spectrum was a more important consideration than the error at a single point.

In the study of hydrolyzed polynucleotides it is difficult to obtain enough of the pure component nucleotides to weigh and thus measure the spectra at known concentrations. The effect of concentration can be eliminated with normalization by dividing each point by the vector length, the area under the curve or the absorbance at a fixed wavelength (e.g. 2600). The solution of the normalized mixture vector is then obtained in fractional parts.

This linear programming method which minimizes the small, nonsystematic, experimental errors leads to a much improved numerical solution of the chemically known polynucleotides over the classical least squares methods. The advantages appear to lie in disallowing negative solutions and in the choosing of significant points on which to base the solution while allowing the possibility of small positive or negative errors to be reflected as slack.


## ACKNOWLEDGMENT

The authors wish to thank Dr. Philip Wolfe of The RAND Corporation for helpful suggestions as to the formulation of the problem.


## REFERENCES


1. MELLON, M. G. (ED.) *Analytical Absorption Spectroscopy*, Ch. 7, 350ff. John Wiley (1950).
2. STERNBERG, J. C., STILLS, H. S., AND SCHWENDEMAN, R. H. Spectrophotometric analysis of multi-component systems using the least squares method in matrix form. *Anal. Chem. 32* (1960), 84–90.
3. REID, J. C., AND PRATT, A. W. Vector analysis of ultraviolet mixture spectra: the composition of ribonucleic acid. *Biochem. Biophys. Res. Com. 3* (1960), 337–342.


A contribution to this department must be in the form of an Algorithm, a Certification, or a Remark. Contributions should be sent in duplicate to the Editor and should be written in a style patterned after recent contributions appearing in this department. An algorithm must be written in ALGOL 60 (see *Communications of the ACM*, January 1963) and accompanied by a statement to the Editor indicating that it has been tested and indicating which computer and programing language was used. For the convenience of the printer, contributors are requested to double space material and underline delimiters and logical values that are to appear in boldface type. Whenever feasible, Certifications should include numerical values.

Although each algorithm has been tested by its contributor, no warranty, express or implied, is made by the contributor, the Editor, or the Association for Computing Machinery as to the accuracy and functioning of the algorithm and related algorithm material, and no responsibility is assumed by the contributor, the Editor, or the Association for Computing Machinery in connection therewith.



# Algorithms


J. H. WEGSTEIN, Editor


## ALGORITHM 150
## SYMINV2


H. RUTISHAUSER
Eidg. Technische Hochschule, Zurich, Switzerland


```
procedure syminv2(a,n) result: (a) exit: (fail); value n; integer n; array a; label fail;
  comment syminv2 obtains inverse of a symmetric matrix a of order n by a method which is similar to that given by Busing and Levy [Comm. ACM 5 (1962), 446] but requires no interchanges of rows and columns nor storage space for an additional matrix Q, yet is numerically equivalent. The procedure requires the upper triangular part of a to be given and overwrites it by the upper triangular part of the inverse which is again denoted by a. All pivots are chosen on the diagonal, and if all further diagonal elements which are eligible as pivots vanish (this is impossible for a positive definite matrix a) then exit through fail occurs;
  begin
    real bigajj;
    integer i, j, k;
    real array p, q[1:n];
    Boolean array r[1:n];
    for i := 1 step 1 until n do r[i] := true;
grand loop:
    for i :=1 step 1 until n do
    begin
search for pivot:
      bigajj := 0;
      for j := 1 step 1 until n do
      begin
        if r[j] ∧ abs(a[j,j]) > bigajj then
        begin
          bigajj := abs(a[j,j]);
          k := j
        end;
      end;
      if bigajj = 0 then go to fail;
preparation of elimination step i:
      r[k] := false;
      q[k] := 1/a[k,k];
      p[k] := 1;
      a[k,k] := 0;
      for j := 1 step 1 until k-1 do
      begin
        p[j] := a[j,k];
        q[j] := (if r[j] then -a[j,k] else a[j,k]) × q[k];
        a[j,k] := 0
      end;
      for j := k+1 step 1 until n do
      begin
        p[j] := if r[j] then a[k,j] else -a[k,j];
        q[j] := -a[k,j] × q[k];
        a[k,j] := 0
      end;
elimination proper:
      for j := 1 step 1 until n do
```

```
        for k := j step 1 until n do
          a[j,k] := a[j,k] + p[j] × q[k]
    end grand loop
end syminv2
```

## ALGORITHM 151
## LOCATION OF A VECTOR IN A LEXICOGRAPHICALLY ORDERED LIST

HENRY F. WALTER
United States Steel Corp., Applied Research Laboratory, Monroeville, Penn.

```
integer procedure LOCATE (min, n, c, v, combinatorial);
value v; integer min, n, c; integer array v;
integer procedure combinatorial;
comment This procedure locates the position, LOCATE, of a
  given vector in a list of vectors without searching the list. The
  list consists of all the combinations of n consecutive digits taken
  d at a time. Min is the smallest of the n integers. Each vector
  (combination) is written in ascending order from left to right,
  as, for example, 378 and the vectors are listed lexicographically,
  by which is meant, that, considered as d digit numbers, the
  vectors are listed in ascending order. For example, with min = 1,
  d = 3, and n = 6, the vectors in order are 123, 124, 125, 126, 134,
  135, ..., 456. Given the vector, v = 356, the procedure locates
  this vector as the 19th in the list;
begin integer i, r, max, part, whole;
  r := 1; v [0] := min − 1; max := min − 1+n;
  for i := 0 step 1 until c−1 do
  begin part := c − i − 1;
    ask: if v[i+1] − v[i] > 1 then
      begin whole := max − v[i] − 1;
        r := r + combinatorial (whole, part);
        v[i] := v[i] + 1;
        go to ask
      end;
  end;
  locate := r
end;
```

## ALGORITHM 152
## NEXCOM

JOHN HOPLEY
Peat, Marwick, Mitchell & Co., London, England

```
procedure nexcom (char, n, setcomplete, nullvector);
array char; integer n;
label setcomplete, nullvector;
comment char is a column vector containing n elements each of
  which is either 1 or 0. Nexcom transforms char into another
  vector containing the same number of 1's and 0's, but in a
  different sequence. Starting with char in the state of having 1
  in each of the element positions 1, ..., r and zeros elsewhere then
  repeated application of nexcom generates all nCr patterns of
  char. The procedure terminates if the presented vector char has
  1 in each of the positions n, n−1, ... n−r+1 and zeros else-
  where. Termination is indicated by exit through the formal label
  'setcomplete'. If char is the null vector then procedure exists
  through the formal label 'nullvector';
begin integer n, p, m;
comment find the first 1 in char;
for n := 1 step 1 until N do if
char [n] = 1 then go to A;
go to nullvector;
comment how many adjacent 1's;
A: p := 0;
for m := n + 1 step 1 until N do
if char [m] = 1 then p := p + 1 else go to B;
comment Have all combinations been generated;
B: if p + n = N then go to setcomplete;
comment Set up next combination; char[n+p+1] := 1;
for m := n + p step − 1 until n do char [m] := 0;
for m := 1 step 1 until p do char [m] := 1;
end nexcom;
```

## ALGORITHM 153
## GOMORY

F. L. BAUER
Johannes Gutenberg-Universität, Mainz, Germany

```
procedure Gomory (a, m, n) result: (a) exit: (no solution);
  value m, n;
  integer m, n;
  integer array a;
  label no solution;
comment Gomory algorithm for all-integer programming. The
  objective of this procedure is to determine the integer solution
  of a linear programming problem with integer coefficients only.
  The tableau-matrix a consists of m + 1 rows and n columns.
  The top row of a is the objective row, the last column represents
  the right-hand sides. The tableau-columns, with the exception
  of the last column, have to be lexicographically positive. The
  algorithm is finished if all entries in the last column, except the
  top most entry, are nonnegative. Then the top most entry of
  the last column represents the value of the objective function.
  The other entries of the last column define the coordinates of
  the optimal solution. There are always the same variables con-
  nected with the same rows. The exit no solution is used if a row
  is found which has a negative entry in the last column, but
  otherwise only nonnegative entries;
begin integer i, k, j, l, r;
      real lambda;
      integer array t[1:n−1], c[1:n];
1:    for i := 1 step 1 until m do if a[i,n] < 0 then
      begin r := i; go to 2 end;
      go to end;
2:    for k := 1 step 1 until n−1 do if a[r,k] < 0 then
      go to 4;
      go to no solution;
4:    l := k;
      for j := k+1 step 1 until n−1 do if a[r,j] < 0 then
        begin i := 0;
          3: if a[i,j] < a[i,l] then l := j else
             if a[i,j] = a[i,l] then
             begin i := i+1; go to 3 end
        end;
      for j := 1 step 1 until n−1 do if a[r,j] < 0 then
        begin if a[0,l] ≠ 0 then t[j] := entier(a[0,j]/a[0,l])
          else t[j] := 1
        end;
      lambda := abs(a[r,1]/t[1]);
      for j := 2 step 1 until n−1 do if a[r,j] < 0 then
        begin if abs(a[r,j]/t[j]) > lambda then
          lambda := abs(a[r,j]/t[j]) end;
      for j := 1 step 1 until n do if j≠l then
        begin c[j] := entier(a[r,j]/lambda);
          if c[j] ≠ 0 then
          for i := 0 step 1 until m do a[i,j] := a[i,j] + c[j] ×
            a[i,l]
        end;
      go to 1;
end: end;
```

## CERTIFICATION OF ALGORITHM 32 MULTINT [R. Don Freeman, *Comm. ACM*, Feb. 1961]

HENRY C. THACHER, JR.*
Reactor Engineering Div., Argonne National Laboratory, Argonne, Ill.

* Work supported by the U. S. Atomic Energy Commission.

The procedure was transcribed into the ACT-III language for the LGP-30 computer, and was tested on the integrals:

$$(1)\quad \int_0^1\int_0^1\int_0^1\int_0^1 k[\cos u - 7u\sin u - 6u^2\cos u + u^3\sin u]\,dw\,dx\,dy\,dz = \sin k$$

where $u = kwxyz$, and

$$(2)\quad \int_0^1\int_0^{\sqrt{1-x^2}}\int_0^{\sqrt{1-x^2-y^2}} \frac{dz\,dy\,dx}{x^2+y^2+(z-k)^2} = \pi\left(2+\frac{1}{2}\left(\frac{1}{k}-k\right)\log\left|\frac{1+k}{1-k}\right|\right).$$

The ALGOL procedures for the second integral are:

```
real procedure Low (j,x);
Low := 0;
real procedure Upp(j,x); comment z ≡ x[3], y ≡ x[2], x ≡
  x(1);
begin
integer i; real temp;
temp := 1.0;
for i := j-1 step - 1 until 1 do
temp := temp - x[j] × x[j];
Upp := sqrt(temp)
end;
real procedure Funev(j,x);
comment The real parameter k is global;
Funev := if j < 3 then 1.0 else 1/(x[1]×x[1]+x[2]×x[2]+(x[3]-k)
  ↑2);
```

The first integral was tested only with $s[j] = 1$, and with various Gaussian formulas for integrals over the interval $(-1,+1)$. Results were as follows:

| $k$ | $\pi/2$ | $\pi$ | $3\pi/2$ | $2\pi$ |
|---|---|---|---|---|
| **true** | 1.0000000 | 0.0000000 | −1.0000000 | 0.0000000 |
| p = 2 | 0.993704 | −0.0333603 | +0.020166 | 6.881490 |
| p = 3 | 1.000032 | 0.0000848 | −1.061651 | −0.597419 |
| p = 4 | 0.999999 | 0.0000001 | −0.998407 | +0.0027035 |
| p = 5 | 1.000000 | −0.0000002 | −1.000028 | −0.0007857 |

For the second integral, two values of $s = s[1] = s[2] = s[3]$ were used, and two values of $p$. Results were as follows:

| $k$ | 1/2 | | 2 | |
|---|---|---|---|---|
| **true** | 11.46027376 | | 1.10609687 | |
| s | 1 | 2 | 1 | 2 |
| p = 2 | 5.454460 | 11.838651 | 1.0368770 | 1.1184305 |
| p = 3 | 9.361666 | 12.408984 | 1.1343551 | 1.1094278 |

The effect of the pole at $(0,0,k)$ is obvious.

For the algorithm to run in any compiler, the semicolon following $x[T]$; in the fourth line above the end of the comment must be deleted. The array bounds on the arrays $r$ and $d$ must be increased to $[1 : T+1]$.

For a system which permits variable array bounds, the introduction of the integer $T$ appears superfluous. For such a system, $T$ may be replaced by $n$ throughout with a probable gain in efficiency. For most translators, the presence of undefined elements in an array will not cause difficulties, provided these elements do not appear in an expression before they are assigned a value. The statement "**for** $j := 1$ **step** 1 **until** $T$ **do** $x[j] := 0.0$;" is thus superfluous. The semicolon before the **end** which precedes the label "*sum*" also appears unnecessary.

In spite of these minor corrections, the algorithm appears to be extremely convenient for multiple quadratures over arbitrary regions using the Cartesian product of any explicit one-dimensional formula (and not merely a Gaussian formula) for integrating over the range $[-1,1]$. If endpoints are used in the formula, it will, of course, repeat the calculation for each section of the range.

## CERTIFICATION OF ALGORITHM 73 INCOMPLETE ELLIPTIC INTEGRALS [David K. Jefferson, *Comm. ACM 4*, Dec. 1961]

NOELLE A. MEYER
E. I. du Pont de Nemours & Co., Wilmington, Del.

*Ellint* was hand-coded in FORTRAN for the IBM 7070. The following corrections were made:

The statement

```
E := (2×n-1)/(2×N);
```

should be

```
E := (2×n-1)/(2×n);
```

The statement

```
F := abs(k) × sqrt(1-sinphi↑2) × (1-k↑2×sinphi↑2) ↑
                                        ((2×n-1)/(2×n));
```

should be

```
F := (abs(k)×sqrt(1-sinphi↑2) ×
                  (1-k↑2×sinphi↑2) ↑ (n-.5))/(2×n)
```

The statement

```
L[2] := L[1] + 1/(n×2×n-1));
```

should be

```
L[2] := L[1] + (1/(n×(2×n-1));
```

In order to accommodate negative $\phi$ the following changes were made:

The statement

```
if abs((sigma[1]+del[1])-sigma[1]) > 0 ∧ phi × sinphi ↑
                                  (2×n) ≥ A[2] then go to step 1;
```

was changed to

```
if abs((sigma[1]+del[1])-sigma[1])>0∧abs(phi×sinphi↑(2×n))
                                  ≥ abs(A[2]) then go to step 1;
```

Also the following was inserted before the last statement (*stop*: **end**)

```
        if phi < 0 then go to wait else go to stop;
wait:   F := -F;
        E := -E;
```

The revised algorithm yielded satisfactory answers when compared with the DiDonato and Hershey tables. Differences occurred in the eighth significant digit as shown in the following difference tables.

### DIFFERENCE TABLES

*F*-TABLE

| $\phi$ (in degrees) \ $\theta$ (in degrees) | 0 | 30 | 60 | 90 |
|---|---|---|---|---|
| 0 | 0. | 0. | 0. | 0. |
| 30 | $-1\times10^{-8}$ | $-1\times10^{-8}$ | $-1\times10^{-8}$ | $-3\times10^{-8}$ |
| 60 | $1\times10^{-8}$ | $1\times10^{-8}$ | $2\times10^{-8}$ | $-3\times10^{-8}$ |
| 90 | 0. | $2\times10^{-8}$ | $6\times10^{-8}$ | 0. |

*E*-TABLE

| | | | | |
|---|---|---|---|---|
| 0 | 0. | 0. | 0. | 0. |
| 30 | $-1\times10^{-8}$ | $-1\times10^{-8}$ | $-1\times10^{-8}$ | $-1\times10^{-8}$ |
| 60 | $1\times10^{-8}$ | $1\times10^{-8}$ | $-7\times10^{-8}$ | $3\times10^{-8}$ |
| 90 | 0. | 0. | $1\times10^{-8}$ | 0. |