inadequacy of the three-point fit [eq. (14)] depends on the function. With ($\sec^2 \pi X$) an error of 10^{-5} in the position of the pole would be needed to make the total error at the point nearest the pole ($X = 0.46$, i.e., 84°) significantly greater than that shown in Figures 1 and 2; at $X = 0.40$ (i.e., 72°) an error of as much as 10^{-3} could be tolerated. In both cases the errors would be very small compared with the errors given by the Simpson formula.

It thus appears that there should generally be no difficulty in knowing the position of the pole accurately enough, although such an error could limit the closeness of approach to the pole.

5. Conclusion

A method has been described for numerically integrating functions that have poles outside the range of integration, and explicit formulas have been given. These expressions have been given in well-conditioned form and are easy to use on an automatic computer. The accuracy of the method has been discussed for a pole of second order.

The "pole" method and the "half-Simpson" (which is the comparable polynomial formula) were tested on the integration of ($\sec^2 \pi X$), $0 < X < 0.5$, considering the pole at $X = 0.5$ only. The "pole" method was superior for $X > 0.1$ and far superior for $X > 0.2$, that is, within 54° of the pole. The new method could be used throughout the range, and was still very accurate close to the pole, where the Simpson formula is useless.

While we have dealt explicitly only with functions having a single pole, the method described in Section 2 could be used to derive formulas for a function having any finite number of poles whose positions and orders are known. The success of the formulas in the integration of ($\sec^2 \pi X$), which has poles in particular at $X = \pm\frac{1}{2}$, suggests that the formulas derived for a single pole can be useful even for functions having more than one pole, as long as the nearest pole is used in eqs. (2), (3), (8), etc.

It has been shown that the method described and the formulas given will allow straightforward numerical integration of functions with the most commonly occurring kinds of singularities, without detailed analysis of the functions.

Acknowledgments. I would like to thank Miss Rochelle Kellman who wrote the computer programs; and R. W. Hamming for helpful criticisms.

RECEIVED AUGUST, 1966; REVISED JANUARY, 1967

REFERENCES

1. DAVIS, P. J., AND RABINOWITZ, P. Ignoring the singularity in approximate integration. *J. SIAM Numer. Anal. Ser. B, 2* (1965), 367–383.
2. HAMMING, R. W. *Numerical Methods for Scientists and Engineers.* McGraw-Hill, New York, 1962.
3. DWIGHT, H. B. *Tables of Integrals and other Mathematical Data.* Macmillan, New York, 3rd Ed., 1957, Art. 601.4.
4. JEFFREYS, SIR H., AND JEFFREYS, LADY B. S. *Methods of Mathematical Physics.* Cambridge U. Press, London, 1962, 3rd ed.

Algorithms

J. G. HERRIOT, Editor

ALGORITHM 299
CHI-SQUARED INTEGRAL [S15]

I. D. HILL AND M. C. PIKE (Recd. 9 Sept. 1965 and 3 Oct. 1966)
Medical Research Council, Statistical Research Unit, 115 Gower St., London W.C.1., England

real procedure *chiprob* (*x, f, bigx, normal, wrong*);
value *x, f, bigx*; **real** *x*; **integer** *f*; **Boolean** *bigx*;
real procedure *normal*; **label** *wrong*;
comment Finds the probability that χ^2, on f degrees of freedom exceeds x, i.e.,

$$\frac{1}{2^{\frac{1}{2}f}\Gamma(\frac{1}{2}f)} \int_x^\infty z^{\frac{1}{2}f-1}\, e^{-\frac{1}{2}z}\, dz \quad (x \geqq 0, \ f \geqq 1)$$

The algorithm is based upon the recurrence formula

$$P\,(\chi_f{}^2 > x) = P\,(\chi_{f-2}^2 > x) + \frac{(\frac{1}{2}x)^{\frac{1}{2}f-1}\, e^{-\frac{1}{2}x}}{\Gamma(\frac{1}{2}f)}$$

[*Handbook of Mathematical Functions*, National Bureau of Standards, Appl. Math. Series 55 (1964), formula 26.4.8] by means of which any χ^2-integral can be reduced to the sum of

(i) a series of terms that can be directly evaluated, and
(ii) a χ^2-integral on 2 degrees of freedom (if f is even), or on 1 degree of freedom (if f is odd).

To evaluate (ii) we have either

$$P\,(\chi_2{}^2 > x) = e^{-\frac{1}{2}x}$$

or

$$P\,(\chi_1{}^2 > x) = (2/\sqrt{2\pi}) \int_{\sqrt{x}}^\infty e^{-\frac{1}{2}z^2}\, dz.$$

The evaluation of the latter expression is performed by the formal **real procedure** *normal* which must evaluate the lower tail area of the standardized normal curve (**real procedure** *Gauss* [D. Ibbetson, Alg. 209, *Comm. ACM 6* (Oct. 1963), 616] may be used as the actual parameter).

The parameter *bigx* should be set to **true** if the value of x is too big for *exp* $(-0.5\times x)$ to be accurately represented by the machine, or **false** otherwise.

For even degrees of freedom the method is exact, and the algorithm is essentially accurate to the accuracy of the machine. For odd degrees of freedom the accuracy will be dictated by the accuracy of the **real procedure** *normal*.

For large degrees of freedom, if speed is more important than great accuracy, it may be found preferable to use an approximation, e.g., the Wilson-Hilferty cubic formula [Wilson, E. B., and Hilferty, M. M., *Proc. Nat. Acad. Sci. 17* (1931), 684] which may be expressed as

chiprob := *normal* $(-$*sqrt* $(4.5\times f)\times((x/f)\uparrow(1/3)+2/(9\times f)-1))$.

This is accurate to 3 decimal places for $f > 40$.

The authors thank the referee and the editor for helpful criticisms and suggestions;

```
begin
  if x < 0 ∨ f < 1 then go to wrong else
  begin
    real a, y, s;
    Boolean even;
    a := 0.5 × x;  even := 2 × (f÷2) = f;
    if even ∨ f > 2 ∧ ¬bigx then y := exp(−a);
    s := if even then y else 2.0 × normal (−sqrt (x));
    if f > 2 then
    begin
      real e, c, z;
      x := 0.5 × (f−1.0);  z := if even then 1.0 else 0.5;
      if bigx then
      begin
        e := if even then 0 else 0.572364942925;  c := ln (a);
        comment 0.572364942925 = ln (sqrt(π));
        for z := z step 1.0 until x do
        begin
          e := ln (z) + e;
          s := exp(c×z−a−e) + s
        end;
        chiprob := s
      end else
      begin
        e := if even then 1.0 else 0.564189583548/sqrt(a);  c := 0;
        comment 0.564189583548 = 1/sqrt(π);
        for z := z step 1.0 until x do
        begin
          e := e × a/z;
          c := c + e
        end;
        chiprob := c × y + s
      end
    end else chiprob := s
  end
end chiprob
```

ALGORITHM 300
COULOMB WAVE FUNCTIONS [S22]

J. H. Gunn (Recd. 19 Feb. 1965)
Nordisk Institut for Teoretisk Atomfysik
Blegdamsvej 15, Copenhagen, Denmark

```
procedure Coulomb(F, Fd, G, Gd, sig, rho, eta, lmax, exit);
  value rho, eta, lmax;
  real rho, eta; integer lmax; array F, Fd, G, Gd, sig; label
    exit;
```

comment The Coulomb wave functions F_L and G_L are defined as the two independent solutions of the differential equation

$$\frac{d^2y}{d\rho^2} + (1-2\eta/\rho-L(L+1)/\rho^2)y = 0$$

having the asymptotic behavior for large ρ

$$F_L \sim \sin\left(\rho-\eta \ln 2\rho-\frac{L}{2}\pi+\sigma_L\right),$$

$$G_L \sim \cos\left(\rho-\eta \ln 2\rho-\frac{L}{2}\pi+\sigma_L\right)$$

where $\sigma_L = \arg \Gamma\ (i\eta+L+1)$. The procedure calculates for a given ρ = *rho* and η = *eta*, the functions F_L and G_L, their derivatives F_L' and G_L', and σ_L for all L from 0 up to *lmax* (>0) and places the results in the arrays *F*, *G*, *Fd*, *Gd*, *sig* respectively, which must have bounds 0:*lmax*. *rho* must lie in the range 5–30 and *eta* in the range 0.1–30: values outside this range cause the procedure to leave via the label *exit*. This range is one that is often used in scattering and reaction problems in physics. Details of the methods used are to be found in: C. E. Fröberg, "Numerical treatment of Coulomb wave functions," *Rev. Mod. Phys.* *27* (1955), 399–411, and in: H. F. Lutz and M. D. Karvelis, "Numerical calculation of Coulomb wave functions for repulsive Coulomb fields," *Nucl. Phys.* *43* (1963), 31–44. The author gratefully acknowledges the extensive help of Miss Margaret Wirt in the preparation of this procedure;

```
begin
  integer n;  real rhom;
  comment jump to label exit if rho and eta lie outside range of
    procedure;
  if rho < 5 ∨ rho > 30 ∨ eta < 0.1 ∨ eta > 30 then
    go to exit;
  begin real sto;  integer i;
    comment phase shifts σL are calculated using formulae
      44–45 of Lutz and Karvelis;
    sto := 16 + eta ↑ 2;
    sig[0] := −eta + eta/2 × ln(sto) + 3.5 × arctan(eta/4) −
      (arctan(eta)+arctan(eta/2)+arctan(eta/3)) − eta/(12×sto) ×
      (1+1/30 × (eta↑2−48)/sto↑2 + 1/105
      × (eta↑4−160×eta↑2+1280)/sto↑4);
    for i := 1 step 1 until lmax do
      sig[i] := sig[i−1] + arctan(eta/i)
  end;
  if rho ≦ (5×eta−15)/3 ∨ rho ≦ eta then
  begin comment G[0] and Gd[0] are calculated using the Riccati
    method (ρ<2η) ref. formulae 9.1–9.4, Fröberg;
    integer i;  real q, psi, psid, f;  array g, gd[0:7], t, s[1:10];
    t[1] := rho/(2×eta);  s[1] := 1 − t[1];  q := sqrt(t[1]×s[1]);
    for i := 2 step 1 until 10 do
    begin t[i] := t[1] × t[i−1];
      s[i] := s[1] × s[i−1]
    end;
    g[0] := q + arctan(t[1]/q) − 1.5707963;
    g[1] := 0.25 × ln(t[1]/s[1]);
    g[2] := −(8×t[2]−12×t[1]+9)/(48×q×s[1]);
    g[3] := (8×t[1]−3)/(64×t[1]×s[3]);
    g[4] := (2048×t[6]−9216×t[5]+16128×t[4]−13440×t[3]−12240
      ×t[2]+7560×t[1]−1890)/(92160×q×t[1]×s[4]);
    g[5] :=  3×(1024×t[3]−448×t[2]+208×t[1]−39)/(8192×t[2]×
      s[6]);
    g[6] = −(262144×t[10]−1966080×t[9]+6389760×t[8]−11714560
      ×t[7]+13178880×t[6]−9225216×t[5]+13520640×t[4]−
      3588480×t[3]+2487240×t[2]−873180×t[1]+130977)/(10321920
      ×q×t[2]×s[7]);
    g[7] := (1105920×t[5]−55296×t[4]+314624×t[3]−159552×t[2]
      +45576×t[1]−5697)/(393216×t[3]×s[9]);
    gd[0] := q/t[1];
    gd[1] := 0.25/(t[1]×s[1]);
    gd[2] := −(8×t[1]−3)/(32×q×t[1]×s[2]);
    gd[3] := 3×(8×t[2]−4×t[1]+1)/(64×t[2]×s[4]);
    gd[4] := −(1536×t[3]−704×t[2]+336×t[1]−63)/(2048×q×t[2]
      ×s[5]);
    gd[5] := 3×(2560×t[4]−832×t[3]+728×t[2]−260×t[1]+39)/
      (4096×t[3]×s[7]);
    gd[6] := (−368640×t[5]−30720×t[4]+114944×t[3]−57792×t[2]
      +16632×t[1]−2079)/(65536×q×t[3]×s[8]);
    gd[7] := 3×(860160×t[6]+196608×t[5]+308480×t[4]−177280×
      t[3]+73432×t[2]−17724×t[1]+1899)/(131072×t[4]×s[10]);
    f := 2 × eta;  psi := psid := 0;  q := −1;
    for i := 0 step 1 until 7 do
```

```
    begin psi := psi + q × f × g[i];
      psid := psid + q × f × gd[i];
      f := f/(2×eta);   q := −q
    end;
    G[0] := exp(psi);   Gd[0] := G[0] × psid/(2× eta);   rhom :=
      rho
end else
if rho ≧ (30×eta+75)/13 ∧ rho < 2 × eta ↑ 2 then

begin comment  G[0] and Gd[0] are calculated using the second
    Riccati method (2η< ρ) ref. formulae 9.6–9.8, Fröberg;
  integer i;   real A, B, psi, phi, M, q;   array x, y, e[1:10];
  x[1] := 2 × eta/rho;   y[1] := 1 − x[1];   q := sqrt(y[1]);   e[1]
    := 2 × eta;
  for i := 2 step 1 until 10 do
  begin x[i] := x[1] × x[i−1];   e[i] := e[1] × e[i−1];
    y[i] := y[1] × y[i−1]
  end;
  psi := −(8×x[3]−3×x[4])/(64×e[2]×y[3])  +  3  ×  x[5]  ×
    (1024 − 448×x[1]+208×x[2]−39×x[3])/(8192×e[4]×y[6])  −
    x[7]  ×  (1105920−55296×x[1]+314624×x[2]−159552×x[3]+
    45576×x[4]−5697×x[5])/(393216×e[6]×y[9]);
  phi := e[1] × (q/x[1] + 0.5 × ln((1−q)/(1+q))) + 0.7853982
    − (9×x[2]−12×x[1]+8)/(48×e[1]×q×y[1])
    − (2048−9216×x[1]+16128×x[2]−13440×x[3]−12240
    ×x[4]+7560×x[5]−1890×x[6])/(92160×e[3]×q×y[4])
    − (130977×x[10]−873180×x[9]+2487240×x[8]−3588480
    ×x[7]+13520640×x[6]−9225216×x[5]+15178880 × x[4]
    −11714560×x[3]+6389760×x[2]−1966080×x[1]
    +262144)/(10321920×e[5]×q×y[7]);
  A := q/x[2] + (8×x[1]−3×x[2])/(32×e[2]×q×y[2])
    − x[3] × (1536−704×x[1]+336×x[2]−63×x[3])/
    (2048×e[4]×q×y[5]) + x[5] × (368640−30720×x[1]
    +114944×x[2]−57792×x[3]+16632×x[4]− 2079×x[5])/
    (65536×e[6]×q×y[8]);
  B := 1/(4×e[1]×y[1]) − 3 × x[2] × (x[2]−4×x[1]+8)/
    (64×e[3]×y[4]) + 3 × x[4] × (2560−832×x[1]+728
    ×x[2]−260×x[3]+39×x[4])/(4096×e[5]×y[7]) − 3
    × x[6] × (1899×x[6]−17724×x[5]+73432×x[4]−177280
    ×x[3]+308480×x[2]+196608×x[1]+860160)/(131072
    ×e[7]×y[10]);
  M := sqrt(1/q) × exp(psi);
  G[0] := M × cos(phi);
  Gd[0] := −x[2] × (A×M×sin(phi)+B×G[0]);   rhom :=rho
end else
if eta < 4 then

begin comment  G[0] and Gd[0] are calculated using an asymp-
    totic expansion, ref. formulae 12.3–12.7, Fröberg;
  real ss, s1, tt, t1, SS, S1, TT, T1, sn, tn, Sn, Tn, An, Bn, theta,
    cth, sth;  integer i;
  rhom := if rho ≧ 2 × eta ↑ 2 then rho else 2 × eta ↑ 2;
  comment  a suitable value of rhom is chosen for which the
    expansion is valid;
  ss := sn := 1;   tt := tn := 0;
  SS := Sn := 0;   TT := Tn := 1 − eta/rhom;
  for i := 0 step 1 until 10, 11, i + 1 while (abs(sn)>10−7
    ×abs(ss)∨abs(tn)>10−7×abs(tt)∨abs(Sn)>10−7
    ×abs(SS)∨abs(Tn)>10−7×abs(TT))∧(abs(sn)
    <abs(s1)∧abs(tn)<abs(t1)∧abs(Sn)<abs(S1)∧ abs(Tn)
    <abs(T1)) do
  begin An := (2×i+1) × eta/(2×(i+1)×rhom);
          Bn := (eta↑2−i×(i+1))/(2×(i+1)×rhom);
           s1 := sn;   t1 := tn;   S1 := Sn;   T1 := Tn;
           sn := An × s1 − Bn × t1;
           tn := An × t1 + Bn × s1;
          Sn := An × S1 − Bn × T1 − sn/rhom;
          Tn := An × T1 + Bn × S1 − tn/rhom;
           ss := ss + sn;   tt := tt + tn;
          SS := SS + Sn;   TT := TT + Tn
  end;
  theta := −eta × ln(2×rhom) + rhom + sig[0];
  cth := cos(theta);   sth := sin(theta);
  G[0] := ss × cth − tt × sth;   Gd[0] := SS × cth − TT × sth
end else
begin comment  G[0] and Gd[0] are calculated on the transition
    line for rhom = 2 × eta, ref. formulae 10.3–10.4, Fröberg;
  G[0] := 1.22340416 × eta ↑ (1/6) × (1+0.0495957017/eta↑ (4/3)
      −0.0088888889/eta↑2+0.00245519918/eta↑ (10/3)
      −0.000910895806/eta↑4+0.000253468412/eta↑ (16/3));
  Gd[0] := −.707881773 × eta ↑ (−1/6) × (1−0.172826037/
    eta↑ (2/3)+0.000317460317/eta↑2−0.00358121485/eta↑ (8/3)
    +0.000311782468/eta↑4−0.000907396643/eta↑ (14/3));
  rhom := 2 × eta
end;
if rhom ≠ rho then

begin comment  Integrate the solutions G[0] and Gd[0] from
    the value of rhom at which they were evaluated to the value
    of rho required using Runge-Kutta formula;
  integer nh, i;   real k1, k2, k3, k4, k1p, k2p, k3p, k4p, y, yp,
    x, h;
  nh := entier(abs(rhom−rho)×10+1);
  h := (rho−rhom)/nh;
  x := rhom;   y := G[0];   yp := Gd[0];
  for i := 1 step 1 until nh do
  begin k1 := h × yp;   k1p := −h × (1−2×eta/x) × y;
    k2 := h × (yp+k1p/2);   k2p := −h × (1−2×eta/ (x+h/2))
    × (y+k1/2);
    k3 := h × (yp+k2p/2);   k3p := −h × (1−2×eta/ (x+h/2))
    × (y+k2/2);
    k4 := h × (yp+k3p);   k4p := −h × (1−2×eta/(x+h)) ×
    (y+k3); y := y + (k1+2×k2+2×k3+k4)/6;
    yp := yp + (k1p+2×k2p+2×k3p+k4p)/6;
    x := x + h
  end;
  G[0] := y;   Gd[0] := yp
end;
n := if rho > lmax then entier(rho+10) else lmax + 10;

begin comment  Use downward recurrence relation (Millers
    method) and normalisation condition to obtain solutions
    F[L];
  array f[0:n];   real fd0, alpha, sto;   integer L;
  f[n] := 0;
  f[n−1] := 1;
  for L := n − 1 step −1 until 1 do
    f[L−1] := L/sqrt(eta↑2+L↑2) × (((2×L+1)×eta/
      (L×(L+1))+(2×L+1)/rho)× f[L]−sqrt(eta↑2
      +(L+1)↑2)/(L+1)×f[L+1]);
  fd0 := (eta+1/rho) × f[0] − sqrt(eta↑2+1) × f[1];
  G[1] := (−Gd[0]+(1/rho+eta)×G[0])/sqrt(1+eta↑2);
  alpha := 1/(sqrt(1+eta↑2)×(f[0]×G[1]−f[1]×G[0]));
  F[0] := alpha × f[0];
  Fd[0] := alpha × fd0;
  comment  Upward  recurrence  relations  for  remaining
    solutions;
  for L := 0 step 1 until lmax−1 do
  begin F[L+1] := alpha × f[L+1];
    sto := sqrt(eta↑2+(L+1)↑2)/(L+1);
    Fd[L+1] := sto × F[L] − (eta/(L+1)+(L+1)/rho) ×
    F[L+1]; G[L+1] := 1/sto × ((eta/(L+1)+(L+1)/rho)
    ×G[L]−Gd[L]); Gd[L+1] := sto × G[L] − (eta/(L+1)+
    (L+1)/rho) × G[L+1]
  end
end
end Coulomb
```