Algorithms

G. E. FORSYTHE, J. G. HERRIOT, Editors

ALGORITHM 252 [Z]
VECTOR COUPLING OR CLEBSCH-GORDAN COEFFICIENTS

J. H. GUNN
(Recd. 17 Aug. 1964, 13 Nov. 1964 and 21 Dec. 1964)
Nordisk Institut for Teoretisk Atomfysik, Copenhagen, Denmark

```
real procedure VCC(J1, J2, J, M1, M2, M, factorial);
value J1, J2, J, M1, M2, M;
integer J1, J2, J, M1, M2, M; array factorial;
comment VCC calculates the vector coupling or Clebsch-Gordan coefficients defined by the following formula
```

$$(j_1 m_1 j_2 m_2 \mid j_1 j_2 j m)$$

$$= \delta(m_1 + m_2, m)\left[\frac{(2j+1)(j_1+j_2-j)!(j_1-j_2+j)!(-j_1+j_2+j)!}{(j_1+j_2+j+1)!}\right]^{\frac{1}{2}}$$

$$\times [(j_1+m_1)!(j_1-m_1)!(j_2+m_2)!(j_2-m_2)!(j+m)!(j-m)!]^{\frac{1}{2}}$$

$$\times \sum_z (-1)^z/[z!(j_1+j_2-j-z)!(j_1-m_1-z)!$$

$$(j_2+m_2-z)!(j-j_2+m_1+z)!(j-j_1-m_2+z)!]$$

```
where j1 = J1/2, j2 = J2/2, j = J/2, m1 = M1/2, m2 = M2/2,
m = M/2. [Reference formula 3.6.11, p. 45 of EDMONDS, Alan
R. Angular momentum in quantum mechanics. In Investigations in Physics, 4, Princeton U. Press, 1957.]. The parameters
of the procedure, J1, J2, J, M1, M2 and M, are interpreted as
being twice their physical value, so that actual parameters may
be integers. Thus to call the procedure to calculate (½ 0 ½ 0 | ½
½ 0 0) the call would be VCC(1, 1, 0, 0, 0, 0, factorial). The procedure checks that the triangle conditions for the existence of a
coefficient are satisfied and that j1 + j2 + j is integral. If the
conditions are not satisfied the value of the procedure is zero.
The parameter factorial is an array containing the factorials
from 0 up to j1 + j2 + j + 1. Since in actual calculations the
procedure VCC will be called many times it is more economical
to have the factorials in a global array rather than compute
them on every entry to the procedure;
begin integer z, zmin, zmax; real cc;
if M1 + M2 ≠ M ∨ abs(M1) > abs(J1) ∨ abs(M2) > abs(J2) ∨
   abs(M) > abs(J) ∨ J > J1 + J2 ∨ J < abs(J1−J2) ∨ J1
   + J2 + J ≠ 2 × ((J1+J2+J)÷2) then VCC := 0 else
  begin zmin := 0;
    if J − J2 + M1 < 0 then zmin := −J + J2 − M1;
    if J − J1 − M2 + zmin < 0 then zmin := −J + J1 + M2;
    zmax: = J1 + J2 − J;
    if J2 + M2 − zmax < 0 then zmax := J2 + M2;
    if J1 − M1 − zmax < 0 then zmax := J1 − M1;
    cc := 0;
    for z := zmin step 2 until zmax do
    cc := cc + (if z=4×(z÷4) then 1 else −1)/(factorial[z÷2]
        × factorial[(J1+J2−J−z)÷2]
        × factorial[(J1−M1−z)÷2]
        × factorial[(J2+M2−z)÷2]
        × factorial[(J−J2+M1+z)÷2]
        × factorial[(J−J1−M2+z)÷2]);
    VCC := sqrt((J+1) × factorial[(J1+J2−J)÷2]
        × factorial[(J1−J2+J)÷2]
        × factorial[(−J1+J2+J)÷2] × factorial[(J1+M1)÷2]
        × factorial[(J1−M1)÷2] × factorial[(J2+M2)÷2]
        × factorial[(J2−M2)÷2] × factorial[(J+M)÷2]
        × factorial[(J−M)÷2]/factorial[(J1+J2+J+2)÷2])
        × cc
  end
end VCC
```

ALGORITHM 253 [F2]
EIGENVALUES OF A REAL SYMMETRIC MATRIX BY THE QR METHOD

P. A. BUSINGER*
(Recd. 17 Aug. 1964, 3 Nov. 1964 and 8 Dec. 1964)
University of Texas, Austin, Texas

* This work was supported in part by the National Science Foundation through grant NSF GP-217 and the Army Research Office through grant DA-ARO(D) 31-124-G388. Thanks are due the referee for suggesting several improvements.

```
procedure symmetric QR 1 (n, g); value n; integer n;
  array g;
comment uses Householder's method and the QR algorithm to
  find all n eigenvalues of the real symmetric matrix whose lower
  triangular part is given in the array g[1:n, 1:n]. The computed
  eigenvalues are stored as the diagonal elements g[i, i]. The
  original contents of the lower triangular part of g are lost during
  the computation whereas the strictly upper triangular part of g
  is left untouched.
```

REFERENCES:
FRANCIS, J. G. F. The QR transformation—Part 2. *Comput. J. 4* (1961), 332-345.
ORTEGA, J. M., AND KAISER, H. F. The LL^T and QR methods for symmetric tridiagonal matrices. *Comput. J. 6* (1963), 99-101.
PARLETT, B. The development and use of methods of LR type. New York U., 1963.
WILKINSON, J. H. Householder's method for symmetric matrices. *Numer. Math. 4*, (1962), 354-361.

TEST RESULTS:
A version of this procedure acceptable to the Oak Ridge ALGOL compiler was tested on a CDC 1604 computer (relative machine precision 1.5_{10}-11). For a number of testmatrices of order up to 64 the dominant eigenvalue was found to at least 8 digits and it was always among the most accurate values computed. In some cases the accuracy of the nondominant eigenvalues varied greatly, in one case the least accurate value had only 4 good digits.

EXAMPLE:
For the 5×5 symmetric matrix whose lower triangular part is

```
5
4  6
3  0  7
2  4  6  8
1  3  5  7  9
```

```
  this prodecure computed the eigenvalues 22.406875305,
  7.5137241530, 4.8489501197, −1.0965951813, 1.3270455994;
begin
  real procedure sum (i, m, n, a); value m, n;
    integer i, m, n; real a;
  begin real s; s := 0;
    for i := m step 1 until n do s := s+a; sum := s
  end sum;
  real procedure max (a, b); value a, b; real a, b;
    max := if a > b then a else b;
  procedure Householder tridiagonalization 1 (n, g, a, bq, norm);
    value n; integer n; array g, a, bq; real norm;
    comment nonlocal real procedure sum, max;
```

```
comment reduces the given real symmetric n by n matrix g
  to tridiagonal form using n−2 elementary orthogonal trans-
  formations (I−2ww′) = (I-gamma uu′). Only the lower tri-
  angular part of g need be given. The diagonal elements and
  the squares of the subdiagonal elements of the reduced matrix
  are stored in a[1:n] and bq[1:n−1] respectively. norm is set
  equal to the infinity norm of the reduced matrix. The columns
  of the strictly lower triangular part of g are replaced by the
  nonzero portions of the vectors u;
begin integer i, j, k; real t, absb, alpha, beta, gamma, sigma;
  array p[2:n];
  norm := absb := 0;
  for k := 1 step 1 until n−2 do
  begin a[k] := g[k, k];
    sigma := bq[k] := sum(i, k+1, n, g[i, k]↑2);
    t := absb+abs(a[k]); absb := sqrt(sigma);
    norm := max(norm, t+absb);
    if sigma ≠ 0 then
    begin alpha := g[k+1, k];
      beta := if alpha < 0 then absb else−absb;
      gamma := 1/(sigma−alpha×beta); g[k+1, k] := alpha−
        beta;
      for i := k+1 step 1 until n do
        p[i] := gamma×(sum(j, k+1, i, g[i, j]×g[j, k])+
          sum(j, i+1, n, g[j, i]×g[j, k]));
      t := 0.5×gamma×sum(i, k+1, n, g[i, k]×p[i]);
      for i := k+1 step 1 until n do p[i] := p[i]−t×g[i, k];
      for i := k+1 step 1 until n do
        for j := k+1 step 1 until i do
          g[i, j] := g[i, j]−g[i, k]×p[j]−p[i]×g[j, k]
    end
  end k;
  a[n−1] := g[n−1, n−1]; bq[n−1] := g[n, n−1]↑2;
  a[n] := g[n, n]; t := abs(g[n, n−1]);
  norm := max(norm, absb+abs(a[n−1])+t);
  norm := max(norm, t+abs(a[n]))
end Householder tridiagonalization 1;
integer i, k, m, m1; real norm, epsq, lambda, mu, sq1, sq2, u,
  pq, gamma, t; array a[1:n], bq[0:n−1];
Householder tridiagonalization 1(n, g, a, bq, norm);
epsq := 2.25₁₀−22×norm↑2; comment The tolerance used in
  the QR iteration depends on the square of the relative ma-
  chine precision. Here 2.25₁₀−22 is used which is appropriate
  for a machine with a 36-bit mantissa;
mu := 0; m := n;
inspect: if m=0 then go to return else i := k := m1 := m−1;
  bq[0] := 0;
  if bq[k] ≦ epsq then
  begin g[m, m] := a[m]; mu := 0; m := k;
    go to inspect
  end;
  for i := i−1 while bq[i] > epsq do k := i;
  if k = m1 then
  begin comment treat 2 × 2 block separately;
    mu := a[m1]×a[m]−bq[m1]; sq1 := a[m1]+a[m];
    sq2 := sqrt((a[m1]−a[m])↑2+4×bq[m1]);
    lambda := 0.5×(if sq1 ≧ 0 then sq1+sq2 else sq1−sq2);
    g[m1, m1] := lambda; g[m, m] := mu/lambda;
    mu := 0; m := m−2; go to inspect
  end;
  lambda := if abs(a[m]−mu) < 0.5×abs(a[m]) then a[m]+0.5×
    sqrt(bq[m1]) else 0.0;
  mu := a[m]; sq1 := sq2 := u := 0;
  for i := k step 1 until m1 do
  begin comment shortcut single QR iteration;
    gamma := a[i]−lambda−u;
    pq := if sq1≠1 then gamma↑2/(1−sq1) else (1−sq2)×
      bq[i−1];
    t := pq+bq[i]; bq[i−1] := sq1×t; sq2 := sq1;
    sq1 := bq[i]/t; u := sq1×(gamma+a[i+1]−lambda);
    a[i] := gamma+u+lambda
  end i;
  gamma := a[m]−lambda−u;
  bq[m1] := sq1×(if sq1≠1 then gamma↑2/(1−sq1) else
    (1−sq2)×bq[m1]);
  a[m] := gamma+lambda; go to inspect;
return: end symmetric QR 1
```

ALGORITHM 254 [F2]
EIGENVALUES AND EIGENVECTORS OF A REAL SYMMETRIC MATRIX BY THE QR METHOD

P. A. BUSINGER*
(Recd. 17 Aug. 1964, 17 Nov. 1964 and 8 Dec. 1964)
University of Texas, Austin, Texas

* This work was supported in part by the National Science Foundation through grant NSF GP-217 and the Army Research Office through grant DA-ARO(D) 31-124-G388. Thanks are due the referee for suggesting several improvements.

```
procedure symmetric QR 2 (n, g, x); value n; integer n;
  array g, x;
comment uses Householder's method and the QR algorithm to
  find all n eigenvalues and eigenvectors of the real symmetric
  matrix whose lower triangular part is given in the array g. The
  computed eigenvalues are stored as the diagonal elements
  g[i, i] and the eigenvectors as the corresponding columns of the
  array x. The original contents of the lower triangular part of g
  are lost during the computation whereas the strictly upper
  triangular part of g is left untouched.
```

REFERENCES:
FRANCIS, J. G. F. The QR transformation—Part 2. *Comput. J.* 4 (1961), 332–345.
PARLETT, B. The development and use of methods of LR type. New York U., 1963.
WILKINSON, J. H. Householder's method for symmetric matrices. *Numer. Math.* 4 (1962), 354–361.

TEST RESULTS:

A version of this procedure acceptable to the Oak Ridge ALGOL compiler was tested on a CDC 1604 computer (relative machine precision 1.5_{10}–11). For a number of testmatrices of order up to 64 the dominant eigenvalue was found to at least 9 digits. Eigenvalues much smaller in magnitude than the dominant eigenvalue have fewer accurate digits. In some cases the components of the eigenvectors were slightly less accurate than the eigenvalues.

EXAMPLE:

For the 5 × 5 symmetric matrix whose lower triangular part is

```
5
4  6
3  0  7
2  4  6  8
1  3  5  7  9
```

this procedure computed the eigenvalues λ_1=22.406875306, λ_2=7.5137241547, λ_3=4.8489501203, λ_4=−1.0965951820, λ_5=1.3270455995, and the corresponding eigenvectors

x_1 = (0.24587793851, 0.30239603954, 0.45321452335, 0.57717715229, 0.55638458400),

x_2 = (0.55096195546, 0.70944033954, −0.34017913315, −0.083410953290, −0.26543567685),

x_3 = (0.54717279573, −0.31256992008, 0.61811207635, −0.11560659356, −0.45549374666),

x_4 = (−0.46935807220, 0.54221219466, 0.54445240360, −0.42586566248, −0.088988503134),

x_5 = (−0.34101304185, 0.11643462042, 0.019590672072, 0.68204303436, −0.63607121400);

```
begin
  real procedure sum (i, m, n, a); value m, n;
    integer i, m, n; real a;
```

```
begin real s;  s := 0;
  for i := m step 1 until n do s := s+a;  sum := s
end sum;
real procedure max (a, b);  value a, b;  real a, b;
  max := if a > b then a else b;
procedure Householder tridiagonalization 2 (n, g, a, b, x, norm);
  value n;  integer n;  array g, a, b, x;  real norm;
  comment nonlocal real procedure sum, max;
comment reduces the given real symmetric n by n matrix g
  to tridiagonal form using n-2 elementary orthogonal trans-
  formations (I-2ww') = (I-gamma uu'). Only the lower
  triangular part of g need be given. The computed diagonal
  and subdiagonal elements of the reduced matrix are stored in
  a[1:n] and b[1:n-1] respectively. The transformations on the
  right are also applied to the n by n matrix x. The columns of
  the strictly lower triangular part of g are replaced by the
  nonzero portion of the vectors u. norm is set equal to the in-
  finity norm of the reduced matrix;
begin integer i, j, k;  real t, sigma, alpha, beta, gamma, absb;
  array p[2:n];
  norm := absb := 0;
  for k := 1 step 1 until n-2 do
  begin a[k] := g[k, k];
    sigma := sum(i, k+1, n, g[i, k]↑2);
    t := absb+abs(a[k]);  absb := sqrt(sigma);
    norm := max(norm, t+absb);  alpha := g[k+1, k];
    b[k] := beta := if alpha < 0 then absb else -absb;
    if sigma ≠ 0 then
    begin gamma := 1/(sigma-alpha×beta);
      g[k+1, k] := alpha-beta;
      for i := k+1 step 1 until n do
        p[i] := gamma×(sum(j, k+1, i, g[i, j]×g[j, k])
          +sum(j, i+1, n, g[j, i]×g[j, k]));
      t := 0.5×gamma×sum(i, k+1, n, g[i, k]×p[i]);
      for i := k+1 step 1 until n do p[i] := p[i]-t×g[i, k];
      for i := k+1 step 1 until n do
        for j := k+1 step 1 until i do
          g[i, j] := g[i, j]-g[i, k]×p[j]-p[i]×g[j, k];
      for i := 2 step 1 until n do
        p[i] := gamma×sum(j, k+1, n, x[i, j]×g[j, k]);
      for i := 2 step 1 until n do
        for j := k+1 step 1 until n do
          x[i, j] := x[i, j]-p[i]×g[j, k]
    end
  end k;
  a[n-1] :=g[n-1,n-1];  a[n] :=g[n,n];  b[n-1] := g[n, n-1];
  t := abs(b[n-1]);
  norm := max(norm, absb+abs(a[n-1])+t);
  norm := max(norm, t+abs(a[n]))
end Householder tridiagonalization 2;
integer i, j, k, m, m1;  real t, norm, eps, sine, cosine, lambda,
    mu, a0, a1, b0, beta, x0, x1;
  array a[1:n], b[0:n], c[0:n-1], cs, sn[1:n-1];
for i := 1 step 1 until n do
begin comment set x equal to the identity matrix;
  x[i, i] := 1;
  for j := i+1 step 1 until n do x[i, j] := x[j, i] := 0
end i;
Householder tridiagonalization 2 (n, g, a, b, x, norm);
eps := norm×1.5₁₀-11;  comment the tolerance used in the
  QR iteration is set equal to the product of the infinity norm
  of the reduced matrix and the relative machine precision
  (here assumed to be 1.5₁₀-11 which is appropriate for a machine
  with a 36-bit mantissa);
b[0] := mu := 0;  m := n;
inspect: if m=0 then go to return else i := k := m1 := m-1;
if abs(b[k]) ≦ eps then
begin
  g[m, m] := a[m];  mu := 0;  m := k;  go to inspect
end;
for i := i-1 while abs(b[i]) > eps do k := i;
lambda := if abs(a[m]-mu) < 0.5×abs(a[m]) ∨ m1=k then
  a[m]+0.5×b[m1] else 0.0;
mu := a[m];  a[k] := a[k]-lambda;  beta := b[k];
for j := k step 1 until m1 do
begin comment transformation on the left;
  a0 := a[j];  a1 := a[j+1]-lambda;  b0 := b[j];
  t := sqrt(a0↑2+beta↑2);
  cosine := cs[j] := a0/t;  sine := sn[j] := beta/t;
  a[j] := cosine×a0+sine×beta;  a[j+1] := -sine×b0+
    cosine×a1;
  b[j] := cosine×b0+sine×a1;  beta := b[j+1];
  b[j+1] := cosine×beta;  c[j] := sine×beta
end j;
b[k-1] := c[k-1] := 0;
for j := k step 1 until m1 do
begin comment transformation on the right;
  sine := sn[j];  cosine := cs[j];
  a0 := a[j];  b0 := b[j];
  b[j-1] := b[j-1]×cosine+c[j-1]×sine;
  a[j] := a0×cosine+b0×sine+lambda;
  b[j] := -a0×sine+b0×cosine;  a[j+1] := a[j+1]×cosine;
  for i := 1 step 1 until n do
  begin x0 := x[i, j];  x1 := x[i, j+1];
    x[i, j] := x0×cosine+x1×sine;  x[i, j+1] := -x0×sine+
      x1×cosine
  end i
end j;
a[m] := a[m]+lambda;  go to inspect;
return: end symmetric QR 2
```

CERTIFICATION OF ALGORITHM 21 [S17] BESSEL FUNCTION FOR A SET OF INTEGER ORDERS

[W. Börsch-Supan, *Comm. ACM 3* (Nov. 1960), 600]

J. Stafford (Recd. 16 Nov. 1964)
Westland Aircraft Ltd., Saunders-Roe Division, East Cowes, Isle of Wight, Eng.

If this procedure is used with a combination of a moderately small argument and a moderately large order, the recursive evaluation of *rec*2 in the last line of the first column can easily lead to overflow. This occurred, for instance, in trying to evaluate $J_{10}(0.01)$.

The following alterations correct this:

(i) Declare a **real** variable *z* and an **integer** variable *m*;

(ii) After line *rec* insert:

```
z := MAX/4 × abs (x/k);
comment MAX is a large positive number approaching in
  size the largest number which can be represented. The nu-
  merical value of MAX/4 is written into the procedure;
```

(iii) At the end of the first column insert:

```
if abs(rec2) > z then
begin
  rec1 := rec1/z;  rec2 := rec2/z;  sum := sum/z;
  for m := n step -1 until p - 1 do J[m] := J[m]/z
end;
```

With these alterations the procedure was run on a National-Elliott 803, for $x = -1, 0, 0.01, 1, 10$ and $n = 0, 1, 2, 10, 20$. The results agreed exactly with published seven-place tables.

[See also Algorithm 236, Bessel Functions of the First Kind (*Comm. ACM 7* (Aug. 1964), 479) which is not restricted to integer values. Although it is a much more complicated program, Algorithm 236 is slightly faster than Algorithm 21 as corrected, at least in some cases.—Ed.]

REMARK ON ALGORITHM 231 [F1] MATRIX INVERSION

[J. Boothroyd, *Comm. ACM 6* (June 1964), 347]

MATS FERRING (Recd. 23 Nov. 1964)
Flygmotor Aeroengine Company, Trollhättan, Sweden

The algorithm cannot accept the pivot element = 0 which reduces the detection of singularities. We suggest the correction:

if $k > i \wedge j > i \wedge abs(a[r[k], c[j]]) > abs(pivot)$ **then**

should be

if $k > i \wedge j > i \wedge abs(a[r[k], c[j]]) \geqq abs(pivot)$ **then**

Revised Algorithms Policy • May, 1964

A contribution to the Algorithms department must be in the form of an algorithm, a certification, or a remark. Contributions should be sent in duplicate to the editor, typewritten double-spaced in capital and lower-case letters. Authors should carefully follow the style of this department, with especial attention to indentation and completeness of references. Material to appear in **boldface** type should be underlined in black. Blue underlining may be used to indicate *italic* type, but this is usually best left to the Editor.

An algorithm must be written in the ALGOL 60 Reference Language [*Comm. ACM 6* (Jan. 1963), 1–17], and normally consists of a commented procedure declaration. Each algorithm must be accompanied by a complete driver program in ALGOL 60 which generates test data, calls the procedure, and outputs test answers. Moreover, selected previously obtained test answers should be given in comments in either the driver program or the algorithm. The driver program may be published with the algorithm if it would be of major assistance to a user.

Input and output should be achieved by procedure statements, using one of the following five procedures (whose body is not specified in ALGOL): [see "Report on Input-Output Procedures for ALGOL 60," *Comm, ACM 7* (Oct. 1964), 628–629].

procedure *inreal* (*channel, destination*); **value** *channel*; **integer** *channel*; **real** *destination*; **comment** the number read from channel *channel* is assigned to the variable *destination*; . . . ;

procedure *outreal* (*channel, source*); **value** *channel, source*; **integer** *channel*; **real** *source*; **comment** the value of expression *source* is output to channel *channel*; . . . ;

procedure *ininteger* (*channel, destination*); **value** *channel*; **integer** *channel, destination*; . . . ;

procedure *outinteger* (*channel, source*); **value** *channel, source*; **integer** *channel, source*; . . . ;

procedure *outstring* (*channel, string*); **value** *channel*; **integer** *channel*; **string** *string*; . . . ;

If only one channel is used by the program, it should be designated by 1. Examples:

outstring (1, '*x* ='); *outreal* (1, *x*);
for *i* := 1 **step** 1 **until** *n* **do** *outreal* (1, *A*[*i*]);
ininteger (1, *digit* [17]);

It is intended that each published algorithm be a well-organized, clearly commented, syntactically correct, and a substantial contribution to the ALGOL literature. All contributions will be refereed both by human beings and by an ALGOL compiler. Authors should give great attention to the correctness of their programs, since referees cannot be expected to debug them. Because ALGOL compilers are often incomplete, authors are encouraged to indicate in comments whether their algorithms are written in a recognized subset of ALGOL 60 [see "Report on SUBSET ALGOL 60 (IFIP)," *Comm. ACM 7* (Oct, 1964), 626–627].

Certifications and remarks should add new information to that already published. Readers are especially encouraged to test and certify previously uncertified algorithms. Rewritten versions of previously published algorithms will be refereed as new contributions, and should not be imbedded in certifications or remarks.

Galley proofs will be sent to the authors; obviously rapid and careful proofreading is of paramount importance.

Although each algorithm has been tested by its author, no liability is assumed by the contributor, the editor, or the Association for Computing Machinery in connection therewith.

Letters—continued from p. 202

On Computers and Programs; Copyrights and Patents

Dear Editor:

I read with great interest your series of articles entitled "Computers and Programs; Copyrights and Patents" which appeared in the October 1964 issue of the *Communications*. Although I am not yet a member of the Bar, as are the authors of those articles, I beg leave to offer a few comments of my own. By way of my qualifications to speak on this subject, I mention that a legal paper I prepared was largely responsible for the Copyright Office's recent decision to register copyrights on computer programs and that I received the first such copyrights. This paper, the only complete legal analysis of the problems of copyright protection for computer programs published so far, appeared in the November 1964 issue of *Columbia Law Review*. Copies of the article may be obtained at no cost by writing to the author at the address given below.

Despite our many areas of complete agreement and my respect for Mr. Lawlor's opinions in this admittedly difficult legal area, I must take issue with his suggestion that copyright protection of a program would not preclude a reproduction in nonreadable form, e.g. a magnetic tape. As my paper points out, the piano roll case which he cites and the magnetic tape reproduction situation may be distinguished on several grounds. Moreover, since computer programs may now be copyrighted in the form of magnetic tapes, there seems to be little fear that a court would avoid finding that a second tape recording, identical in every way with the original recording, is an infringement within the meaning of the copyright law.

Mr. Hamlin and Mr. Jacobs both argue forcefully that computer programs should have some form of legal protection and that they should be patentable. Although I am fully in accord with their first point, I would like to suggest that copyright protection would be preferable from the point of view of both the programmer and the computer industry. Patents are expensive, take several years to secure, have a high mortality rate in the courts, and are available only to inventions representing a high degree of creativity and novelty. By contrast, copyrights are inexpensive, offer immediate protection, are favored by the courts, and require little showing of creativity. In return, they offer substantial protection and do not require a wide public disclosure. From the point of view of the data processing community, they also seem to be preferable. It would be illegal for anyone to use a patented program during the 17-year monopoly without the patentee's permission. On the other hand, anyone would be free to create a program similar to a copyrighted one if only he didn't copy from the copyrighted program; a freedom he would not have with respect to a patented program. It is fair, I think, to ask whether the advantages to be gained from patent monopolies on programs would be commensurate with the restrictions on other programmers which must follow as a matter of law.

Whatever my areas of disagreement with these articles, the authors and the editors of ACM are to be congratulated for keeping their readers informed in this important area.

JOHN F. BANZHAF III
Columbia Law Review
435 West 116 Street
New York, N.Y.