and

$$\int_0^1 dx \int_0^{\sqrt{1-x^2}} dy(x^2+y^2)^{-1/2},$$

have been successfully integrated with this procedure to 1% accuracy.

B. Functions having an infinite number of zeros in the interval of integration. Such integrals as

$$\int_0^1 dy\ \sqrt{y} \sin(1.5 \ln y),$$

$$\int_0^1 dy\ y^{-1/2} \sin(0.5 \ln y),$$

and

$$\int_1^2 dx \int_0^1 dy\ xy^{(x-1)} \sin(x \ln y),$$

have been successfully integrated with this procedure to 1% accuracy.

C. Functions having high-frequency oscillations or a large number of discontinuities. The function

$$f(x) = \begin{cases} 2 \text{ if the least significant bit of } x \text{ is } 1 \\ 0 \text{ otherwise} \end{cases}$$

is almost as discontinuous as can be represented in a binary number computer. One hundred attempts at integrating this function on the interval 0 to 1 gave an average of the absolute value of the error ≈ 0.13.

The main limitation in integrating anomalous functions of the above type is in the hardware or software of the particular machine being used. The procedure will fail when the interval is subdivided to a point where it is smaller than the smallest in magnitude nonzero number representable in the machine.

A histogram is given below of the errors in the evaluation of the integrals

$$\int_0^1 dy\ xy^{(x-1)} \sin(x \ln y)$$

and

$$\int_0^1 dy\ xy^{(x-1)} \cos(x \ln y)$$

for $x = 1.04(0.04)2.00$, with error tolerances 10^{-3} and 10^{-4}.

Number of occurrences	0	0	1	1	53	41	4	0	0	0
$\log_{10}(\epsilon/\epsilon_0)$	−5	−4	−3	−2	−1	0	1	2	3	4	5

Here ϵ_0 is the error requested, ϵ is the error obtained.

The formal parameter *fx* is an arithmetic expression dependent on *x*. In translating to another language it may be desirable to make this parameter a procedure identifier with appropriate modifications in the body of the program;

```
if a = b then Integral := 0
else
begin real fl, fr, c;
  real procedure Int(a, x, b, fx, fc2, error);
    value a, b, fc2, error;
    real a, x, b, fx, fc2, error;
  begin real dx, dxc, fc1, fc3;
    error := error × 0.577;
    comment The factor 0.577 is an approximation to 1/√3.
      The assumption here is that error contributed by the indi-
      vidual panels is random and not additive, thus the error
      from three panels is assumed to be √3 (not 3) times the
      error of one panel;
    dxc := (random number+0.5) × (b−a)/3;
    dx := (b−a−dxc)/2;
    x := a + dx/2; fc1 := fx;
    x := b − dx/2; fc3 := fx;
    Int :=
    if abs(dx×(fc1−2×fc2+fc3)) ≦ error then
      dx × (fc1+fc3) + dxc × fc2
    else
      Int (a, x, a+dx, fx, fc1, error)
      +Int (a+dx, x, b−dx, fx, fc2, error)
      +Int (b−dx, x, b, fx, fc3, error)
  end;
  c := a + (random number+0.5) × (b−a)/2;
  x := (a+c)/2; fl := fx;
  x := (c+b)/2; fr := fx;
  error := abs(error) × 14.6;
  comment The factor 14.6 can be thought of as an empirical
    constant. There is some theoretical justification for calculat-
    ing an optimum value for this factor, but in practice it was
    determined empirically;
  Integral :=
    Int(a, x, c, fx, fl, error)
    +Int(c, x, b, fx, fr, error)
end
```

ALGORITHM 304
NORMAL CURVE INTEGRAL [S15]

I. D. Hill and S. A. Joyce (Recd. 21 Nov. 1966)
Medical Research Council, Statistical Research Unit, 115 Gower Street, London W.C.1., England

```
real procedure normal (x, upper);
  value x, upper; real x; Boolean upper;
comment calculates the tail area of the standardized normal
  curve, i.e.,
```

$$\frac{1}{\sqrt{2\pi}} \int e^{-1/2t^2}\, dt\,.$$

If *upper* is **true** the limits of integration are *x* and ∞.
If *upper* is **false** the limits are −∞ and *x*.
If *x* lies in the central area of the curve the method used is the convergent series

$$e^{(1/2)x^2} \int_0^x e^{-(1/2)t^2}\, dt = x + \frac{x^3}{} + \frac{x^5}{3 \times 5} + \frac{x^7}{3 \times 5 \times 7} + \cdots .$$

(See [1, 26.2.11].)
If *x* lies in one of the tails the method used is the continued fraction

$$e^{(1/2)x^2} \int_x^\infty e^{-(1/2)t^2}\, dt = \frac{1}{x+}\,\frac{1}{x+}\,\frac{2}{x+}\,\frac{3}{x+}\,\frac{4}{x+}\cdots .$$

(See [1, 26.2.14].)
The changeover point between the two methods is at $abs(x) = 3.5$ if the required area is greater than 0.5. This value is chosen

on grounds of speed. If, however, the required area is less than 0.5, a changeover as far out as 3.5 will lead to the loss of three significant decimal figures due to cancellation error upon making a subtraction. In this case speed is sacrificed to accuracy and the changeover point is at $abs(x) = 2.32$, chosen as the point at which the area is 0.01. The value 2.32 may be changed to 1.28 (the point at which the area is 0.1) if the full accuracy of the machine is desired over the range $1.28 < abs(x) \leqslant 2.32$, but this leads to a considerable loss of speed and the accuracy lost by using 2.32 is only one decimal place.

Except for this subtraction error, the procedure works virtually to the accuracy of the machine (provided that the constant $1/sqrt(2\pi)$ is given to this accuracy) for $x \leqslant 7$ but to 1 decimal place less than the accuracy of the machine for $x > 7$.

REFERENCE: [1]. ABRAMOVITZ, M. AND STEGUN, I. A. *Handbook of Mathematical Functions*, National Bureau of Standards, Appl. Math. Ser. 55, US Government Printing Office, Washington, D.C., 1964;

```
if x = 0 then normal := 0.5 else
begin
  real n, x2, y;
  upper := upper ≡ x > 0;
  x := abs(x);  x2 := x × x;
  y := 0.3989422804014 × exp (−0.5×x2);
  comment 0.3989422804014 = 1/sqrt(2×π);
  n := y/x;
  if ¬ upper ∧ 1.0 − n = 1.0 then normal := 1.0 else
  if upper ∧ n = 0 then normal := 0 else
  begin
    real s, t;
    if x > (if upper then 2.32 else 3.5) then
    begin
      real p1, p2, q1, q2, m;
      q1 := x;  p2 := y × x;
      n := 1.0;  p1 := y;
      q2 := x2 + 1.0;
      if upper then
      begin
        s := m := p1/q1;
        t := p2/q2
      end else
      begin
        s := m := 1.0 − p1/q1;
        t := 1.0 − p2/q2
      end;
      for n := n + 1.0 while m ≠ t ∧ s ≠ t do
      begin
        s := x × p2 + n × p1;
        p1 := p2;  p2 := s;
        s := x × q2 + n × q1;
        q1 := q2;  q2 := s;
        s := m;  m := t;
        t := if upper then p2/q2 else 1.0 − p2/q2
      end;
      normal := t
    end else
    begin
      s := x := y × x;  n := 1.0;  t := 0;
      for n := n + 2.0 while s ≠ t do
      begin
        t := s;  x := x × x2/n;
        s := s + x
      end;
      normal := if upper then 0.5 − s else 0.5 + s
    end
  end
end normal
```

REMARK ON ALGORITHM 179 [S 14] INCOMPLETE BETA RATIO [Oliver G. Ludwig, *Comm. ACM 6* (June 1963), 314]

M. C. PIKE AND I. D. HILL (Recd. 8 Oct. 1965 and 12 Jan. 1966)

Medical Research Council's Statistical Research Unit, University College Hospital Medical School, London, England

Algorithm 179 has the following two typographical errors:

(1) the line

```
if x ⩽ 0.5 then alter := false else
```

should read

```
if x ⩽ 0.5 then alter := false else
```

(2) the line

```
end:end incompletebeta
```

should read

```
End:end incompletebeta
```

With these changes Algorithm 179 ran successfully on the ICT Atlas computer using Algorithm 221 [Walter Gautschi, *Comm. ACM 7* (Mar. 1964), 143], to evaluate the factorials required. A minor improvement might be to call *epsilon* by value.

As the algorithm stands, the permitted range of p and q is dictated by overflow problems associated with finding the values of factorials. For most machines this will mean that $p+q$ will have to be less than about 70. In the statistical applications of this algorithm which we describe below this restriction is very serious. However, these factorials appear essentially only in the form of ratios, and by making use of this fact the permitted range of p and q can be enormously extended. This is most simply accomplished by using the real procedure *loggamma* [Algorithm 291, M. C. Pike and I. D. Hill, *Comm. ACM 9* (Sept. 1966), 684] modifying Algorithm 179 as follows: replace the instructions

```
temp := temp1 := factorial(qrecur−1);
```

to

```
temp := x ↑ p × (infsum×term/(p×temp)+finsum×term1×
(1−x) ↑ q/(q×temp1))/factorial(p−1);
```

inclusive, by

```
temp := x ↑ p × (infsum×exp(loggamma(qrecur+p)−loggamma
(qrecur)−loggamma(p+1.0))+finsum×(1.0−x) ↑ q
×exp(loggamma(p+q)−loggamma(p)−loggamma(q+1.0)));.
```

This also means that the declarations of *temp*1 and *term*1 are not required. For even moderately large values of p or q this will also have the effect of speeding up the algorithm [see Remark on Algorithm 291, M. C. Pike and I. D. Hill, *Comm. ACM 9* (Sept. 1966), 685].

The following real procedures use this algorithm to evaluate three of the most frequently required statistical distribution functions.

```
real procedure Ftail (k, f1, f2, epsilon);
value k, f1, f2, epsilon;  real k, f1, f2, epsilon;
comment Ftail evaluates the probability that a random variable
  following an F distribution, on f1 and f2 degrees of freedom, ex-
  ceeds a positive constant k;
  Ftail := incompletebeta(f2/(f2+f1×k), 0.5×f2, 0.5×f1, epsilon);

real procedure Student(k, f, epsilon);
value k, f, epsilon;  real k, f, epsilon;
comment Student evaluates the probability that the absolute
  value of a random variable following a t distribution, on f de-
  grees of freedom, exceeds a positive constant k;
```

```
Student := incompletebeta(f/(f+k ↑ 2), 0.5×f, 0.5, epsilon);
real procedure Binomial(k, n, p, epsilon);
value k, n, p, epsilon; real k, n, p, epsilon;
comment Binomial evaluates the probability that a random
  variable following a binomial distribution, with parameters n
  and p, takes a value greater than or equal to k;
  Binomial := incompletebeta(p, k, n−k+1.0, epsilon);
```

CERTIFICATION OF ALGORITHM 253 [F2] EIGENVALUES OF A REAL SYMMETRIC MATRIX BY THE QR METHOD [P. A. Businger, *Comm. ACM 8* (April 1965), 217]

JOHN H. WELSCH (Recd. 3 June 1965, 1 Aug. 1966 and 1 Mar. 1967)
Stanford Linear Accelerator Center, Stanford, California

The procedure *symmetric QR* 1 was transcribed into ALGOL for the Burroughs B5500 (39-bit mantissa) and tested with no syntax or logic changes (except to change the tolerance from 2.25_{10}–22 to 3.35_{10}–24). The eigenvalues of the matrix in the example given in the procedure declaration were found to 15 units in the 11th significant place and in the order given.

Two defects of this algorithm have been found (personal communication from Prof. W. Kahan); one concerning the convergence, the other concerning the numerical stability.

The procedure *symmetric QR* 1 was slow to converge on matrices of large order with the form

$$\begin{bmatrix} 0 & 1 & & & & \\ 1 & 0 & 1 & & & \\ & 1 & 0 & 1 & & \\ & & \cdot & & & \\ & & & \cdot & & \\ & & & & \cdot & \\ & & & & 1 & 0 \end{bmatrix}$$

The trouble is caused by a poor choice of the shift, *lambda*, for accelerating convergence. The fault was corrected as described in the Certification of Algorithm 254.

The second defect is not as easy to detect or correct. On matrices of large order with pairs of eigenvalues of opposite sign, members of the pairs were found to varying accuracy. Another indication of an instability was a distinct jump in the computed values of the eigenvalues of the matrix

$$\begin{bmatrix} x & 1 & & \\ 1 & 1 & 1 & \\ & 1 & -x & 1 \\ & & 1 & -1 \end{bmatrix}$$

at $x = 10^{-5}$, as x was given the values $10^{-3}, 10^{-4}, \cdots, 10^{-11}$.

It appears that the square-root-free *QR* Algorithm described by Ortega and Kaiser ("The LL^T and *QR* methods for symmetric tridiagonal matrices," *Comput. J. 6* (1963), 99–101) is numerically unstable; therefore Algorithm 253 should be avoided. [Rutishauser (Letter to the Editor, *Comput. J. 6* (1963), 133) suggested a modification which is also mentioned by Wilkinson (*The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965, p. 567). However, even with this modification the Algorithm is numerically unstable as was pointed out in a private communication from Wilkinson to Kahan (1966)—REF.]

CERTIFICATION OF ALGORITHM 254 [F2] EIGENVALUES AND EIGENVECTORS OF A REAL SYMMETRIC MATRIX BY THE QR METHOD [P. A. Businger, *Comm. ACM 8* (April 1965), 218]

JOHN H. WELSCH (Recd. 3 June 1965, 1 Aug. 1966 and 1 Mar. 1967)
Stanford Linear Accelerator Center, Stanford, California

The procedure *symmetric QR* 2 was transcribed into ALGOL for the Burroughs B5500 (39-bit mantissa) and tested with no syntax or logic changes (except to change the tolerance from $1.5_{10} - 11$ to $1.83_{10} - 12$). The eigenvalues of the matrix given in the initial comment of the procedure declaration were found to 8 units in the 11th significant place and in the order given. The components of the eigenvectors found by the procedure differed from those given by at most 7 units in the 10th significant place and that occurred in the smallest component of X_2. The computed vectors X_3 and X_4 were the negative of those given.

It was found (personal communication from Prof. W. Kahan, University of Toronto) that *symmetric QR* 2 was slow to converge on matrices of large order with the form

$$\begin{bmatrix} 0 & 1 & & & & \\ 1 & 0 & 1 & & & \\ & 1 & 0 & 1 & & \\ & & \cdot & & & \\ & & & \cdot & & \\ & & & & \cdot & \\ & & & & 1 & 0 \end{bmatrix}$$

The trouble observed seems to be caused by a poor choice of the shift, *lambda*, for accelerating convergence. The following change corrects this fault and did not change the results of these tests except that the eigenvalues are found in a different order. Replace the 8 lines following the line labeled *inspect* by:

```
if abs(b[k]) ≦ eps then
  begin g[m, m] := a[m]; m := k; go to inspect end;
for i := i − 1 while abs(b[i]) > eps do k := i;
comment find eigenvalues of lower 2 × 2;
b0 := b[m1] ↑ 2; a1 := sqrt((a[m1]−a[m])↑2+4×b0);
t := a[m1] × a[m] − b0; a0 := a[m1] + a[m];
lambda := 0.5 × (if a0≧0 then a0+a1 else a0−a1);
t := t/lambda; comment compute the shift;
if abs(t−mu) < 0.5 × abs(t) then mu := lambda := t
else if abs(lambda−mu) < 0.5 × abs(lambda) then mu := lambda
else begin mu := t; lambda := 0 end;
a[k] := a[k] − lambda; beta := b[k];
```

The modified procedure (called *QR* 2 below) was compared with the procedures given by J. H. Wilkinson [*Numer. Math. 4* (1962), 354–376] of the Householder tridiagonalization, Sturm sequence bisection, and inverse iteration algorithms. Evaluation of the Sturm sequence caused exponent underflows and overflows, so the procedures were modified (referred to as *HSI* below) by scaling and overflow detection.

To measure the effectiveness of the procedures, two quantities, E_1 and E_2, were evaluated for each of eleven matrices used as test data. These quantities are suggested by Prof. W. Kahan (in "Inclusion Theorems for Clusters of Eigenvalues of Hermitian Matrices," University of Toronto, Feb. 1967) and are defined as follows. Let A be a Hermitian matrix, Λ a diagonal matrix of its approximate eigenvalues and V a matrix whose columns are approximate eigenvectors ordered to correspond with Λ. Define $W = V^*V - I$ and $R = AV - V\Lambda$, then

$$E_1 = \| W \|_2 \text{ and } E_2 = \| R \|_2 / \| \Lambda \|_2 ,$$

where $\| X \|_2^2 =$ maximum eigenvalue of X^*X. Then it is shown that the maximum absolute error in an eigenvalue is less than or

equal to

$$\frac{E_2 \| A \|_2}{\sqrt{1 - E_1}} \quad \text{if } E_1 < 1.$$

The computation of W and R was done with double-precision inner products.

The results of the tests are summarized as follows:

(a) Both *QR* 2 and *HSI* found the dominant eigenvalues to better relative accuracy, but the same or worse absolute accuracy than the other eigenvalues.

(b) *QR* 2 was on the average 1.8 times faster than *HSI* (*QR* 2 required 2.5 seconds on a Hilbert segment of order 15).

(c) *QR* 2 always found orthogonal eigenvectors ($E_1 \sim 10^{-11}$);

(d) in most cases $E_1 \sim 10^{-11}$ for *HSI* also, but several times *HSI* found two eigenvectors almost parallel ($E_1 \sim 1.0$).

(e) $E_2 \sim 10^{-11}$ for both *QR* 2 and *HSI* with neither being consistently better than the other.

Conclusions. The orthonormalized eigenvectors, speed, and comparable accuracy would recommend *symmetric QR* 2 over the Wilkinson procedures for finding all of the eigenvalues and eigenvectors of a real symmetric matrix. The latter procedures are good for finding selected eigenvalues and eigenvectors.

REMARK ON ALGORITHM 296 [E2] GENERALIZED LEAST SQUARES FIT BY ORTHOGONAL POLYNOMIALS

[G. J. Makinson, *Comm. ACM 10* (Feb. 1967), 87]

G. J. Makinson (Recd. 21 Mar. 1967)
University of Liverpool, Liverpool 3, England

The second sentence of the first comment should read "The weights should be provided inversely proportional to the square of the standard error of the observations."

instead of

"The weights should be provided inversely proportional to the standard error of the observations."

REMARKS ON: ALGORITHM 123 [S15] REAL ERROR FUNCTION, ERF(x)

[Martin Crawford and Robert Techo *Comm. ACM 5* (Sept. 1962), 483]

ALGORITHM 180 [S15] ERROR FUNCTION—LARGE X

[Henry C. Thacher Jr. *Comm. ACM 6* (June 1963), 314]

ALGORITHM 181 [S15] COMPLEMENTARY ERROR FUNCTION—LARGE X

[Henry C. Thacher Jr. *Comm. ACM 6* (June 1963), 315]

ALGORITHM 209 [S15] GAUSS

[D. Ibbetson. *Comm. ACM 6* (Oct. 1963), 616]

ALGORITHM 226 [S15] NORMAL DISTRIBUTION FUNCTION

[S. J. Cyvin. *Comm. ACM 7* (May 1964), 295]

ALGORITHM 272 [S15] PROCEDURE FOR THE NORMAL DISTRIBUTION FUNCTIONS

[M. D. MacLaren. *Comm. ACM 8* (Dec. 1965), 789]

ALGORITHM 304 [S15] NORMAL CURVE INTEGRAL

[I. D. Hill and S. A. Joyce. *Comm. ACM 10* (June 1967), 374]

I. D. Hill and S. A. Joyce (Recd. 21 Nov. 1966)
Medical Research Council,
Statistical Research Unit, 115 Gower Street, London W.C.1., England

These algorithms were tested on the ICT Atlas computer using the Atlas Algol compiler. The following amendments were made and results found:

ALGORITHM 123

(i) **value** x; was inserted.

(ii) $abs(T) \leqslant_{10} -10$ was changed to $Y - T = Y$ both these amendments being as suggested in [1].

(iii) The labels 1 and 2 were changed to $L1$ and $L2$, the **go to** statements being similarly amended.

(iv) The constant was lengthened to 1.12837916710.

(v) The extra statement $x := 0.707106781187 \times x$ was made the first statement of the algorithm, so as to derive the normal integral instead of the error function.

The results were accurate to 10 decimal places at all points tested except $x = 1.0$ where only 2 decimal accuracy was found, as noted in [2]. There seems to be no simple way of overcoming the difficulty [3], and any search for a method of doing so would hardly be worthwhile, as the algorithm is slower than Algorithm 304 without being any more accurate.

ALGORITHM 180

(i) $T := -0.56418958/x/exp(v)$ was changed to $T := -0.564189583548 \times exp(-v)/x$. This is faster and also has the advantage, when v is very large, of merely giving 0 as the answer instead of causing overflow.

(ii) The extra statement $x := 0.707106781187 \times x$ was made as in (v) of Algorithm 123.

(iii) **for** $m := m + 1$ was changed to **for** $m := m + 2$. $m+1$ is a misprint, and gives incorrect answers.

The greatest error observed was 2 in the 11th decimal place.

ALGORITHM 181

(i) Similar to (i) of Algorithm 180 (except for the minus sign).

(ii) Similar to (ii) of Algorithm 180.

(iii) m was declared as **real** instead of **integer**, as an alternative to the amendment suggested in [4].

The results were accurate to 9 significant figures for $x \leqslant 8$, but to only 8 significant figures for $x = 10$ and $x = 20$.

ALGORITHM 209

No modification was made. The results were accurate to 7 decimal places.

ALGORITHM 226

(i) $10 \uparrow m/(480 \times sqrt(2 \times 3.14159265))$ was changed to $10 \uparrow m \times 0.000831129750836$.

(ii) **for** $i := 1$ **step** 1 **until** $2 \times n$ **do** was changed to $m := 2 \times n$; **for** $i := 1$ **step** 1 **until** m **do**.

(iii) $-(i \times b/n) \uparrow 2/8$ was changed to $-(i \times b/n) \uparrow 2 \times 0.125$.

(iv) **if** $i = 2 \times n - 1$ was changed to **if** $i = m - 1$

(v) $b/(6 \times n \times sqrt(2 \times 3.14159265))$ was changed to $b/(15.0397696478 \times n)$.

Tests were made with $m = 7$ and $m = 11$ with the following results:

x	*Number of significant figures correct*		*Number of decimal places correct*	
	$m = 7$	$m = 11$	$m = 7$	$m = 11$
−0.5	7	11	7	11
−1.0	7	10	7	10
−1.5	7	10	8	10
−2.0	7	9	8	10
−2.5	6	9	8	11
−3.0	6	7	8	9
−4.0	5	7	10	11
−6.0	2	1	12	10
−8.0	0	0	11	9

Perhaps the comment with this algorithm should have referred to decimal places and not significant figures. To ask for 11 significant figures is stretching the machine's ability to the limit, and where 10 significant figures are correct, this may be regarded as acceptable.

ALGORITHM 272

The constant .99999999 was lengthened to .9999999999.

The accuracy was 8 decimal places at most of the points tested, but was only 5 decimal places at $x = 0.8$.

ALGORITHM 304

No modification was made. The errors in the 11th significant figure were:

$abs(x)$	$x > 0 \equiv upper$	$x > 0 \not\equiv upper$
0.5	1	1
1.0	1	2
1.5	21[a](5)	2
2.0	25[a](0)	4
3.0	0	0
4.0	2	3
6.0	6	0
8.0	14	0
10.0	23	0
20.0	35	0

[a] Due to the subtraction error mentioned in the comment section of the algorithm. Changing the constant 2.32 to 1.28 resulted in the figures shown in brackets.

To test the claim that the algorithm works virtually to the accuracy of the machine, it was translated into double-length instructions of Mercury Autocode and run on the Atlas using the EXCHLF compiler (the constant being lengthened to 0.39894228040143267793994 6). The results were compared with hand calculations using Table II of [5]. The errors in the 22nd significant figure were:

$abs(x)$	$x > 0 \equiv upper$	$x > 0 \not\equiv upper$
1.0	2	3
2.0	7	1
4.0	2	0
8.0	8	0

Timings. Timings of these algorithms were made in terms of the Atlas "Instruction Count," while evaluating the function 100 times. The figures are not directly applicable to any other computer, but the relative times are likely to be much the same on other machines.

INSTRUCTION COUNT FOR 100 EVALUATIONS

$abs(x)$	Algorithm number							
	123	180	181	209	226 $m = 7$	272	304[a]	304[b]
0.5	58			8	97	24	25	24
1.0	65[c]			8	176	24	29	29
1.5	164	128	127	9	273	25	35	35
2.0	194	78	90	8	387	24	39	39
2.5	252	54	68	10	515	24	131	44
3.0		42	51	9	628	25	97	50
4.0		27	39	9	900[d]	25	67	44
6.0		15	30	6	1400[d]	16	49	23
8.0		9	28	7	2100[d]	18	44	11
10.0		10	25	5	2700[d]	16	38	11
20.0		9	22	5	6500[d]	16	32	11
30.0		9	9	5	10900[d]	16	11	11

[a] Readings refer to $x > 0 \equiv upper$.

[b] Readings refer to $x > 0 \not\equiv upper$.

[c] Time to produce incorrect answer. A count of 120 would fit a smooth curve with surrounding values.

[d] 100 times Instruction Count for 1 evaluation.

Opinion. There are advantages in having two algorithms available for normal curve tail areas. One should be very fast and reasonably accurate, the other very accurate and reasonably fast. We conclude that Algorithm 209 is the best for the first requirement, and Algorithm 304 for the second.

Algorithms 180 and 181 are faster than Algorithm 304 and may be preferred for this reason, but the method used shows itself in Algorithm 181 to be not quite as accurate, and the introduction of this method solely for the circumstances in which Algorithm 180 is applicable hardly seems worth while.

Acknowledgment. Thanks are due to Miss I. Allen for her help with the double-length hand calculations.

REFERENCES:

1. THACHER, HENRY C. JR. Certification of Algorithm 123. *Comm. ACM 6* (June 1963), 316.

2. IBBETSON, D. Remark on Algorithm 123. *Comm. ACM 6* (Oct. 1963), 618.

3. BARTON, STEPHEN P., AND WAGNER, JOHN F. Remark on Algorithm 123. *Comm. ACM 7* (Mar. 1964), 145.

4. CLAUSEN, I., AND HANSSON, L. Certification of Algorithm 181. *Comm. ACM 7* (Dec. 1964), 702.

5. SHEPPARD, W. F. *The Probability Integral.* British Association Mathematical Tables VII, Cambridge U. Press, Cambridge, England, 1939.