METHOD II. Suppose $d_i = p_iC$ and $p_i \geqq 0$ for $i = 1, 2, \cdots, m$. This case can have a binary coded table consisting of two rows. The first row, s, is used to keep a record of which sequence value has been generated by using the second row. Let $t_i = p_i$ if $p_i \neq 0$; otherwise let $t_i = 1$. Let $B_{sj} = 1$ for $j = t_1, t_1 + t_2, \cdots, t_1 + t_2 + \cdots + t_m$. Let $B_{2j} = 1$ for $j = 1 + \sum_{i=1}^{k-1} t_i, \cdots, p_k + \sum_{i=1}^{k-1} t_i$ where $k = 1, 2, \cdots, m$. The s row contains m bits while the second row contains $\sum_{i=1}^{m} p_i$ bits. This method requires $2\sum_{i=1}^{m} t_i$ bit positions of memory. For an example of this method, Sequence B can have the binary coded table given in Table I by using Rows 8 and 9, where 8 is the s row. In this case the coding to find x_n, given n, is done so that the x_0 accumulator is incremented by C according to the second row, to the nth bit in the s row. If $p_i \neq 0$ for $i = 1, 2, \cdots, m$ the second row would not have a blank position and therefore by suitable programming would not be necessary.

Suppose $d_i = D + p_iC$ and $p_i \geqq 0$ for $i = 1, 2, \cdots, m$. This sequence could have the same binary coded table as $d_i = p_iC$ and $p_i \geqq 0$ for $i = 1, 2, \cdots, m$ if in the program D was added to the x_0 accumulator for each $B_{sj} = 1$ where $j \leqq n$.

It is possible to use one s row when binary coding q sequences using Method II. Let $L_i = \max(t_{1i}, \cdots, t_{qi})$ where t_{ji} is t_i of the jth sequence. Let $B_{sj} = 1$ for $j = L_1, L_1 + L_2, \cdots, L_1 + L_2 + \cdots + L_m$. Let $B_{Hn} = 1$ for $n = 1 + \sum_{i=1}^{k-1} L_i, \cdots, p_k + \sum_{i=1}^{k-1} L_i$ where $k = 1, 2, \cdots, m$ and for $H = 1, 2, \cdots, q$. For example, Sequences B, C and D can be generated using in Table I Rows 11, 12, 13 and 14, where Row 11 is the s-row. This same procedure could be followed to binary code $d_i = t_{1i}C_1 + t_{2i}C_2 + \cdots + t_{qi}C_q$, where $t_{ji} \geqq 0$ and $i = 1, 2, \cdots, m$.

TABLE I

Sequences:

A	*B*	*C*	*D*
0	1	16	119
1	3	16	120
2	4	18	122
3	7	24	122
5	8	24	123
7	10	26	125
10	12	30	127
11	13	32	128
14	13	34	129
16	15	36	129
18			
21			
22			

1	×	×	×				×					×				
2				×	×				×	×						
3						×		×			×					
4																
5	×	×	×			×	×	×			×	×				
6				×	×	×		×	×	×	×					
7																
8		×	×			×	×		×		×	×	×		×	
9	×	×	×	×	×	×	×	×	×	×	×	×		×	×	
10																
11		×		×			×	×		×		×	×	×		×
12	×	×	×		×	×	×	×	×	×	×	×	×		×	×
13			×		×	×	×		×		×	×	×	×	×	
14	×		×	×				×	×	×	×	×	×	×		

MAVIS GRIEBROK
22750 Myrtle St.
Hayward, California

RECEIVED MAY, 1963

Algorithms

G. E. FORSYTHE, Editor

ALGORITHM 223
PRIME TWINS

M. SHIMRAT (Recd 7 June 1963; in final form 2 Jan. 1964)
University of Alberta, Calgary, Alberta, Canada

```
procedure Prime Twins (t, Twin1, Twin2, Storage, Act);
  value Storage; integer t, Twin1, Twin2, Storage;
  procedure Act;
comment This procedure will generate successive "prime
  twins," i.e. pairs of primes Twin1, Twin2 which differ by 2.
  Storage is the maximum number of primes that can be stored.
  Act is any procedure for recording, examining, or utilizing each
  pair of twins as it is generated. t is a serial number for the
  twins. P[Storage] ↑ 2 is the last number examined;
begin integer array P[1: Storage]; integer j, m, previous,
    current;
  comment P[j] is the jth prime;
  P[1] := 2; P[2] := 3; j := 2; previous := 3; t := 0;
  for current := 5 step 2 until P[j] × P[j] do
  begin m := 1; for m := m + 1 while P[m] × P[m] ≦ cur-
    rent do
    if current = (current ÷ P[m]) × P[m] then go to NoPrime;
    comment If this point is reached, current is not divisible by
      any prime up to sqrt(current) and so is a prime. We now
      record the new prime, if storage permits, then check if it
      is the second of twins;
    if j < Storage then
      begin j := j + 1; P[j] := current
      end;
    if current = previous + 2 then
      begin t := t + 1; Twin1 := previous; Twin2 := current;
        Act (t, Twin1, Twin2)
      end;
    previous := current;
    NoPrime:
  end;
end procedure Prime Twins
```

ALGORITHM 224
EVALUATION OF DETERMINANT

LEO J. ROTENBERG
(Recd 7 Oct. 1963; in final form 20 Dec. 1963)
Box 2400, 362 Memorial Dr., Cambridge, Mass.

```
real procedure determinant (a, n);
  value n; real array a; integer n;
comment This procedure evaluates a determinant by triangulari-
  zation. The matrix supplied by the calling procedure is modified
  by this program. This procedure is an extensive revision and
  correction of Algorithm 41;
begin real product, factor, temp, div, piv, abpiv, maxpiv;
  integer ssign, i, j, r, imax;
  ssign := 1; product := 1.0;
  for r := 1 step 1 until n−1 do
  begin maxpiv := 0.0;
    for i := r step 1 until n do
```

```
      begin piv := a[i, r];
        abpiv := abs(piv);
        if abpiv > maxpiv then
        begin maxpiv := abpiv;
          div := piv;
          imax := i
        end
      end;
      if maxpiv ≠ 0.0 then
      begin if imax = i then go to resume else
        begin for j := r step 1 until n do
          begin temp := a[imax, j];
            a[imax, j] := a[r, j];
            a[r, j] := temp
          end;
          ssign := - ssign;
          go to resume
        end
      end;
      determinant := 0.0;
      go to return;
resume: for i := r+1 step 1 until n do
      begin factor := a[i, r]/div;
        for j := r+1 step 1 until n do
        a[i, j] := a[i, j] - factor × a[r, j]
      end
    end;
    for i := 1 step 1 until n do
    product := product × a[i, i];
    comment Exponent overflow or underflow will most likely
      occur here if at all. For large or small determinants the user
      is cautioned to replace this with a call to a machine-language
      product routine which will handle extremely large or small
      real numbers;
    determinant := ssign × product;
return:
end
```

CERTIFICATION OF ALGORITHM 182 NONRECURSIVE ADAPTIVE INTEGRATION [W. M. McKeeman and Larry Tesler, *Comm. ACM 6* (June 1963), 315]

HAROLD S. BUTLER (Recd 8 Nov. 1963; rev. 6 Dec. 1963)
Stanford Linear Accelerator Center, Stanford, Calif.

A BALGOL transliteration of *Simpson* has been prepared at Stanford by its authors and it has been used in a number of problems involving numerical integration. Its value was most strikingly displayed when it was utilized in a triple integral in which the final integration was over a strongly peaked function that spanned seven orders of magnitude. *Simpson* effectively minimized the number of evaluations and completed the integration five times faster than alternate schemes to subdivide the region of interest. The values of the integral agreed with independent calculations well within the required tolerance.

The following changes should be made to the published algorithm:

Line 13 should be changed to:

```
lvl := 0;  absarea := est := 1.0;  da := b-a;
```

Line 17 should read:

```
sx := dx[lvl]/6.0;  F1 := 4.0 × F(a + dx[lvl]/2.0);
```

Line 20 should read:

```
epsp[lvl] := eps;  F4[lvl] := 4.0 × F(x3[lvl] + dx[lvl]/2.0);
```

The condition of line 27 should be changed to:

```
if ((abs(est-sum) ≤ epsp[lvl] × absarea) ∧ (est ≠ 1.0)) ∨
  (lvl ≧ 30) then
```

CERTIFICATION OF ALGORITHMS 191 AND 192 HYPERGEOMETRIC AND CONFLUENT HYPERGEOMETRIC [A. P. Relph, *Comm. ACM 6* (July 1963), 388]

HENRY C. THACHER, JR.* (Recd 2 Dec. 1963)
Argonne National Laboratory, Argonne, Ill.

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

The bodies of these two procedures were transcribed for the Dartmouth SCALP processor for the LGP 30 computer. No syntactical errors were found, and the programs gave results agreeing within roundoff (7D) with tabulated values for the following special cases: ${}_2F_1(0.5, 0.5; 1; k^2) = (2/\pi)\,K(k)$; ${}_2F_1(0.5, -0.5; 1; k^2) = (2/\pi)E(k)$ where K and E are complete elliptic integrals of the first and second kinds; ${}_1F_1(.5; 1; iy) = J_0(x)$, and with ${}_1F_1(-1; 0.1; x)$; ${}_1F_1(-0.5; 0.1; x)$, and ${}_1F_1(-0.5; 0.5; x)$.

It should be observed that the function calculated by 191 is ${}_2F_1(a, b; c; z)$, not ${}_1F_2(a, b; c; z)$ as stated in the comment. These programs evaluate the functions by direct summation of the hypergeometric series. They are, therefore, relatively general, but inefficient. Precautions must also be taken against attempting to compute outside the range of effective convergence of the series.

CERTIFICATION OF ALGORITHM 222 INCOMPLETE BETA FUNCTION RATIOS [Walter Gautschi, *Comm. ACM 7* (March 1964), 143]

WALTER GAUTSCHI (Recd 2 Jan. 1964)
Purdue Univ., Lafayette, Ind.

```
begin integer n; array I1, I2, I3[0: 10];
comment This program calls the procedures Incomplete beta q
  fixed and Incomplete beta p fixed to calculate test values of
  I.4(.5+n, 7), I.4(5, 1+n), I.8(5, 1+n) for n = 0(1)10 to 6
  significant digits. The following results were obtained on the
  CDC 1604-A computer, using the Oak Ridge ALGOL compiler:
```

n	$I_{.4}(.5+n, 7)$	$I_{.4}(5, 1+n)$	$I_{.8}(5, 1+n)$
0	.99143646185	.010239999997	.32768000004
1	.93951533330	.040959999972	.65536000000
2	.83567307612	.096255999927	.85196799999
3	.69444760641	.17367039987	.94371839999
4	.54111709640	.26656767980	.98041856000
5	.39800862042	.36689674211	.99363061758
6	.27831789503	.46722580441	.99803463679
7	.18624810627	.56182177742	.99941875711
8	.11995785836	.64695815314	.99983399319
9	.074724512738	.72074301208	.99995395031
10	.045203802963	.78272229360	.99998753828

```
All results are in agreement with those tabulated in [1];
  Incomplete beta q fixed (.4, .5, 7, 10, 6, I1);
  Incomplete beta p fixed (.4, 5, 1, 10, 6, I2);
  Incomplete beta p fixed (.8, 5, 1, 10, 6, I3);
  for n := 0 step 1 until 10 do
    write ( I1[n], I2[n], I3[n])
  end Driver incomplete beta function ratios
```

In the original publication of the algorithm, the following correction of a printer's error is needed in the real procedure *Isubx p and q small*. The statement labelled *L0* should read as follows:

$$u := (k - q + 1) \times x \times u/(k + 1);$$

[1] PEARSON, K. *Tables of the Incomplete Beta-Function.* Cambridge University Press, London, 1934.