Algorithms

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.

CERTIFICATION OF ALGORITHM 13 COMPLEX EXPONENTIAL INTEGRAL (A. Beam, *Comm. ACM*, July, 1960)

P. J. Rader and Henry C. Thacher, Jr.*
Argonne National Laboratory, Argonne, Illinois

EKZ was programmed by hand for the Royal-Precision LGP-30 computer, using a 28-bit mantissa floating-point interpretive system (24.2 modified). To facilitate comparison with existing tables (National Bureau of Standards Applied Mathematics Series 51 and 37), the real and imaginary parts of $E_k(z)$ were computed from u and v. Results are shown in the following table. In all cases, the values agreed with tabulated values within the tolerance specified.

x	y	k	ϵ	n
1×10^{-8}	1.0	1	10^{-1}	7
1×10^{-8}	1.0	1	10^{-2}	14
1×10^{-8}	1.0	1	10^{-3}	24
1×10^{-8}	1.0	1	10^{-4}	37
1×10^{-8}	1.0	1	10^{-5}	52
1×10^{-8}	1.0	1	10^{-6}	70
1×10^{-8}	1.0	1	10^{-7}	90
1×10^{-8}	1.0	1	10^{-8}	114
1×10^{-8}	2.0	1	10^{-6}	37
1×10^{-8}	3.0	1	10^{-6}	26
1×10^{-8}	4.0	1	10^{-6}	21
1.0	1×10^{-8}	1	10^{-6}	40
1.0	1.0	1	10^{-6}	34
1.0	2.0	1	10^{-6}	26
1.0	3.0	1	10^{-6}	21
2.0	1×10^{-8}	1	10^{-6}	23
2.0	1.0	1	10^{-6}	22
2.0	2.0	1	10^{-6}	20
2.0	3.0	1	10^{-6}	17
3.0	1×10^{-8}	1	10^{-6}	17
3.0	1.0	1	10^{-6}	17
3.0	2.0	1	10^{-6}	16
3.0	3.0	1	10^{-6}	15
4.0	0.0	0	10^{-6}	20
4.0	0.0	1	10^{-6}	15
4.0	0.0	2	10^{-6}	16
4.0	0.0	3(1)14	10^{-6}	17
4.0	0.0	15, 16	10^{-6}	16

It thus appears that the algorithm gives satisfactory accuracy, but that in certain ranges of the variables, the time required may be excessive for extensive use.

REMARK ON ALGORITHM 20 REAL EXPONENTIAL INTEGRAL (S. Peavy, *Comm. ACM*, October 1960)

S. Peavy
National Bureau of Standards, Washington, D. C.

A printing error has been called to our attention by J. A. Beutler of E. I. duPont de Nemours and Co. Lines 15 through 17 of Algorithm 20 should read

z := ((((.00107857 × x − .00976004) × x
+ .05519968) × x − .24991055) × x
+ .99999193) × x − .57721566 − ln (x)

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

CERTIFICATION OF ALGORITHM 3 SOLUTION OF POLYNOMIAL EQUATIONS BY BAIRSTOW HITCHCOCK METHOD (A. A. Grau, *Comm. ACM*, February, 1960)

James S. Vandergraft
Stanford University, Stanford, California

Bairstow was coded for the Burroughs 220 computer using the Burroughs Algol. Conversion from Algol 60 was made by hand on a statement-for-statement basis. The integer declaration had to be extended to include n, k, n, NAT, EX, and the corrections noted in the certification by Henry C. Thacher, Jr., *Communications ACM*, June, 1960, were incorporated.

By selecting the input parameters carefully, all branches of the routine were tested and the program ran smoothly. The following polynomials equations were solved:

$x^6 - 14x^4 + 49x^2 - 36 = 0$, $x = \pm 1.0000000$
$x = \pm 1.9999998$
$x = \pm 3.0000001$

$x^8 - 30x^6 + 273x^4 - 820x^2 + 576 = 0$, $x = \pm 1.0000000$
$x = \pm 2.0000000$
$x = \pm 2.9999999$
$x = \pm 4.0000001$

Several minor errors were found in the certification by Mr. Thacher. The constant term in the first polynomial should be 54 instead of 43, the second pair of roots for that polynomial should be $+ 2.470639 \pm 4.6405330$ i, and the second pair of roots for the second polynomial should be $-1.0 \pm$ i.

ALGORITHM 31 GAMMA FUNCTION

Robert M. Collinge
Burroughs Corporation, Pasadena, California

```
real procedure Gamma (x); real x;
comment For x in the range 2 ≤ x ≤ 3 an approximating polynomial is used. In this range the maximum absolute error ε(x) is | ε(x) | < 0.25 × 10⁻⁷. For x > 3 we write Γ(x) = (x−1)(x−2)...(x−n)Γ(x−n) where 2 ≤ (x−n) ≤ 3, and for x < 2 we write Γ(x) = Γ(x+n) / (x(x+1)...(x+n−1)) where 2 ≤ (x−n) ≤ 3. For x = 0 or a negative integer Γ(x) is set equal to a large value 10⁵⁰.
begin
  real h, y;
  h := 1.0;  y := x;
A1: if y = 0 then h := 10⁵⁰
    else if y = 2.0 then go to A2
    else if y < 2.0 then begin
    h := h/y;  y := y + 1.0;  go to A1 end
    else if y ≥ 3.0 then begin
    y := y − 1.0;  h := h × y;  go to A1 end
    else begin y := y − 2.0;
    h := (((((((.0016063118 × y + .0051589951) × y
      + .0044511400) × y + .0721101567) × y
      + .0821117404) × y + .4117741955) × y
      + .4227874605) × y + .9999999758) × h end;
A2: Gamma := h end Gamma.
```

ALGORITHM 32
MULTINT

R. Don Freeman Jr.
Michigan State University, East Lansing, Michigan

```
real procedure MULTINT (n, Low, Upp, Funev, s, P, u, w);
        value n;
        real procedure Low, Upp, Funev; array s, u,
        w; integer n;
comment MULTINT will perform a single, double, triple,..., T-order integration depending on whether n=1, 2,..., T. The result is:
```

$$\text{MULTINT} = \int_{\text{Low}(1)}^{\text{Upp}(1)} \text{Funev}(1, x_1)\, dx_1 \int_{\text{Low}(2,x_1)}^{\text{Upp}(2,x_1)} \text{Funev}(2,x_1,x_2)\, dx_2 \ldots \int_{\text{Low}(n,x_1,\ldots,x_{n-1})}^{\text{Upp}(n,x_1,\ldots,x_{n-1})} \text{Funev}(x_1,\ldots,x_n)\, dx_n$$

```
The variable of integration is x[j]. j=1 refers to the outermost integral, j=n, the innermost integral. The code divides each interval equally into s[j] subintervals and performs a P-point Gaussian integration on each subinterval with weight functions w[k[j]] and abscissas u[k[j]]. P is the size of the arrays of weight functions and abscissas and must be provided by the main code along with these arrays.

Since the values x[1], x[2],..., x[n], are stored in an array, as are a, b, c, d, r, it is necessary to substitute an integer for the upper bound T of these arrays before the program is executed. This means, for example, if 3 is substituted for T, then the procedure will not do a 4th order integral unless it is retranslated with T ≥ 4.

The values of the lower and upper bounds and functions must of course be specified at the time of use. If each of these constituted a separate procedure, it would require writing and translating 3n different procedures. This is eliminated by grouping them into Low, Upp, and Funev which compute the lower and upper bounds and value of the functions respectively in each of the jth integrals. Since these are each essentially a collection of "subprocedures," the first statement of each should be a switch directing the code to the "subprocedure" which is used in the jth integral. Note that, for example, Low(3,x) is formally a function of x[1], x[2],..., x[T]; this is done merely because it is more convenient to make Low(j,x) formally a function of the whole array x for all j. Actually of course Low(3, x) would be a function of x[1] and x[2] only;
begin   real array a, b, c, d, r, x[1:T];
        integer array k, h[1:T]; real f; integer j, m;
        for j :=1 step 1 until T do
            x[j] := 0.0;
        m := 1;
        r[n+1] := d[n+1] := 1.0;
setup:  for j := m step 1 until n do
        begin
          a[j] := Low(j,x);
          b[j] := Upp(j,x);
          d[j] := (b[j]- a[j])/s[j];
          c[j] := a[j] + 0.5 × d[j];
          x[j] := c[j] + 0.5 × d[j] × u[1];
          r[j] := 0.0;
          h[j] := k[j] := 1; end;
          j := n;
sum:    f := Funev(j,x);
        r[j] := r[j] + r[j+1] × d[j+1] × f × w[k[j]];
        if (k[j] < P) then go to labk;
        if (h[j] < s[j]) then go to labh;
        j := j-1;
        if (j = 0) then go to exit;
        go to sum;
labh:   h[j] := h[j] + 1;
        c[j] := a[j] + (h[j] - 0.5) × d[j];
        k[j] := 1;
        go to initalx;
labk:   k[j] := k[j] + 1;
initalx: x[j] := c[j] + 0.5 × d[j] × u[k[j]];
        if (j=n) then go to sum;
        m := j+1;
        go to setup;
exit:   MULTINT := r[1] × d[1] × 0.5↑ n; end
```

ALGORITHM 33
FACTORIAL

M. F. Lipp
RCA Digital Computation and Simulation Group, Moorestown, New Jersey

```
real procedure Factorial (n) ;
        value n ; integer n ;
comment This procedure makes use of the implicitly defined recursive property of Algol to compute n!;
begin   Factorial := if n = 0 then 1. else n× Factorial (n-1)
        end
```

ALGORITHM 34
GAMMA FUNCTION

M. F. Lipp
RCA Digital Computation and Simulation Group, Moorestown, New Jersey

```
real procedure Gamma (x) ; real x ;
comment This procedure generalizes the recursive factorial routine, finding Γ(1+x) for reasonable values of x. Accuracy vanishes for large x(|x| > 10) and for negative x with small fractional parts. For x being a negative integer the impossible value zero is given;
begin   test: if x < 0 then go to minus else if x < 1 then
              begin integer i ; real y ; array a [1:8] ;
                a [1] := -.57719165 ;
                a [2] := .98820589 ; a [3] := -.89705694 ;
                a [4] := .91820686 ;
                a [5] := -.75670408 ; a [6] := .48219939 ;
                a [7] := -.19352782 ;
                a [8] := .03586834 ; y := a [1];
              for i := 2 step 1 until 8 do y := y × x + a [i] ;
              Gamma := y end hastings
              else Gamma := x × Gamma (x-1) ; go to endgam;
              minus: if x = -1 then Gamma := 0 else
              Gamma := Gamma (x+1) / x ;
              endgam : end gam
```