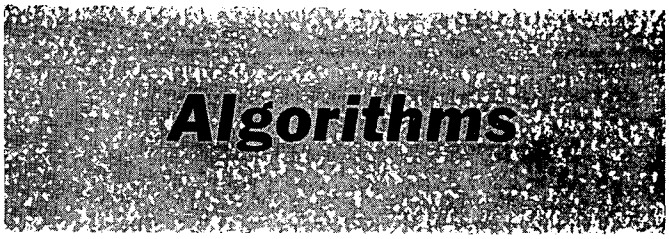


G. E. FORSYTHE, Editor

ALGORITHM 225
GAMMA FUNCTION WITH CONTROLLED ACCURACY

S. J. CYVIN AND B. N. CYVIN (Recd. 25 Oct. 1963)
Technical University of Norway, Trondheim, Norway

```
real procedure GAMMA (m, x); value m, x; integer m;
  real x;
comment Γ(x) is calculated with at least m significant figures
  (disregarding the machine's roundoff). The range of x is reduced
  by recursion to 5 ≦ x ≦ 6, for which Γ(x) is found (with m−2
  significant decimals) according to

        Γ(x) = ∫_0^T t^{x−1}e^{−t} dt + ∫_T^∞ t^{x−1}e^{−t} dt.

  Simpson's formula is applied to the former integral, which is
  divided into 2n parts. Here n, as well as T, are chosen auto-
  matically to give a result with the required accuracy. For x
  near zero or a negative integer, Γ(x) is put equal to a large value,
  10^60. The procedure is slower than other algorithms for Γ(x)
  [see Nos. 31, 34, 54, 80], but has the advantage of controlled
  accuracy;
begin integer i,n,f,T; real y,h,S;
    h := 1; y := x;
A: if abs(y) < 10−60 then begin GAMMA := 1060; go to
     E end else
   if y > 6 then begin y := y−1; h := h×y; go to A end else
   if y < 5 then begin h := h/y; y := y+1; go to A end else
   begin real a;
     T := 20;
U:   if (T↑5 + 4×T↑4 + 16×T↑3 + 48×T↑2 + 96×T +
       96)×exp(−T)>.25×10↑(2−m) then begin T := T+5;
       go to U end;
     n := 1 + entier(sqrt(sqrt(T↑5×10↑(m−2)/30)));
     S := 0; f := 4;
     for i := 1 step 1 until 2×n do
     begin
       a := .5×i×T/n; S := S + f×a↑(y−1)×exp(−a);
       f := if i = 2×n−1 then 1 else if f = 4 then 2 else 4
     end
   end;
   GAMMA := (S×T/(6×n) + (.5×T↑5 + 3×T↑4 + 12×T↑3
     + 36×T↑2 + 72×T + 72)×exp(−T))×h;
E:
end of GAMMA
```

ALGORITHM 226
NORMAL DISTRIBUTION FUNCTION

S. J. CYVIN (Recd. 15 Oct. 1963)
Technical University of Norway, Trondheim, Norway

```
real procedure Fi(m,x); value m,x; integer m; real x;
comment Φ(x) = (1/√2π)∫_{−∞}^x exp(−½u²) du is found by com-
  puting ∫_0^x exp(−½u²) du with aid of Simpson's formula. The
  latter integral is divided into 2n parts, where n automatically
  is adjusted to give a result with at least m significant decimals
  (disregarding the machine's roundoff). The error function is
  obtainable as erf(x) = 2Φ(x/√2) − 1. The practical use of the
  present method is not restricted to small or large ranges of x.
  Probably the method has some advantages compared to Algo-
  rithms 123, 180, and 209;
begin integer i,n,f; real b,S;
  b := abs(x);
  n := 1 + entier(sqrt(sqrt(b↑5×10↑m/
    (480×sqrt(2×3.14159265)))));
  if n < 4 then n := 4; S := 1; f := 4;
  for i := 1 step 1 until 2×n do
  begin
    S := S + f × exp(−(i×b/n)↑2/8);
    f := if i = 2×n−1 then 1 else if f=4 then 2 else 4
  end;
  Fi := .5 + sign(x)×S×b/(6×n×sqrt(2×3.14159265))
end Fi
```

ALGORITHM 227
CHEBYSHEV POLYNOMIAL COEFFICIENTS

S. J. CYVIN (Recd. 15 Oct. 1963)
Technical University of Norway, Trondheim, Norway

```
procedure Tcheb(n,A); value n; integer n; integer array A;
comment This procedure finds (by recursion) the coefficients
  of T_n(x), rather than the value of the polynomial, which is the
  subject of Algorithms 10 and 36. The (n+2)÷2 nonvanishing
  coefficients are stored in one-dimensional integer array A in
  the following way:

    T_{2p}(x) = Σ_{i=0}^{p} A[i+1] x^{2i}   (n even),

                T_{2p+1}(x) = Σ_{i=0}^{p} A[i+1]x^{2i+1}   (n odd);

begin integer i,j; integer array B[1:(n+2)÷2]; Boolean
  EVEN;
  A[1] := B[1] := 1; EVEN := n÷2×2 = n; if n > 1 then
  for i := 2 step 1 until (n+2)÷2 do
  for j := i step −1 until 1 do
  begin
    A[j] := if j=i then 2×B[j−1] else if j=1 then −A[1]
    else 2 ×B[j−1] − A[j];
    B[j] := if j=i then 2 × A[i] else 2 × A[j] − B[j]
  end i loop;
  for i := 1 step 1 until (n+2)÷2 do
    A[i] := if EVEN then A[i] else B[i]
end Tcheb
```

ALGORITHM 228
Q-BESSEL FUNCTIONS $\bar{I}_n(t)$

J. M. S. SIMÕES PEREIRA (Recd. 21 Sept. 63 and 6 Jan. 64)
Gulbenkian Scientific Computing Ctr, Lisboa, Portugal

```
procedure qBesselbar (t,q,n,j,s); integer n, j; real t,q,s;
comment This procedure computes values of any q-Bessel
  function Ī_n(t) for n integer (positive, negative or zero) by the
  use of the expansion Ī_n(t) = Σ_{k=0}^∞ (t^{n+2k}/((q)_k(q)_{n+k}))) where
  (q)_n = (1−q)(1−q²)···(1−q^n), (q)_0 = 1 and (1/(q)_{−n})=0 (n=1,
  2, ···). This series is convergent for t ∈ (−∞, +∞) if | q | > 1
  and for | t | < 1 if | q | < 1. j+1 denotes the number of terms
  (at least 2) retained in the summation and s stands for the sum
  of these first terms. See L. Carlitz, The product of q-Bessel
  functions, Portugaliae Mathematica, vol. 21;
```

```
begin integer k,m,p; real c,u; m := abs(n); c := 1;
    if n = 0 then go to A;
    for p := 1 step 1 until m do c := c×(1−q↑p);
    if n < 0 then go to B;
A: u := (t↑n)/c; s := u;
    for k := 1 step 1 until j do
    begin u := u×(t↑2)/((1−q↑k)×(1−q↑(n+k))); s :=
      s + u end;
    go to C;
B: u := t ↑ (n+2×m)/c; s := u;
    for k := m + 1 step 1 until j do
    begin u := u×(t↑2)/((1−q↑k)×(1−q↑(n+k))); s :=
      s + u end;
C: end
```

ALGORITHM 229
ELEMENTARY FUNCTIONS BY CONTINUED FRACTIONS

James C. Morelock (Recd. 1 Oct. 63 and in final form 24 Jan. 64)
Computation Lab., Marshall Space Flight Ctr, NASA, Huntsville, Ala.

```
procedure CONFRAC (x, n, parm, answer);
  integer parm, n; real x, answer;
comment This procedure utilizes a continued fraction which is
  equivalent to the diagonal of the Padé table for exp z, with
  error in the computed convergent less than x^2n/(2 × 6^2 ×
  (10)^2 × ··· × (4n − 2)^2(4n + 2)). This fraction was developed by
  J. C. Morelock, Note on Padé Table Approximation, Internal
  Note MIN-COMP-62-9, Marshall Space Flight Center, Hunts-
  ville, Alabama, 1962. For source reference see Nathaniel Macon,
  On the computation of exponential and hyperbolic functions
  using continued fractions, J. ACM, 2(1955), 262–266. The argu-
  ment, x, is assumed to be less than π/4. For such x any desired
  level of accuracy is quickly computed for each function specified
  as follows:

  parm := 1, answer := sin x     parm := 5, answer := sinh x
  parm := 2, answer := cos x     parm := 6, answer := cosh x
  parm := 3, answer := tan x     parm := 7, answer := tanh x
  parm := 4, answer := exp x
```

The body of this procedure has been tested using extended Algol for the B-5000 Computer. It gave the following results:

```
x = 0.50  n = 1  parm = 1  answer = 0.47938 801530
x = 0.50  n = 2  parm = 1  answer = 0.47942 547125
x = 0.50  n = 3  parm = 1  answer = 0.47942 553854
x = 0.50  n = 4  parm = 1  answer = 0.47942 553860
x = 0.50  n = 1  parm = 2  answer = 0.87760 305992
x = 0.50  n = 2  parm = 2  answer = 0.87758 259869
x = 0.50  n = 3  parm = 2  answer = 0.87758 256193
x = 0.50  n = 4  parm = 2  answer = 0.87758 256189
x = 0.50  n = 1  parm = 3  answer = 0.54624 697337
x = 0.50  n = 2  parm = 3  answer = 0.54630 239019
x = 0.50  n = 3  parm = 3  answer = 0.54630 248974
x = 0.50  n = 4  parm = 3  answer = 0.54630 248985
x = 0.50  n = 1  parm = 4  answer = 1.64864 864865
x = 0.50  n = 2  parm = 4  answer = 1.64872 139973
x = 0.50  n = 3  parm = 4  answer = 1.64872 127057
x = 0.50  n = 4  parm = 4  answer = 1.64872 127070
x = 0.50  n = 1  parm = 5  answer = 0.52104 563580
x = 0.50  n = 2  parm = 5  answer = 0.52109 539374
x = 0.50  n = 3  parm = 5  answer = 0.52109 530541
x = 0.50  n = 4  parm = 5  answer = 0.52109 530549
x = 0.50  n = 1  parm = 6  answer = 1.12760 301285
x = 0.50  n = 2  parm = 6  answer = 1.12762 600598
x = 0.50  n = 3  parm = 6  answer = 1.12762 596516
x = 0.50  n = 4  parm = 6  answer = 1.12762 596521
x = 0.50  n = 1  parm = 7  answer = 0.46208 251473
x = 0.50  n = 2  parm = 7  answer = 0.46211 721881
x = 0.50  n = 3  parm = 7  answer = 0.46211 715720
x = 0.50  n = 4  parm = 7  answer = 0.46211 715726
```

The value of n selects the continued fraction convergent;

```
begin integer i, ndigt;
  real r, f;
  r := if parm ≦ 3 then − x ↑ 2 else x ↑ 2;
  f := 4 × n + 2;
  for i := n step −1 until 1 do f := 4 × i − 2 + r/f;
  ndigt := if parm ≦ 3 then parm + 1 else parm −3;
  answer := if ndigt = 1 then (f+x)/(f−x)
    else if ndigt = 2 then 2 × x × f/((f ↑ 2) − r)
    else if ndigt = 3 then ((f↑2)+r)/((f↑2)−r)
    else if ndigt = 4 then 2 × x × f/((f↑2)+r)
    else x;
end
```

REMARKS ON ALGORITHM 91
CHEBYSHEV CURVE FIT [A. Newhouse, *Comm. ACM 5* (May 1962), 281; *6* (April 1963), 167]

Peter Naur (Recd. 27 Sept. 1963)
Regnecentralen, Copenhagen, Denmark

In addition to the corrections noted by R. P. Hale [op. cit., April 1963] the following are necessary:

1. The arrays X, Y, and A cannot be declared to be local within the procedure body.
2. The identifier A must be included as a formal parameter.
3. It should be noted that the $X[i]$ must form a monotonic sequence.
4. **comment** cannot follow the colon following a label. This occurs in four places.
5. The **end** following **go to** *FIT* must be removed.

In addition, a large number of details can be made more concise and unnecessary operations can be eliminated. Also, it seems desirable to produce the maximum deviation as a result.

CERTIFICATION OF ALGORITHM 146
MULTIPLE INTEGRATION [W. M. McKeeman, *Comm. ACM 5* (Dec. 1962), 604]

Niklaus Wirth (Recd. 6 Jan. 1964)
Computer Science Div., Stanford U., Stanford, Calif.

Algorithm 146 was translated into a generalized Algol [cf. N. Wirth, A generalization of ALGOL, *Comm. ACM 6* (Sept. 1963), 547–554] and successfully run on the Stanford IBM 7090 computer. Algorithm 60, Romberg Integration [*Comm. ACM 4* (June 1961), 255; *5* (Mar. 1962), 168; *5* (May 1962), 281] was used for the real procedure *Integral*.

The main disadvantage of Algorithm 146 is that the bounds of the domain of integration must be constant, i.e. the domain must always have the form of a rectangular hyperbox.

REMARK ON ALGORITHM 175
SHUTTLE SORT [C. J. Shaw and T. N. Trimble, *Comm. ACM 6* (June 1963), 312; G. R. Schubert, *Comm. ACM 6* (Oct. 1963), 619; O. C. Juelich, *Comm. ACM 6* (Dec. 1963), 739]

Otto C. Juelich (Recd. 18 Dec. 1963)
North American Aviation, 4300 E. Fifth Ave., Columbus, Ohio

The appearance of Schubert's certification has caused me to restudy the algorithm. What I supposed were errors amount to a rearrangement of the order in which the comparisons are carried out. The efficiency of the algorithm is not much affected by the rearrangement, since the number of executions of the statements labeled *Exchange* remains the same.

CERTIFICATION OF ALGORITHM 215
SHANKS [H. C. Thacher, Jr., *Comm. ACM 6* (Nov. 1963), 662]

LARRY SCHUMAKER (Recd. 16 Dec. 63)
Computation Ctr., Stanford U., Stanford, Calif.

Algorithm 215 was coded in Extended ALGOL for the Burroughs B-5000 and was tested on a large number of sequences. One apparent typographical error was noted. The statement *lim* := *j* − *nmin* should have read *limk* := *j* − *nmin*. The following tables were reproduced exactly: (a) tables on p. 5 and p. 33 of [1]; (b) Table I on p. 95 of [2]; (c) Tables III and IV on p. 28 of [3].

REFERENCES:

1. SHANKS, D. Non-linear transformations of divergent and slowly convergent sequences. *J. Math. Phys. 34* (1955), 1–42.
2. WYNN, P. On a device for computing the $e_m(S_n)$ transformation. *MTAC 10* (1956), 91–96.
3. WYNN, P. On repeated application of the ϵ-algorithm. *Chiffres 4* (1961), 19–22.

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 **bold-face** 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):

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]);

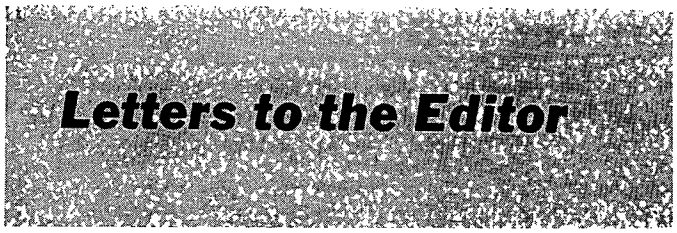
It is intended that each published algorithm be a well-organized, clearly commented, syntactically correct, and a substantial contribution to the ALGOL literature. All contributions will be refereed both by human beings and by an ALGOL compiler. Authors should give great attention to the correctness 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, e.g., the IFIP subset [*Comm. ACM 7* (May 1964), 273–283].

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.

The reproduction of algorithms appearing in this department is explicitly permitted without any charge. When reproduction is for publication purposes, reference must be made to the algorithm author and to the *Communications* issue bearing the algorithm. G. E. F.



GEORGE E. FORSYTHE, Editor

More on Merging

Dear Editor:

With regard to the "Three Letters on Merging," it may be of interest to your readers to know that analytic derivations of the results supplied by Donald Knuth exist.

The reduction factor for a cascade merge with $n+1$ tapes is the principal root of

$$\sum_{m=0}^{n} \binom{[(n+m)/2]}{m} (-1)^{[(m+1)/2]} x^{n-m} = 0.$$

An approximation for x is $x \cong 1+[2(n-1)/\pi]$.

In the case of a polyphase merge, if y is the principal root of $y^n - \sum_{m=0}^{n-1} y^m = 0$, then the reduction factor is $y^{\{(n+1)y-2n\}/(y-1)}$. An approximation for y is

$$y \cong \frac{1+4(n-1)(2^n-1)}{1+(n-1)(2^{n+1}-1)}.$$

Numerically, these results agree quite well with Mr. Knuth's results. The proofs of these results are being prepared as an article.

DAVID E. FERGUSON
Programmatics, Inc.
Los Angeles, California

On Semantics

Dear Editor:

I read with interest the February 1964 issue of the *Communications* and I have some comments. I am very interested in the area of formal definition of languages. So I found the General Discussion disappointing, with the semantic question as the main topic. The participants in this discussion did not seem to realize that semantics in formal languages have a dual aspect. This may explain the misunderstandings.

Two different kinds of semantics are involved in the syntax method: *semantic definition* and *semantic interpretation.*

Semantic definition means the possibility of additional semantic definition beyond the syntactic definition. For example, a label in a control statement has to appear again unchanged as a statement number, or subscripted variables have to be declared by their right name in array declarations, etc.

Semantic interpretation refers to the interpretation of the syntactic units, which is done in a compiler system by the substitution of the appropriate machine (or assembler) instructions for the syntactic units.

I hope that this helps to clarify the problem of the semantic question.

WALTER H. BURKHARDT
10 Redondo Drive
Poughkeepsie, New York

Letters are continued on page 314