Acknowledgment. The author would like to express his appreciation to K. R. Blake for many stimulating discussions on this subject. Also appreciated are valuable comments by Prof. D. E. Knuth and the referees of this paper.

RECEIVED FEBRUARY, 1966; REVISED MAY, 1966

REFERENCES

1. Revised report on the algorithmic language ALGOL 60. *Comm. ACM 6* (Jan. 1963), 1–17.
2. WIRTH, N. A generalization of ALGOL. *Comm. ACM 6* (Sept. 1963), 547–554.
3. IBM Operating System/360. PL/I: language specifications. Form C28-6571, IBM Corp., 1966.
4. MCILROY, M. DOUGLAS. Macro extensions of compiler languages. *Comm. ACM 3* (Apr. 1960), 214–220.
5. FERGUSON, DAVID E. The evolution of the meta-assembly program. *Comm. ACM 9* (March 1966), 190–193.
6. CHOMSKY, NOAM, AND MILLER, GEORGE A. Introduction to the formal analysis of natural languages. *Handbook of Mathematical Psychology, Vol. II.* John Wiley, New York, 1963, pp. 283–306.
7. SCHORRE, D. V. Meta II. A syntax oriented compiler writing language. Proc. ACM 19th Nat. Conf., Philadelphia, Pa., Aug. 1964, ACM Publ. P-64.
8. CONWAY, MELVIN E. Design of a separable transition-diagram compiler. *Comm. ACM 6* (July 1963), 396–408.

LARSEN—cont'd from p. 789

an effective data processing technique. The physical realization of this concept has been seen to be highly modular and suitable for programming implementation.

The Data Filter behaves as a two-port data filter within an interpretive processing environment, and represents the physical realization of the data filtering concept. This is accomplished by associating format declarations with its input and output ports which define its processing characteristics. The desired data string is sequentially constructed in the OUT buffer by filtering datum in the IN or HOLD buffers through these format declarations. A Procedural Controller is employed to synchronize loading of the IN buffer and dumping of the OUT buffer to achieve specific data processing results as implied by the job being processed.

RECEIVED FEBRUARY, 1966; REVISED JUNE, 1966

REFERENCES

1. Functional design specification of a data retrieval model (ABACUS). SID 66-175-2, North American Aviation, Inc., Downey, Calif., March 1966, 156pp.
2. ABACUS user's manual. SID 66-175-1, North American Aviation, Inc., Downey, Calif., March 1966, 148pp.

Algorithms

J. G. HERRIOT, Editor

Algorithms Policy as revised in August, 1966 to include FORTRAN appears on page 823.

ALGORITHM 292
REGULAR COULOMB WAVE FUNCTIONS

WALTER GAUTSCHI (Recd. 8 Oct. 1965)
Purdue University, Lafayette, Indiana and Argonne National Laboratory, Argonne, Illinois
Work performed under the auspices of the U. S. Atomic Energy Commission.

```
real procedure t(y); value y; real y;
comment This procedure evaluates the inverse function t = t(y)
  of y = t ln t in the interval y ≧ −1/e, to an accuracy of about
  4 percent, or better. Except for the addition of the case
  −1/e ≦ y ≦ 0, and an error exit in case y < −1/e, the procedure
  is identical with the real procedure t of Algorithm 236;
begin real p, z;
  if y < −.36788 then go to alarm 1;
  if y ≦ 0 then t := .36788 + 1.0422 × sqrt(y + .36788) else
  if y ≦ 10 then
  begin
    p := .000057941 × y − .00176148; p := y × p + .0208645;
    p := y × p − .129013; p := y × p + .85777;
    t := y × p + 1.0125
  end
  else
  begin
    z := ln(y) − .775; p := (.775−ln(z))/(1+z);
    p := 1/(1+p); t := y × p/z
  end
end t;
procedure minimal (eta, omega, eps, la1, dm);
  value eta, omega, eps; real eta, omega, eps, la1, dm;
comment This procedure assigns the value of λ1' to la1, accurately to within a relative error of eps, where {λL'} is the minimal
  solution (normalized by λ0'=1) of the difference equation

$$\lambda_{L+1} - \frac{2L+1}{L+1}\omega\lambda_L - \frac{L^2+\eta^2}{L(L+1)}\lambda_{L-1} = 0 \quad (\omega \neq 0).$$

  (For terminology, see [3].) If {λL} denotes the solution corresponding to initial values λ0 = 1, λ1 = ω − η, the procedure also
  assigns to dm the value λ1 − λ1'. The negative logarithm of
  |λ1 − λ1'| may be considered a measure of the "degree of minimality" of the solution {λL};
begin integer L, nu; real eta2, r, ra;
    eta2 := eta ↑ 2;
    nu := 20; ra := 0;
L1: r := 0;
    for L := nu step −1 until 1 do
    r := −(L ↑ 2+eta2)/(L×((2×L+1)×omega−(L+1)×r));
  if abs(r−ra) > eps × abs(r) then
  begin
    ra := r; nu := nu + 10; go to L1
  end;
  la1 := r; dm := omega − eta − r
end minimal;
```

```
procedure Coulomb (eta, ro, Lmax, d, F);
  value eta, ro, Lmax, d; integer Lmax, d; real eta, ro;
  array F;
comment This procedure generates to d significant digits the
  regular Coulomb wave functions F_L(η, ρ) for fixed η ≧ 0, ρ ≧ 0,
  and for L = 0(1)Lmax. (For notation, see [2, Ch. 14]). The
  results are put into the array F. Letting

  f_L = 2^L L! / ((2L)! C_L(η)) F_L(η, ρ),   C_L(η) = 2^L e^{-πη/2} |Γ(L + 1 + iη)| / (2L + 1)!,

  the procedure first obtains f_L as the minimal solution of the
  recurrence relation

  L[(L+1)^2 + η^2] / ((L+1)(2L+3)) y_{L+1} - [η + L(L+1)/ρ] y_L + (L+1)/(2L-1) y_{L-1} = 0,

  using for normalization the identity

      ρ e^{iωρ} = Σ_{L=0}^{∞} λ_L f_L ,   λ_L = i^L P_L^{(iη,-iη)}(-iω),

  where P_L^{(α,β)}(z) denotes the Jacobi polynomial of degree L. The
  parameter ω is so chosen as to avoid undesirable cancellation
  effects. The final results F_L are obtained recursively, by

  F_L(η, ρ) = c_L f_L ,

  c_L = (2L - 1) / (L(2L + 1)) [L^2 + η^2]^{1/2} c_{L-1}(L = 1, 2, 3 ···),   c_0 = (2πη / (e^{2πη} - 1))^{1/2} .

  A detailed justification of the process is to appear elsewhere
  ([3]). For large positive η and ρ, the generation of the coefficients
  λ_L is subject to some loss of accuracy. If 0 ≦ η ≦ 20, 0 ≦ ρ ≦ 20,
  none, or only a few decimal digits will be lost, however. Writing
  the procedure minimal in double precision will resolve the
  problem for η, ρ up to about 50, for normal accuracy require-
  ments. In any case, if higher precision is desirable, the procedure
  puts out a message to this effect. There is an error exit, if ρ < 0;
begin integer L, nu, nu1, mu, mu1, i, k;
  real epsilon, ro1, eta2, omega, d1, sum, r, r1, s, t1, t2;
  array lambda, lmin[0:1], Fapprox, Rr[0:Lmax];
  switch coefficients := L2, L1, M1;
  if ro < 0 then go to alarm2;
  if ro = 0 then
  begin
    for L := 0 step 1 until Lmax do F[L] := 0;
    go to L5
  end;
  epsilon := .5 × 10↑(-d);  ro1 := 1/ro;  eta2 := eta↑2;
  t1 := if eta > 0 then .5 × ro/eta else 0;
  omega := if eta < 1 then 0 else
  if t1 ≧ 1 then 1.570796327/t1 else
    (1.570796327 - arctan(sqrt(1/t1-1)) + sqrt(t1×(1-t1)))/t1;
  lambda [0] := lmin[0] := 1;  lambda[1] := omega - eta;
  sum := ro × exp(omega×ro);
  for L := 0 step 1 until Lmax do Fapprox[L] := 0;
  d1 := 2.3026 × d + 1.3863;
  t1 := 1.3591 × ro;
  L := if Lmax < t1 then 1 + entier(t1) else Lmax;
  t1 := exp(1.5708×eta);  s := sqrt(1+omega↑2);
  t1 := if omega = 0 then t1 + 1/t1 else
    exp(-eta×arctan(1/omega));
  t2 := omega + s;
  r := 1.3591 × ro × t2;
  s := (d1+ln(t1×sqrt(t2/s))-omega× ro)/r;
  nu := if s ≧ -.36788 then entier(r×t(s)) else 1;
  nu1 := entier(L×t(.5×d1/L));
  nu := if nu < nu1 then nu1 else nu;
  nu1 := 1;
  if omega = 0 then i := 1 else i := 2;
L0: begin own array lambda[0:nu];
  comment Dynamic own array declarations are not per-
    mitted in most of the current ALGOL compilers. It can be
    avoided here, at the cost of extra storage, if lambda is de-
    clared as an array of dimension [0:300] at the beginning of
    the procedure Coulomb. The same remark applies to the
    array lmin declared later in the block labeled M1;
  go to coefficients [i];
L1: minimal (eta, omega, 10-m, r1, d1);
  comment The letter m in 10-m is a place holder for a ma-
    chine-dependent integer, namely one less than the number of
    decimal digits carried in the precision mode (single, or
    double precision) of the procedure minimal. Similarly for
    the letter n in the next statement, which is a place holder
    for the integer m + 1. Both m and n are to be properly
    substituted by the user;
  if abs(d1×epsilon) ≧ 10-n then begin i := 1; go to L2 end;
  outstring (1, 'The requested accuracy cannot be guaranteed.
  Use of the procedure minimal in a higher precision mode
  appears indicated');
  i := 3;  mu1 := 0;
M1: begin array Rra, lam[0:nu]; own array lmin[0:nu];
    mu := entier (1.25×nu);
    for L := mu1 step 1 until nu do lam[L] := 0;
M2: r := 0;
    for L := mu step -1 until mu1 + 1 do
    begin
      r := -(L↑2+eta2)/(L×((2×L+1)×omega-(L+1)×r));
      if L ≦ nu then Rra[L-1] := r
    end;
    for L := mu1 + 1 step 1 until nu do
      lmin[L] := Rra[L-1] × lmin[L-1];
    for L := mu1 step 1 until nu do
    if abs(lmin[L]-lam[L]) > epsilon × abs(lmin[L]) then
    begin
      for k := mu1 step 1 until nu do lam[k] := lmin[k];
      mu := mu + 5;
      if mu < 5 × nu then go to M2 else
      begin
        outstring (1, 'convergence difficulty in the generation of
          the coefficients lambda sub L');
        go to L5
      end
    end;
    lam[0] := -r1;  lam[1] := 1;  t1 := d1/(1 + r1↑2);
    for L := 2 step 1 until nu do
    begin
      lam[L] := ((2×L-1)×omega×lam[L-1]+
        ((L-1)↑2+eta2)×lam[L-2]/(L-1))/L;
      lambda[L] := lmin[L] + t1 × (lam[L]+r1×lmin[L])
    end
  end;
  go to L3;
L2: for L := nu1 step 1 until nu - 1 do
      lambda[L+1] := ((2×L+1)×omega×lambda[L]+
        (L↑2+eta2)×lambda[L-1]/L)/(L+1);
L3: r := s := 0;
  for L := nu step -1 until 1 do
  begin
    t1 := eta/(L+1);
    r := 1/((2×L-1)×(t1/L+ro1-(1+t1↑2)×r/(2×L+3)));
    s := r × (lambda[L]+s);
    if L ≦ Lmax then Rr[L-1] := r
  end;
  F[0] := sum/(1+s);
  for L := 1 step 1 until Lmax do F[L] := Rr[L-1] × F[L-1];
  comment The for-statement which follows is of purely
    precautionary nature, making sure that the results have the
    required accuracy. If speed is important, the statement
    may be omitted;
```

```
    for L := 0 step 1 until Lmax do
    if abs(F[L]-Fapprox[L]) > epsilon × abs(F[L]) then
    begin
      for k := 0 step 1 until Lmax do Fapprox[k] := F[k];
      nu1 := mu1 := nu;  nu := nu + 10;
      if nu < 300 then go to L0 else
      begin
        outstring (1, 'convergence difficulty in Coulomb');
        go to L5
      end
    end
  end;
  t1 := 6.2831853072 × eta;
  comment The constant 2π in the preceding statement must be
    supplied more accurately if more than 11 significant digits are
    desired in the final results;
  if abs(t1) < 1 then
  begin
    t2 := s := 1;  L := 1;
L4: L := L + 1;
    t2 := t1 × t2/L;  s := s + t2;
    if abs(t2) > epsilon × abs(s) then go to L4;
    s := sqrt(1/s)
  end
  else
    s := sqrt(t1/(exp(t1)-1));
  F[0] := s × F[0];
  for L := 1 step 1 until Lmax do
  begin
    s := (L-.5) × sqrt(L↑2+eta2) × s/(L×(L+.5));
    F[L] := s × F[L]
  end;
L5: end Coulomb;
```

comment The procedure *Coulomb* was tested on the CDC 3600 computer, with the procedure *minimal* in single precision (unless stated otherwise). The tests included the following:

(i) Generation of $\Phi_L(\eta, \rho) = [C_L(\eta)\rho^{L+1}]^{-1}F_L(\eta, \rho)$, $L = 0(1)21$, to 8 significant digits ($d=8$) for $\eta = 0, -5(2)5$, $\rho = .2, 1(1)5$. The results were in complete agreement with values tabulated in [4].

(ii) Computation of $F_0(\eta, \rho)$, $F_0'(\eta, \rho) = (d/d\rho)F_0(\eta, \rho)$ to 6 significant digits for $\eta = 0(2)12$, $\rho = 0(5)40$, using $F_0' = (\rho^{-1}+\eta)F_0 - (1+\eta^2)^{\frac{1}{2}}F_1$. Comparison with [5] revealed frequent discrepancies of one unit in the last digit. In addition, beginning with $\eta = 8$, the results became progressively worse for $\rho = 30, 35, 40$, being correct to only 2–3 digits when $\eta = 12$, $\rho = 40$. With the procedure *minimal* in double precision, however, these errors disappeared.

(iii) Computation to 8 significant digits of $F_0(\eta, \rho)$, $F_0'(\eta, \rho)$ for $\rho = 2\eta$, $\rho = .5(.5)20(2)50$. The results agreed with those published in [1] for $\rho \leqq 16$, but became increasingly inaccurate for larger values of ρ. Complete agreement was observed, however, when the procedure *minimal* was operating in the double-precision mode;

REFERENCES:

1. ABRAMOWITZ, M., AND RABINOWITZ, P. Evaluation of Coulomb wave functions along the transition line. *Phys. Rev. 96* (1954), 77–79.
2. ABRAMOWITZ, M., AND STEGUN, I. A. (Eds.). *Handbook of Mathematical Functions*. NBS Appl. Math. Ser. 55, U. S. Gov't. Printing Off., Washington, D. C., 1964.
3. GAUTSCHI, W. Computational aspects of three-term recurrence relations. *SIAM Rev.*, to appear.
4. NATIONAL BUREAU OF STANDARDS. *Tables of Coulomb Wave Functions, Vol. I*. Appl. Math. Ser. 17, U. S. Gov't. Printing Office, Washington, D. C., 1952.
5. TUBIS, A. Tables of nonrelativistic Coulomb wave functions. LA-2150, Los Alamos Scientific Lab., Los Alamos, New Mexico, 1958.

CERTIFICATION OF ALGORITHM 257 [D1] HAVIE INTEGRATOR [Robert N. Kubik, *Comm. ACM 8* (June 1965), 381]

KENNETH HILLSTROM (Recd. 28 Feb. 1966, 29 Apr. 1966 and 15 July 1966)

Applied Mathematics Division, Argonne National Laboratory, Argonne, Illinois

Work performed under the auspices of the U.S. Atomic Energy Commission.

Havie Integrator was coded in CDC 3600 FORTRAN. This routine and a FORTRAN-coded Romberg integration routine based upon Algorithm 60, Romberg Integration [*Comm. ACM 4* (June 1961), 255] were tested with five and four integrands, respectively.

The results of these tests are tabulated below. (The ALGOL-coded Havie routine was transcribed and tested for the two integrands used by Kubik, with identical results in both cases.)

In the following table, A is the lower limit of the interval of integration, B is the upper limit, EPS the convergence criterion, VI the value of the integral and VA the value of the approximation.

Integrand	A	B	EPS	VI	*Routine*	VA	*Number of Function Evaluations*
$\cos x$	0	$\pi/2$	10^{-6}	1.0	Havie	0.9999999981	17
					Romberg	1.000000000	17
e^{-x^2}	0	4.3	10^{-6}	0.886226924	Havie	0.886226924	17
					Romberg	0.886336925	65
$\ln x$	1	10	10^{-6}	14.0258509	Havie	14.02585084	65
					Romberg	14.02585085	65
$\left(\frac{(x)^{1/2}}{e^{x-4}+1}\right)$	0	20	10^{-6}	5.7707276	Havie	5.770724810	32,769
					Romberg	5.770724810	16,385
$\cos(4x)$	0	π	10^{-6}	0.0	Havie	3.1415926536	3[a]

[a] Since in the Havie procedure, the sample points of the interval, chosen for function evaluation, are determined by halving the interval and are, therefore, function-independent, there are functions for which the convergence criterion is satisfied before the requisite accuracy is obtained. An example is the integrand $f(x) = \cos(4x)$ integrated over the interval $[0, \pi]$. The value obtained from the routine is $= \pi$. The true value of the integral is 0.

This inherent limitation applies to all integration algorithms that obtain sample points in a fixed manner.

REMARK ON ALGORITHM 286 [H] EXAMINATION SCHEDULING [J. E. L. Peck and M. R. Williams, *Comm. ACM 9* (June 1966), 433].

The 6th and 7th lines from the end of the procedure should be corrected by the insertion of a **begin end** pair so that they read

```
if row [i] < 0 then
  begin outinteger (1, i); outinteger (1, row [i]); outinteger
    (1, w[i])
  end
```

1966 Algorithms Index will appear in the December issue of *Communications*.