path Scheduling algorithm was transliterated into
FACT-ALGOL-1 and tested on the FACT EDB. The modifications
suggested by Alexander [*Comm. ACM* (Sept. 1961)] were included.
Results were correct in all tested schedules.

Contributions to this department must be in the form
stated in the Algorithms Department policy statement
(*Communications*, February, 1960) except that ALGOL 60
notation should be used (see *Communications*, May 1960).
Contributions should be sent in duplicate to J. H. Wegstein,
Computation Laboratory, National Bureau of Standards,
Washington 25, D. C. Algorithms should be in the Reference
form of ALGOL 60 and written in a style patterned after the
most recent algorithms appearing in this department. For
the convenience of the printer, please underline words that
are delimiters to appear in boldface type.

Although each algorithm has been tested by its contrib-
utor, no warranty, express or implied, is made by the contrib-
utor, the editor, or the Association for Computing
Machinery as to the accuracy and functioning of the algo-
rithm and related algorithm material, and no responsi-
bility is assumed by the contributor, the editor, or the
Association for Computing Machinery in connection there-
with.

The reproduction of algorithms appearing in this depart-
ment is explicitly permitted without any charge. When re-
production is for publication purposes, reference must be
made to the algorithm author and to the *Communications*
issue bearing the algorithm.

REMARK ON ALGORITHM 73
 INCOMPLETE ELLIPTIC INTEGRALS [David K.
 Jefferson, *Comm. ACM* (Dec. 1961)]

DAVID K. JEFFERSON
 U. S. Naval Weapons Laboratory, Dahlgren, Virginia

In regard to Algorithm 73, two errors were found:
 The 34th line of the procedure

$$F := \text{abs}(k) \times \text{sqrt}(1 - \sin^2 \phi) \uparrow 2) \\
 \times (1 - k \uparrow 2 \times \sin^2 \phi \uparrow 2) \uparrow ((2 \times n - 1)/(2 \times n));$$

should read

$$F := \text{abs}(k) \times \text{sqrt}(1 - \sin^2 \phi \uparrow 2) \\
 \times (1 - k \uparrow 2 \times \sin^2 \phi \uparrow 2) \uparrow ((2 \times n - 1)/2)/(2 \times n);$$

The 37th line

$$L[2] := L[1] + 1/(n \times 2 \times n - 1);$$

should read

$$L[2] := L[1] + 1/(n \times (2 \times n - 1));$$

In addition, efficiency is improved by interchanging lines 13
 and 14:

Step 1: $n := n + 1;$
 $\cos \phi := \cos(\phi);$

can be replaced by

$\cos \phi := \cos(\phi);$
 Step 1: $n := n + 1;$

CERTIFICATION OF ALGORITHM 87
 PERMUTATION GENERATOR [John R. Howell,
Comm. ACM (Apr. 1962)]
 G. F. SCHRACK and M. SHIMRAT
 University of Alberta, Calgary, Alb., Canada

PERMUTATION GENERATOR was translated into FORTRAN
 for the IBM 1620 and it performed satisfactorily. The algorithm
 was timed for several small values of n . For purposes of comparison
 we include the times (in seconds) for PERMULEX (Algorithm
 102).

	n	3	4	5	6	7
PERMUTATION GENERATOR		3	41	558	—	—
PERMULEX		—	3	6	37	278

As can be seen from this table, PERMUTATION GENERATOR is
 considerably slower. It is probable that one could speed up
 PERMUTATION GENERATOR to a great extent by rearranging
 the algorithm in such a manner that the digits of a number to a
 certain base are permuted rather than the elements of a sequence.

CERTIFICATION OF ALGORITHM 93
 GENERAL ORDER ARITHMETIC [Millard H. Per-
 stein, *Comm. ACM* (June 1962)]
 RICHARD GEORGE
 Particle Accelerator Div. Argonne National Laboratory,
 Argonne, Ill.

Algorithm 93 was programmed for the IBM 1620, using
 "FORTRAN-recursion" (i.e., generous use of the copy rule). The
 program ran without any modifications and was tested through
 tetration. Further levels were available, but were too time-
 consuming to reach.

CERTIFICATION OF ALGORITHM 115
 PERM [H. F. Trotter, *Comm. ACM* (Aug. 1962)]
 G. F. SCHRACK
 University of Alberta, Calgary, Alb., Canada

PERM was translated into FORTRAN for the IBM 1620 and it
 performed satisfactorily. Timing tests were carried out under the
 same conditions as for PERMUTATION (Algorithm 71) and
 PERMUTE (Algorithm 86).

PERM is indeed the fastest permutation generator so far en-
 countered. For $n = 8$, PERM is 25% faster than PERMUTE
 (989 against 1316 sec.). The values for r_n are (for a definition of
 r_n , see Certification of Algorithm 71, *Comm. ACM*, Apr. 1962):

n	6	7	8
r_n	.92	.95	.98



ACM Sort Symposium

November 29, 30, 1962

Nassau Inn, Princeton, N. J.

Featuring

- ... Latest techniques implemented by computer manufacturers and users
- ... Hypothetical solutions for future generation computers
- ... New techniques for computers having special peripheral devices

Open Discussion Periods Will Follow Each Paper

* * *

Are you a member of the computing profession?

Are you interested in sorting?

If so, you are urged to attend and participate

Advance Registration Required
 No Registration Fee
 Only advance payment for lunch

Conference Chairman: MARTIN A. GOETZ,
 Applied Data Research, Inc., Princeton, N. J.

Address registration requests and inquiries to:
 Mrs. L. R. Becker, Applied Data Research, Inc.,
 759 State Road, Princeton, N. J.