Algorithms

J. G. HERRIOT, Editor

ALGORITHM 272
PROCEDURE FOR THE NORMAL DISTRIBUTION FUNCTIONS* [S15]

M. D. MacLaren

(Recd. 28 July 1964, 17 Nov. 1964 and 26 July 1965)

Argonne National Laboratory, Argonne, Ill., and Boeing Scientific Research Laboratories, Seattle, Wash.

* Work performed in part under the auspices of the US Atomic Energy Commission.

real procedure *phi*(*a*, *k*); **value** *a*, *k*; **real** *a*; **integer** *k*;
comment Before use, this procedure must be called once with $k = 3$ to initialize **own** variables. Thereafter for $k = 1$ the procedure gives

$$\Phi(a) = \frac{1}{(2\pi)^{\frac{1}{2}}} \int_{-}^{a} \exp(-t^2/2)\, dt,$$

and for $k = 2$ it gives

$$\Phi^*(a) = 2(\Phi(|a|) - .5) = \left(\frac{2}{\pi}\right)^{\frac{1}{2}} \int_0^{|a|} \exp(-t^2/2)\, dt;$$

begin own integer *N*;
own real *B*, *EPS*, *EPS2*, *EPS3*, *ONE*, *DELTA*, *DELTA2*, *PI2*;
comment $\Phi^*(a)$ is computed by Taylor's series expansion in the interval $0 \leq a \leq B$, and by asymptotic series in the interval $B < a$. The Taylor's series expanson is made about one of the points $0, B/N, 2B/N, \cdots, B$, and the coefficients in the series are computed using the recursion formula for Hermite polynomials. The number of terms to take in the series is determined by an error estimate based on a majorizing series. This procedure, which is essentially the familiar one of interpolating in a stored table of values, gives a fast program and can be used effectively for many functions. In this case another significant increase in speed could be obtained by also storing a table of values of the first derivative of Φ^*. The **own** variables *B*, *EPS* and *N* might be called program parameters. By suitably choosing their values the programmer may make the procedure as accurate as desired and may increase the speed of the procedure at the cost of extra storage space. This is the advantage of this procedure over others previously published in this journal (see [1-4]).

The values of these program parameters are determined when the procedure is coded, not when it is called. They are set by means of an initializing call with $k = 3$. The other **own** variables are computed from *B*, *EPS* and *N* when the initializing call is made. If FORTRAN IV were used, all the **own** variables could be set by use of a DATA statement. An alternative to either method is to replace all occurrences of the parameters by the appropriate constants.

The choice of the parameter *N* depends mainly on speed versus storage considerations. The larger *N* is, the faster the procedure will be and the more storage will be needed. Note, however, that *N* must be chosen large enough so that $B^2(1/(2N) + 1/(4N^2)) \leq 1$, for otherwise the method of estimating the error in the Taylor's series may fail. The choice of *B* may also affect the speed, because for smaller values of *a* the asymptotic series for $\Phi^*(a)$ will take longer than the Taylor's series. The choice of *B* depends, however, mainly on the error desired. Neglecting roundoff, the maximum error in the computed value of $\Phi^*(a)$ will be *EPS* if $a \leq B$ or $\max(EPS, \delta(a)/2)$ if $B < a$, where $\delta(a)$ is the absolute value of the smallest term in the asymptotic series for $\Phi^*(a)$. Some values of $\delta(a)$ are: $\delta(4) = 3.0 \times 10^{-8}$, $\delta(5) = 3.0 \times 10^{-12}$, $\delta(5.5) = 1.4 \times 10^{-14}$, and $\delta(6) = 4.4 \times 10^{-17}$. If *N* is large enough, roundoff will be no problem. (The referee has pointed out that the computation for $B < a$ could be made by continued fractions, as in Algorithm 180. The advantage of this would be that the continued fraction expansion converges for all $a > 0$, but roundoff errors may be significant for smaller values of *a*.)

With the program parameters having the values given below, the procedure was compiled as a FORTRAN II subroutine on the IBM 1620, using eight-digit floating point arithmetic, and tested for many values of *a*. The error never exceeded 2×10^{-8}. The program was also compiled with $B = 6.0$, $EPS = 2 \times 10^{-15}$ and $N = 60$, using 15 digit arithmetic. Spot checks turned up no errors greater than 2×10^{-15};

```
own real array C[0:16];
comment The array C must give the value of Φ*(a) at the
  point of expansion, i.e., C[m] must equal Φ*(mB/N). Tables of
  Φ*(a) to fifteen decimal places are published by the National
  Bureau of Standards [5]. The upper bound for the array must
  equal the value of the program parameter N;
real f, f1, f2, x, y, z, t, t2, xt;
integer m;
real procedure max(x, y); value x, y; real x, y;
begin max := if x ≤ y then y else x;
end max;
if k = 3 then
begin comment initialize own variables;
  EPS := .00000002; B := 4.0; N := 16; C[0] := 0.0;
    C[1] := .19741265;
  C[2] := .38292492; C[3] := .5467530;
    C[4] := .68268949;
  C[5] := .78870045; C[6] := .86638560;
    C[7] := .91988169;
  C[8] := .95449974; C[9] := .97555105;
    C[10] := .98758067;
  C[11] := .99404047; C[12] := .99730020;
    C[13] := .99884595;
  C[14] := .99953474; C[15] := .99982317;
    C[16] := .99993666;
  ONE := .99999999;
  comment ONE is the largest number less than 1 which may
    be represented in the machine. This prevents loss of ac-
    curacy in some implementations of floating point sub-
    traction;
  PI2 := .797884560802865;
  comment PI2 = (2/π)^1/2;
  DELTA := B/N;
  DELTA2 := .5 × DELTA;
  EPS3 := 2.0 × EPS;
  t2 := max(B×DELTA, sqrt(2.0)×DELTA2);
  t := DELTA2 × (B+DELTA2);
  x := (t+sqrt(t)) × exp(.5×t);
  y := t2 × (1.0+t2) × exp(.5×t2↑2);
  if t2 ≤ 1 ∧ y ≤ x then EPS2 := EPS/y else EPS2 := EPS/x;
  phi := 0
end initialization
else
begin comment compute Φ(a);
  y := abs(a);
  if y > B then
  begin comment computation by asymptotic series;
    x := y ↑ 2; f := PI2 × exp(-.5×x)/y;
    x := 1.0/x; z := f; f1 := -f × x;
```

```
    for m := 3, m + 2 while abs(f1) < abs(f) do
    begin z := z + f1; f := f1; f1 := -f1 × m × x;
        if abs(f) ≤ EPS3 then go to L1
    end;
L1:     z := ONE - z + .5 × f
  end asymptotic computation
  else
  begin comment Taylor's series computation;
    m := entier (y/DELTA);
    x := m × DELTA; t := y - x;
    if DELTA2 < t then
    begin m := m + 1; x := x + DELTA; t := y - x end;
    xt := x × t; t2 := t ↑ 2;
    f1 := t × PI2 × exp(-.5×x ↑ 2);
    f2 := -.5 × xt × f1;
    z := C[m] + f1 + f2;
    for m := 3, m + 1 while (m-1) × EPS2 < max(abs(f1),
      abs(f2)) do
    begin
      f := (-xt×f2-t2×(m-2)×f1)/(m-1)/m;
      z := z + f; f1 := f2; f2 := f;
    end
  end Taylor's series computation;
  if k = 1 then
  begin
    z := if 0 ≤ a then .5 + .5 × z else .5 - .5 × z
    end;
  phi := z
 end computation
end phi
```


REFERENCES:

1. CRAWFORD, M., AND TECHO, R. Algorithm 123, Real error function, *ERF(x)*. *Comm. ACM 5* (Sept. 1962), 482.
2. THACHER, H. C., JR. Algorithm 180, Error function—large *X*. *Comm. ACM 6* (June 1963), 314.
3. IBBETSON, D. Algorithm 209, Gauss. *Comm. ACM 6* (Oct. 1963), 616.
4. CYVIN, S. J. Algorithm 226, Normal distribution function. *Comm. ACM 7* (May 1964), 295.
5. NATIONAL BUREAU OF STANDARDS. *Tables of Normal Probability Functions*. Applied Math. Series, No. 23, US Government Printing Off., Washington, D.C., 1953.

CERTIFICATION OF ALGORITHM 124 [S17] HANKEL FUNCTION [Luis J. Schaeffer, *Comm. ACM 5* (Sept. 1962), 483]

GEORGE A. REILLY (Recd. 5 Oct. 1964 and 4 Nov. 1964) Westinghouse Research Laboratories, Pittsburgh, Pa.

This procedure, after modification, was run on the B-5000 using B-5000 ALGOL. Values obtained checked with US National Bureau of Standards *Handbook of Mathematical Functions*, Applied Mathematics Series 55, US Government Printing Office, Washington, D.C. 1964.

For $N = 0$, 1 and 2, accuracy was to 10 decimals for $X < 8.0$. It deteriorated to 6 decimals for $8 < X < 17.5$. For $3 \leq N \leq 9$ accuracy was to the 5 decimals of the tables.

Some changes proved necessary to make the algorithm run. Since the algorithm is short and the changes are involved, the algorithm is restated here. Note that a test for a zero argument X is included in the body of the procedure since $H[2]$ ought to be minus infinity when $X = 0$.

```
procedure HANKEL (N, X, H); value N, X; integer N;
  real X; array H;
begin real K, P, R, A, S, T, D, L; integer Q;
  if X = 0 then
  begin comment In this case H[2] is minus infinity. M denotes
      the largest number which can be represented in the machine.
      The numerical value of M is to be written into the
        procedure:
    H[2] := -M;
    H[1] := if N = 0 then 1 else 0;
    go to exit
  end;
  A := R := 1; H[1] := H[2] := S := 0;
  if N = 0 then begin R := 1; S := D := 0 end
  else
  begin for Q := 1 step 1 until N do
    begin R := R × Q; S := S + 1/Q end; D := R/N
  end;
  R := 1/R; K := X × X/4; P := K ↑ N; T := ln (K) -
    1.1544313298631;
```

* THE NUMBERS 263 AND 264 WERE EACH INADVERTENTLY ASSIGNED TO TWO ALGORITHMS. WHEN GIVING REFERENCES TO THESE ALGORITHMS, PLEASE BE CAREFUL TO DO SO UNAMBIGUOUSLY.

Key—1st column: A1, B1, B3, etc. is the key to the underlined Modified Share Classification heading each group of algorithms; 2d column: number of the algorithm in *CACM*; 3d column: title of algorithm; 4th column: month, year and page (in parens) in *CACM*, or reference elsewhere. This index supplements the previously published indexes: Index by Subject to Algorithms, 1960-1963 [*CACM* 7 (Mar. 1964), 146-149], and 1964 [*CACM* 7 (Dec. 1964), 703].

```
comment The last constant is 2 × gamma, Euler's constant;
for Q := 0, Q + 1 while Q ≤ N ∨ L ≠ H [2] do
begin L := H[2]; H[1] := H[1] + A × R;
  H[2] := H[2] + A × (R×(T−S) − (if q<N then D/P else 0));
  A := A × K/(Q+1); R := −R/(Q+N+1);
  S := S + 1/(Q+1) + 1/(Q+N+1);
  if Q + 1 < N then D := D/(N−Q−1);
end;
P := (X/2) ↑ N; H[1] := H[1] × P; H[2] := 0.318309886184
  × H[2] × P;
comment The multiplicative constant is 1/Pi;
exit:
end HANKEL
```