8. Conclusion

Because the output of one pass is read backwards by the next pass, this method is particularly suited to machines with tapes that can be read in either direction. Disk files or drums are also suitable for the intermediate storage, and can lead to very fast second and third passes.

On a machine with a large main memory many methods of optimization which use large parts of core are feasible. Since the first pass is usually highly input limited, it might be an advantage to use the method outlined above on several programs simultaneously; that is, the compiler program is time-shared by several inputs and auxiliary devices, each using a small piece of memory for data storage.

Partial optimization of the object code in a machine independent fashion is certainly feasible using a small amount of main memory with a reversible auxiliary store. On most machines it is likely that the input time for the source program will dominate the compile speed. Optimization of the use of machine features such as index registers is a harder problem, probably requiring two further passes, one in each direction, since in the proposed compiler, information about which addresses would best be in index registers is not available until pass II, and pass II must generate code whose length can be determined by the end of pass II.

RECEIVED MARCH, 1965

REFERENCES

1. FLOYD, R. W. An algorithm for coding efficient arithmetic operations. *Comm. ACM 4* (Jan. 1961), 42.
2. GEAR, C. W. Optimization of the address field compilation in the ILLIAC II assembler. *Comput. J. 6* (Jan. 1964), 332.
3. HUXTABLE, D. H. R. On Writing an Optimizing Translator for ALGOL 60. In *Introduction to System Programming*, P. Wegner (Ed.), Academic Press, 1964, 137.
4. IBM. Systems Manual for 704 FORTRAN and 709 FORTRAN Appl. Programming Dept., IBM, April, 1960.
5. SAMELSON, K., AND BAUER, F. L. Sequential formula translation. *Comm. ACM 3* (Feb. 1960), 76.

Algorithms

J. G. HERRIOT, Editor

ALGORITHM 259
LEGENDRE FUNCTIONS FOR ARGUMENTS LARGER THAN ONE* [S16]

WALTER GAUTSCHI (Recd. 5 Mar. 1965)
Purdue University, Lafayette, Ind. and Argonne National Laboratory, Argonne, Ill.

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

begin
comment Control is transferred to a nonlocal label, called *alarm*, whenever the input variables are not in the intended range;
procedure *integer Legendre* 1 $(x, a, nmax, P)$;
value x, a, $nmax$; **integer** a, $nmax$; **real** x; **array** P;
comment This procedure generates the associated Legendre functions of the first kind,

$$P_a{}^n(x) = \frac{(x^2-1)^{n/2}}{2^a\, a!}\frac{d^{a+n}}{dx^{a+n}}(x^2-1)^a,$$

for $n = 0(1)nmax$, assuming $a \geq 0$ an integer, and $x > 1$. The results are stored in the array P. The method of computation is derived from the (finite) continued fraction

$$(n+a)F_n/F_{n-1} = \frac{(n+a)\,(a+1-n)}{nx_1+}\,\frac{(n+a+1)\,(a-n)}{(n+1)x_1+}\,\frac{(n+a+2)\,(a-n-1)}{(n+2)x_1+}\cdots\frac{2a\cdot 1}{ax_1}\quad(1 \leqq n \leqq a),$$

where $F_n = P_a{}^n(x)/(n+a)!$, $x_1 = 2x(x^2-1)^{-\frac{1}{2}}$, and the identity

$$F_0 + 2\sum_{n=1}^{a} F_n = [x+(x^2-1)^{\frac{1}{2}}]^a/a!.$$

If x is very close to 1, the computation of x_1 is subject to cancellation of significant digits. In such cases it would be better to use $y = x-1$ as input variable, and to compute $(x^2-1)^{\frac{1}{2}}$ by $[y(2+y)]^{\frac{1}{2}}$ everywhere in the procedure body;

```
begin integer n; real x1, c, sum, r, s;
  array Rr[0:nmax-1];
  if x < 1 ∨ a < 0 ∨ nmax < 0 then go to alarm;
  if x = 1 ∨ a = 0 then
  begin
    P[0] := 1; for n := 1 step 1 until nmax do P[n] := 0;
    go to L
  end;
  for n := a+1 step 1 until nmax do P[n] := 0;
  x1 := sqrt (x↑2-1);
  c := 1; for n := 2 step 1 until a do c := n × c;
  sum := (x+x1)↑a/c; x1 := 2 × x/x1;
  r := s := 0;
  for n := a step - 1 until 1 do
  begin
    r := (a+1-n)/(n×x1+(n+a+1)×r); s := r × (2+s);
    if n ≤ nmax then Rr[n-1] := r
  end;
  P[0] := c × sum/(1+s);
```

```
  for n := 0 step 1 until if nmax ≤ a then nmax−1 else a−1 do
    P[n+1] := (n+a+1) × Rr[n] × P[n];
L: end integer Legendre 1;
procedure integer Legendre 2(x, m, nmax, d, Q);
  value x, m, nmax, d;  integer m, nmax, d;  real x; array Q;
```

comment This procedure generates to d significant digits the associated Legendre functions of the second kind, $Q_n^m(x)$, for $n = 0(1)nmax$, assuming $m \geq 0$ an integer, and $x > 1$. The results are stored in the array Q. The procedure first generates $Q_0^m(x)$ from the recurrence relation

$$Q_n^{r+1} + \frac{2rx}{(x^2-1)^{\frac{1}{2}}} Q_n^r + (r+n)(r-n-1)Q_n^{r-1} = 0 \qquad (r = 1, 2, \cdots, m-1) \tag{1}$$

with $n = 0$, and the initial values

$$Q_0^0(x) = \frac{1}{2} \ln \frac{x+1}{x-1}, \qquad Q_0^1(x) = -(x^2-1)^{-\frac{1}{2}}.$$

Then a variant of the backward recurrence algorithm of J. C. P. Miller is applied to the recursion

$$(n-m+1)Q_{n+1}^m - (2n+1)xQ_n^m + (n+m)Q_{n-1}^m = 0 \qquad (n=1, 2, 3, \cdots). \tag{2}$$

(For more details see [2]. See also [4] for a very similar algorithm.) If $m > 1$, the leading coefficient in (2) vanishes for $n = m - 1$, which invalidates the theoretical justification for the backward recurrence procedure. Nevertheless, it appears that the procedure produces valid results for arbitrary $m \geq 0$. Convergence of the backward recurrence algorithm is slow for x near 1, but improves rapidly as x increases;

```
begin integer n, nu, p;  real x1, Q0, Q1, Q2, epsilon, r;
  array Qapprox, Rr[0: nmax];
  if x ≤ 1 ∨ nmax < 0 ∨ m < 0 then go to alarm;
  x1 := sqrt (x↑2−1);
  Q1 := .5 × ln ((x+1)/(x−1));
  if m = 0 then Q[0] := Q1 else
  begin
    Q2 := −1/x1;  x1 := 2 × x/x1;
    for n := 1 step 1 until m − 1 do
    begin
      Q0 := Q1;  Q1 := Q2;
        Q2 := −n × x1 × Q1 − n × (n−1) × Q0
    end;
    Q[0] := Q2
  end;
  for n := 0 step 1 until nmax do Qapprox[n] := 0;
  epsilon := .5 × 10↑(−d);
  nu := 20 + entier (1.25 × nmax);
L0: r := 0;
  for n := nu step − 1 until 1 do
  begin
    r := (n+m)/((2×n+1)×x−(n−m+1)×r);
    if n ≤ nmax then Rr[n−1] := r
  end;
  for n := 0 step 1 until nmax−1 do Q[n+1] := Rr[n] × Q[n];
  for n := 0 step 1 until nmax do
  if abs(Q[n]−Qapprox[n]) > epsilon × abs(Q[n]) then
  begin
    for p := 0 step 1 until nmax do Qapprox[p] := Q[p];
    nu := nu + 10;  go to L0
  end
end integer Legendre 2;
procedure integer Legendre 3(x, n, mmax, d, Q);
  value x, n, mmax, d;  integer n, mmax, d;  real x;  array Q;
```

comment This procedure generates to d significant digits, and stores in the array Q, the Legendre functions of the second kind, $Q_n^m(x)$, for $m = 0(1)mmax$, assuming $n \geq 0$ an integer, and $x > 1$. The procedure *integer Legendre* 2 is used to obtain initial values Q_n^0, Q_n^1, and subsequent values are obtained from the recursion (1) of the preceding comment;

```
begin integer m;  real x1;  array Q1[0:n];
  if n < 0 ∨ mmax < 0 then go to alarm;
  integer Legendre 2(x, 0, n, d, Q1);  Q[0] := Q1[n];
  x1 := 2 × x/sqrt(x↑2−1);
  if mmax > 0 then
  begin
    integer Legendre 2(x, 1, n, d, Q1);  Q[1] := Q1[n]
  end;
  for m := 1 step 1 until mmax−1 do
    Q[m+1] := −m × x1 × Q[m] − (m+n) × (m−n−1) × Q[m−1]
end integer Legendre 3;
procedure Legendre 1(x, alpha, nmax, d, P1);
  value x, alpha, nmax, d;  integer nmax, d;
  real x, alpha;  array P1;
```

comment This procedure evaluates to d significant digits the Legendre functions

$$P_\alpha^n(x) = \frac{\Gamma(\alpha+n+1)}{\pi\Gamma(\alpha+1)} \int_0^\pi [x + (x^2-1)^{\frac{1}{2}} \cos t]^\alpha \cos nt \, dt$$

for $n = 0(1)nmax$, where $x > 1$ and α is real. The results are stored in the array $P1$. It is assumed that a nonlocal procedure *gamma* be available which evaluates $\Gamma(z)$ for $0 < z \leq 2$. (See [3].) The procedure first generates the quantities $f_n = P_\alpha^n(x)/\Gamma(\alpha+n+1)$ from the recurrence relation

$$f_{n+1} + \frac{2nx}{(n+\alpha+1)(x^2-1)^{\frac{1}{2}}} f_n + \frac{n-\alpha-1}{n+\alpha+1} f_{n-1} = 0,$$

and the identity

$$f_0 + 2\sum_{n=1}^{\infty} f_n = \frac{[x + (x^2-1)^{\frac{1}{2}}]^\alpha}{\Gamma(\alpha+1)},$$

applying a variant of the backward recurrence algorithm of J. C. P. Miller. (See [2] for more details.) Then $P_\alpha^n(x) = \Gamma(\alpha+n+1)f_n$ is obtained recursively. If $\alpha < -\frac{1}{2}$, we let $a = -\alpha-1$ and compute $P_\alpha^n(x) = P_a^n(x)$. The substitution is made to avoid loss of accuracy when x is large. The rate of convergence of this procedure decreases as x increases. A general idea of the speed of convergence may be obtained from the graphs in [2, §6]. If x is very close to 1, the same changes as mentioned in the first procedure are recommended;

```
begin integer n, nu, m;  real a, epsilon, x1, sum, c, r, s;
  array Papprox, Rr[0:nmax];
  if x < 1 ∨ nmax < 0 ∨ entier(alpha) − alpha = 0 then
  go to alarm;  if x = 1 then
  begin
    P1[0] := 1;  for n := 1 step 1 until nmax do P1[n] := 0;
    go to L1
  end;
  a := if alpha < −.5 then − alpha − 1 else alpha;
  for n := 0 step 1 until nmax do Papprox[n] := 0;
  epsilon := .5 × 10↑(−d);
  if a ≤ 1 then c := gamma(1+a) else
  begin
    m := entier(a) − 1;  c := gamma(a−m);
    for n := 0 step 1 until m do c := (a−n) × c
  end;
  x1 := sqrt (x↑2−1);  sum := (x+x1)↑a/c;  x1 := 2 × x/x1;
 nu := 20 + entier ((37.26+.1283×(a+38.26)×x)×nmax/
    (37.26+.1283×(a+1)×x));
L0: r := s := 0;
  for n := nu step − 1 until 1 do
  begin
    r := (a+1−n)/(n×x1+(n+a+1)×r);  s := r × (2+s);
    if n ≤ nmax then Rr[n−1] := r
  end;
```

```
  P1[0] := sum/(1+s);
  for n := 0 step 1 until nmax - 1 do
    P1[n+1] := Rr[n] × P1[n];
  for n := 0 step 1 until nmax do
    if abs (P1[n]-Papprox[n]) > epsilon × abs (P1[n]) then
    begin
      for m := 0 step 1 until nmax do Papprox [m] := P1[m];
      nu := nu + 10; go to L0
    end;
  P1[0] := c × P1[0];
  for n := 1 step 1 until nmax do
  begin
    c := (a+n) × c;  P1[n] := c × P1[n]
  end;
L1:  end Legendre 1;
```

```
procedure Legendre 2(x, a, m, nmax, d, P2);
  value x, a, m, nmax, d;  integer m, nmax, d;  real x, a;
  array P2;
comment This procedure evaluates to d significant digits the
  Legendre functions P^m_{a+n}(x) for fixed x > 1, a, m ≥ 0, and for
  n = 0(1)nmax. The results are stored in the array P2. They are
  obtained recursively from
```

$$P^m_{a+n+1}(x) = \frac{2n+2a+1}{n+a-m+1}\,xP^m_{a+n}(x) - \frac{n+a+m}{n+a-m+1}\,P^m_{a+n-1}(x),$$

```
  the initial values being calculated with the help of the proce-
  dure Legendre 1;
begin integer n;  array P1[0: m];
  if m < 0 then go to alarm;
  Legendre 1(x, a, m, d, P1);  P2[0] := P1[m];
  if nmax > 0 then
  begin
  Legendre 1(x, a+1, m, d, P1);  P2[1] := P1[m]
  end;
  for n := 1 step 1 until nmax-1 do
    P2[n+1] := ((2×n+2×a+1)×x×P2[n]
      - (n+a+m)×P2[n-1])/(n+a-m+1)
end Legendre 2;
```

```
procedure conical (x, tau, nmax, d, P);
  value x, tau, nmax, d;  integer nmax, d;  real x, tau;  array P;
comment This is an adaption of the procedure Legendre 1 to the
  case α = -1/2 + iτ, where τ is real. The procedure thus generates
  Mehler's conical functions P^n_{-1/2+iτ}(x) to d significant digits for
  n = 0(1)nmax and x > 1. The results are stored in the array P.
  To avoid excessively large and excessively small numbers, we
  let f_n = P^n_{-1/2+iτ}(x)/n! and first compute f_n from the recurrence
  relation
```

$$f_{n+1} + \frac{2nx}{(n+1)\,(x^2-1)^{\frac{1}{2}}}f_n + \frac{(n-\frac{1}{2})^2+\tau^2}{n(n+1)}f_{n-1} = 0,$$

and the identity

$$f_0 + \sum_{n=1}^{\infty}\lambda_n f_n = [x+(x^2-1)^{\frac{1}{2}}]^{-\frac{1}{2}} \cos\,(\tau \ln\,[x+(x^2-1)^{\frac{1}{2}}]),$$

where

$$\lambda_n = n!\left[\frac{\Gamma(\frac{1}{2}+i\tau)}{\Gamma(\frac{1}{2}+i\tau+n)} + \frac{\Gamma(\frac{1}{2}-i\tau)}{\Gamma(\frac{1}{2}-i\tau+n)}\right].$$

The λ's are obtained recursively by

$$\lambda_1 = \frac{1}{\frac{1}{4}+\tau^2}, \qquad \lambda_2 = \frac{3-4\tau^2}{(\frac{1}{4}+\tau^2)\,(\frac{9}{4}+\tau^2)},$$

$$\lambda_{n+1} = \frac{1+\dfrac{1}{n}}{\left(1+\dfrac{1}{2n}\right)^2+\left(\dfrac{\tau}{n}\right)^2}\,(2\lambda_n - \lambda_{n-1}) \qquad (n = 2,3,\cdots).$$

```
  The procedure converges rather slowly if x and τ are both large
  (see the graphs in §6 of [2]). If the accuracy requirement as
  specified by d is too stringent the procedure may not converge
  at all due to the accumulation of rounding errors;
begin integer n, nu, m;  real epsilon, t, x1, x2, sum, lambda 1,
  lambda 2, lambda, r, s;  array Papprox, Rr[0:nmax];
  if x < 1 ∨ nmax < 0 then go to alarm;
  if x = 1 then
  begin
    P[0] := 1;  for n := 1 step 1 until nmax do P[n] := 0;
  go to L3
  end;
  t := tau↑2;
  for n := 0 step 1 until nmax do Papprox[n] := 0;
  epsilon := .5 × 10↑(-d);
  x1 := sqrt(x↑2-1);  x2 := x + x1;
  sum := cos(tau×ln(x2))/sqrt(x2);  x1 := 2 × x/x1;
  nu := 30 + entier ((1+(.140+.0246×tau) × (x-1))×nmax);
L0:  n := 2;
  lambda 1 := 1/(.25+t);
  lambda 2 := (3-4×t)/((.25+t)×(2.25+t));
L1:  lambda := (1+1/n) × (2×lambda 2-lambda 1)/
  ((1+.5/n)↑2 + (tau/n)↑2);
  if n < nu then
  begin
  lambda 1 := lambda 2;  lambda 2 := lambda;
    n := n + 1;  go to L1
  end;
  r := s := 0;
L2:  r := -((1-.5/n)↑2+(tau/n)↑2)/(x1+(1+1/n)×r);
  s := r × (lambda 2+s);
  if n ≤ nmax then Rr[n-1] := r;
  lambda 1 := lambda 2;
  lambda 2 := 2 × lambda 2 - ((1+.5/n)↑2+(tau/n)↑2)
    × lambda/(1+1/n);
  lambda := lambda 1;
  n := n - 1;  if n ≥ 1 then go to L2;
  P[0] := sum/(1+s);
  for n := 0 step 1 until nmax - 1 do P[n+1] := Rr[n] × P[n];
  for n := 0 step 1 until nmax do
    if abs (P[n]-Papprox[n]) > epsilon × abs(P[n]) then
    begin
      for m := 0 step 1 until nmax do Papprox[m] := P[m];
      nu := nu + 60;  comment To avoid an infinite loop in
        case of divergence the user should provide for an upper
        bound on nu, say 1000, and exit from the procedure when
        nu exceeds this bound, printing an appropriate error
        message;
      go to L0
    end;
  t := 1;
  for n := 1 step 1 until nmax do
  begin
    t := n × t;  P[n] := t × P[n]
  end;
L3:  end conical;
```

```
procedure toroidal (x, m, nmax, d, Q);
  value x, m, nmax, d;  integer m, nmax, d;  real x;  array Q;
comment This procedure generates to d significant digits the
  toroidal functions of the second kind, Q^m_{-1/2+n}(x), for n = 0(1)
  nmax, where x > 1, and m is an integer, positive, negative or
  zero. The method of computation is based on the recurrence
  relation
```

$$(n-m+\tfrac{1}{2})Q^m_{-\frac{1}{2}+n+1}(x) - 2nxQ^m_{-\frac{1}{2}+n}(x) + (n+m-\tfrac{1}{2})Q^m_{-\frac{1}{2}+n-1}(x) = 0,$$

and the identity

$$Q^m_{-\frac{1}{2}}(x) + 2\sum_{n=1}^{\infty} Q^m_{-\frac{1}{2}+n}(x) = (-1)^m\sqrt{\frac{\pi}{2}}\,\Gamma(m+\tfrac{1}{2})(x-1)^{-\frac{1}{2}}\left(\frac{x+1}{x-1}\right)^{m/2},$$

to which a variant of J. C. P. Miller's backward recurrence algorithm is applied. (See [2] for more details.) The convergence of this procedure is slow for x near 1, and improves rapidly as x increases;

```
begin integer n, nu, p;  real epsilon, x1, c, sum, r, s;
  array Qapprox, Rr[0:nmax];
  if x ≤ 1 ∨ nmax < 0 then go to alarm;
  for n := 0 step 1 until nmax do Qapprox[n] := 0;
  epsilon := .5 × 10↑(−d);
  c := 2.2214414691;
  if m ≥ 0 then
    for n := 0 step 1 until m−1 do c := −(n+.5) × c
  else
    for n := 0 step −1 until m+1 do c := −c/(n−.5);
  sum := c × ((x+1)/(x−1))↑(m/2)/sqrt(x−1);  x1 := 2 × x;
  nu := 20 + entier ((1.15+(.0146+.00122×m)/(x−1))×nmax);
L0:  r := s := 0;
  for n := nu step −1 until 1 do
  begin
    r := (n+m−.5)/(n×x1−(n−m+.5)×r);  s := r × (2+s);
    if n ≤ nmax then Rr[n−1] := r
  end;
  Q[0] := sum/(1+s);
  for n := 0 step 1 until nmax − 1 do Q[n+1] := Rr[n] × Q[n];
  for n := 0 step 1 until nmax do
    if abs(Q[n]−Qapprox[n]) > epsilon × abs(Q[n]) then
    begin
      for p := 0 step 1 until nmax do Qapprox[p] := Q[p];
      nu := nu + 10;  go to L0
    end
end toroidal;
```

comment All procedures were tested on the CDC 3600 computer. Some of the tests that were run are described below;

comment The procedures *integer Legendre* 1–3 were driven to print test values to 6 significant digits of $P_n^m(x)$, $Q_m^n(x)$, $Q_n^m(x)$, $m = 0(1)10$, for $x = 1.5$, 3, 10, and $n = 0(1)5$. As far as possible, the results were compared with values tabulated in [5], and found to be in complete agreement. Similarly, test values of $P^m_{-\frac12+n}(x)$, $m = 0(1)4$, were obtained from the procedure *Legendre* 1, for $x = 1.5$, 3, 10, and $n = 0(1)5$. All agreed with values tabulated in [5]. More extensive tests could be run by having the procedure "verify" the addition theorem

$$P_\alpha(xy-\sqrt{(x^2-1)}\sqrt{(y^2-1)}) = P_\alpha(x)P_\alpha(y) + 2\sum_{m=1}^{\infty}(-1)^m\frac{\Gamma(\alpha-m+1)}{\Gamma(\alpha+m+1)}P_\alpha{}^m(x)P_\alpha{}^m(y), \quad x>1, y>1;$$

comment The procedure *conical* (with d=6) was run to produce test values of $P^m_{-\frac12+i\tau}(x)$, $m = 0$, 1, for $x = 1.5$, 5, 10, 20, and $\tau = 0(10)30$. The results agreed to 6 significant digits with those in [10], [11];

comment The procedure *toroidal* was driven to generate test values to 6 significant digits of $Q^m_{-\frac12+n}(x)$, $Q^{-m}_{-\frac12+n}(x)$, $n = 0(1)5$, for $x = 1.5$, 3, 10, and $m = 0(1)4$. All values of $Q^m_{-\frac12+n}(x)$ were checked against those in [5]. There were no discrepancies. The values of $Q^{-m}_{-\frac12+n}(x)$ were compared with those of $[\Gamma(n-m+\frac12)/\Gamma(n+m+\frac12)]Q^m_{-\frac12+n}(x)$. The largest relative error observed was $1.5_{10}-9$, occurring at $m = 4$, $n = 5$, $x = 1.5$;

comment Integrals of the form

$$f_n(k^2,\alpha) = (-1)^n\int_0^{\pi/2}[1-k^2\sin^2\psi]^\alpha\cos 2n\psi\,d\psi, \qquad 0<k<1,$$

are repeatedly encountered in applied mathematics (see, e.g., [6]–[9], where $\alpha = -\frac32$, or $\alpha = -\frac52$). It is readily seen that

$$f_n(k^2,\alpha) = (-1)^n\frac{\pi}{2}\frac{\Gamma(\alpha+1)}{\Gamma(\alpha+n+1)}(1-k^2)^{\alpha/2}P_\alpha{}^n\left(\frac{2-k^2}{2\sqrt{(1-k^2)}}\right).$$

The program that follows generates $(1-k^2)f_n(k^2,\alpha)$, $n = 0(1)10$, for $\alpha = -\frac32, -\frac52$, and $k^2 = .1$, .5, .9, calling for an accuracy of 6 significant digits. Selected results are shown below.

α	k^2	n	$(1-k^2)f_n(k^2,\alpha)$
−1.5	.1	0	1.5307576371
		5	$5.2456440472_{10}-8$
		10	$9.0801648667_{10}-16$
	.5	1	$3.4378228849_{10}-1$
		4	$2.8295844423_{10}-3$
		7	$1.8215954880_{10}-5$
	.9	2	$4.8615561237_{10}-1$
		6	$5.2878408708_{10}-2$
		9	$8.8107743954_{10}-3$
−2.5	.1	0	1.6169191877
		5	$2.3969022984_{10}-7$
		10	$7.3394117106_{10}-15$
	.5	1	$8.4721308463_{10}-1$
		4	$1.4940149605_{10}-2$
		7	$1.4764302684_{10}-4$
	.9	2	4.9389962376
		6	$9.7073200383_{10}-1$
		9	$2.1695170317_{10}-1$

Those for $\alpha = -\frac32$ were compared with values tabulated in [6]. There was agreement in all four decimal places given;

```
begin integer n;  real alpha, k2, c;  array P1[0:10];
  for alpha := −1.5, −2.5 do
  for k2 := .1, .5, .9 do
    begin
      c := 1.570796327 × (1−k2)↑(1+alpha/2);
      Legendre 1 (.5×(2−k2)/sqrt(1−k2), alpha, 10, 6, P1);
      for n := 0 step 1 until 10 do
      begin
        P1[n] := c × P1[n];  c := −c/(n+alpha+1);
        outreal (1, P1[n])
      end
    end;
  go to skip;
alarm:  outstring (1, 'parameters not in range');
skip: end;
```

comment The integrals

$$\Omega_j(k) = \int_0^\pi (1-k^2\cos\phi)^{-\frac12-j}\,d\phi, \qquad 0 \leqq k < 1, j = 0, 1, 2, \cdots$$

arose in recent radiation field studies ([1]). One has

$$\Omega_j(k) = \pi(1-k^4)^{-(1+j)/2}P_{-\frac12+j}((1-k^4)^{-\frac12}).$$

The program below calculates $\Omega_j(k)$ to 8 significant digits for $k^2 = .2(.2).8$, $j = 0(1)9$. The results agree to 8 figures with the values tabulated in [1];

```
begin integer j;  real k2, x, x1;  array P2, omega [0:9];
  for k2 := .2 step .2 until .9 do
  begin
    x := 1/sqrt(1−k2↑2);
    Legendre 2(x, −.5, 0, 9, 8, P2);
    x1 := 3.1415926536 × sqrt(x);
    omega [0] := x1 × P2[0];
    for j := 1 step 1 until 9 do
    begin
      x1 := x × x1; omega[j] := x1 × P2[j]
    end;
    for j := 0 step 1 until 9 do outreal (1, omega[j])
  end;
  go to skip;
alarm: outstring (1, 'parameters not in range');
skip: end
end
```


REFERENCES:

1. EPSTEIN, L. F., AND HUBBELL, J. H. Evaluation of a generalized elliptic-type integral. *J. Research NBS 67B* (1963), 1–17.

2. Gautschi, W. Computational aspects of three-term recurrence relations. Unpublished.
3. ——. Algorithm 221—Gamma function. *Comm. ACM 7* (Mar. 1964), 143.
4. Herndon, J. R. Algorithm 62—A set of associate Legendre polynomials of the second kind. *Comm. ACM 4* (July 1961), 320–321; Remark on Algorithm 62. *Comm. ACM 4* (Dec. 1961), 544.
5. *NBS Tables of Associated Legendre Functions*. Columbia University Press, New York, 1945.
6. Riegels, F. Formeln und Tabellen für ein in der räumlichen Potentialtheorie auftretendes elliptisches Integral. *Archiv der Mathematik 2* (1949/50), 117–125.
7. Siekmann, J. Concerning an integral occurring in airfoil theory. *SIAM Review 3* (1961), 243–246.
8. ——. Analysis of ring aerofoils of elliptic cross section, Part I: General theory. *J. SIAM 11* (1963), 941–963.
9. ——. Note on a Riegels-type integral. *Z. Angew. Math. Phys. 15* (1964), 79–83.
10. Žurina, M. I., and Karmazina, L. N. Tablicy funkciĭ Ležandra $P_{-\frac{1}{2}+i\tau}(x)$, Vol. I. Akad. Nauk SSSR, Moscow, 1962.
11. ——, and ——. Tablicy funkciĭ Ležandra $P^1_{-\frac{1}{2}+i\tau}(x)$. Akad. Nauk SSSR, Moscow, 1963.

ALGORITHM 260
6-J SYMBOLS [Z]

J. H. Gunn (Recd. 13 Nov. 1964)
Nordisk Institut for Teoretisk Atomfysik, Copenhagen, Denmark


```
real procedure SJS (J1, J2, J3, L1, L2, L3, factorial);
  value J1, J2, J3, L1, L2, L3;
  integer J1, J2, J3, L1, L2, L3;
  array factorial;
comment SJS calculates the 6-j symbols defined by the following formula
```

$$\begin{Bmatrix} j1 & j2 & j3 \\ l1 & l2 & l3 \end{Bmatrix} = \Delta(j1, j2, j3)\Delta(j1, l2, l3)\Delta(l1, j2, l3)\Delta(l1, l2, j3) \times \sum_z (-1)^z (z+1)!/((z-j1-j2-j3)!(z-j1-l2-l3)!(z-l1-j2-l3)!(z-l1-l2-j3)!(j1+j2+l1+l2-z)!(j2+j3+l2+l3-z)!(j3+j1+l3+l1-z)!)$$

where

$$\Delta(a, b, c) = \left[\frac{(a+b-c)!\,(a-b+c)!\,(-a+b+c)!}{(a+b+c+1)!}\right]^{\frac{1}{2}}$$

and where $j1 = J1/2$, $j2 = J2/2$, $j3 = J3/2$, $l1 = L1/2$, $l2 = L2/2$' $l3 = L3/2$. [Reference formula 6.3.7 page 99 of Edmonds, A. R. Angular momentum in quantum mechanics. In *Investigations in Physics, 4*, Princeton U. Press, 1957]. The parameters of the procedure $J1$, $J2$, $J3$, $L1$, $L2$, $L3$ are interpreted as being twice their physical value, so that actual parameters may be inserted as integers. Thus to calculate the 6-j symbol

$$\begin{Bmatrix} 2 & 2 & 0 \\ 2 & 2 & 0 \end{Bmatrix}$$

the call would be *SJS* (4, 4, 0, 4, 4, 0, *factorial*). The procedure checks that the triangle conditions for the existence of a coefficient are satisfied and that $j1 + j2 + j3$, $j1 + l2 + l3$, $l1 + j2 + l3$ and $l1 + l2 + j3$ are integral. If the conditions are not satisfied the value of the procedure is zero. The parameter *factorial* is an array containing the factorials from 0 up to at least 1 + largest of $j1 + j2 + j3$, $j1 + l2 + l3$, $l1 + j2 + l3$ and $l1 + l2 + j3$. Since in actual calculations the procedure *SJS* will be called many times it is more economical to have the factorials in a global array rather than compute them on every entry to the procedure. The notation is consistent with that used in the procedure for calculating Vector-coupling coefficients. See Algorithm 252, Vector Coupling or Clebsch-Gordan Coefficients [*Comm. ACM 8* (Apr. 1965), 217];

```
begin integer w, wmin, wmax;
real omega;
real procedure delta (a, b, c);
  value a, b, c;
  integer a, b, c;
begin delta := sqrt (factorial [(a+b-c)÷2]
  × factorial [(a-b+c)÷2]
  × factorial [(-a+b+c)÷2]/factorial [(a+b+c+2)÷2])
end delta;
if J1 + J2 < J3 ∨ abs(J1 - J2) > J3 ∨ J1 + J2 + J3 ≠
  2 × ((J1+J2+J3)÷2)
∨ J1 + L2 < L3 ∨ abs(J1-L2) > L3 ∨ J1 + L2 + L3 ≠ 2 ×
  ((J1+L2+L3)÷2)
∨ L1 + J2 < L3 ∨ abs(L1-J2) > L3 ∨ L1 + J2 + L3 ≠ 2 ×
  ((L1+J2+L3)÷2)
∨ L1 + L2 < J3 ∨ abs(L1-L2) > J3 ∨ L1 + L2 + J3 ≠ 2 ×
  ((L1+L2+J3)÷2)
then SJS := 0 else
begin
  omega := 0;
  wmin := J1 + J2 + J3;
  if wmin < J1 + L2 + L3 then wmin := J1 + L2 + L3;
  if wmin < L1 + J2 + L3 then wmin := L1 + J2 + L3;
  if wmin < L1 + L2 + J3 then wmin := L1 + L2 + J3;
  wmax := J1 + J2 + L1 + L2;
  if wmax > J2 + J3 + L2 + L3 then wmax := J2 + J3 +
    L2 + L3;
  if wmax > J3 + J1 + L3 + L1 then wmax := J3 + J1 +
    L3 + L1;
  for w := wmin step 2 until wmax do
  omega := omega + (if w=4×(w÷4) then 1 else -1)
    × factorial [w÷2+1]/(factorial [(w-J1-J2-J3)÷2]
    × factorial [(w-J1-L2-L3)÷2]
    × factorial [(w-L1-J2-L3)÷2]
    × factorial [(w-L1-L2-J3)÷2]
    × factorial [(J1+J2+L1+L2-w)÷2]
    × factorial [(J2+J3+L2+L3-w)÷2]
    × factorial [(J3+J1+L3+L1-w)÷2]);
  SJS := delta (J1, J2, J3) × delta (J1, L2, L3)
    × delta (L1, J2, L3) × delta (L1, L2, J3) × omega;
end
end SJS
```

ALGORITHM 261
9-J SYMBOLS [Z]

J. H. Gunn (Recd. 13 Nov. 1964)
Nordisk Institut for Teoretisk Atomfysik, Copenhagen, Denmark


```
real procedure NJS(J11, J12, J13, J21, J22, J23, J31, J32, J33,
    factorial);
  value J11, J12, J13, J21, J22, J23, J31, J32, J33;
  integer J11, J12, J13, J21, J22, J23, J31, J32, J33;
  array factorial;
comment NJS calculates the 9-j symbols defined by the following formula
```

$$\begin{Bmatrix} j11 & j12 & j13 \\ j21 & j22 & j23 \\ j31 & j32 & j33 \end{Bmatrix} = \sum_k (-1)^{2k}(2k+1) \begin{Bmatrix} j11 & j21 & j31 \\ j32 & j33 & k \end{Bmatrix} \begin{Bmatrix} j12 & j22 & j32 \\ j21 & k & j23 \end{Bmatrix} \begin{Bmatrix} j13 & j23 & j33 \\ k & j11 & j12 \end{Bmatrix}.$$

where $j11 = J11/2$, $j12 = J12/2$, $j13 = J13/2$, $j21 = J21/2$,

$j22 = J22/2$, $j23 = J23/2$, $j31 = J31/2$, $j32 = J32/2$, $j33 = J33/2$ [Reference formula 6.4.3 page 101 of EDMONDS, A. R. Angular momentum in quantum mechanics. In *Investigations in Physics*, *4*, Princeton U. Press, 1957]. The parameters of the procedure $J11$, $J12$, $J13$, $J21$, $J22$, $J23$, $J31$, $J32$, $J33$ are interpreted as being twice their physical value, so that actual parameters may be inserted as integers. Thus to calculate the 9-*j* symbol

$$\begin{Bmatrix} 2 & 2 & 0 \\ 2 & 2 & 0 \\ 0 & 0 & 0 \end{Bmatrix}$$

the call would be *NJS* (4, 4, 0, 4, 4, 0, 0, 0, 0, *factorial*). The procedure checks that the triangle conditions for the existence of a coefficient are satisfied and that $j11 + j21 + j31$, $j21 + j22 + j23$, $j31 + j32 + j33$, $j11 + j12 + j13$, $j12 + j22 + j32$, $j13 + j23 + j33$ are integral. If the conditions are not satisfied the value of the procedure is zero. The parameter *factorial* is an array containing the factorials from 0 up to at least $1 +$ largest of $j11 + j21 + j31$, $j21 + j22 + j23$, $j31 + j32 + j33$, $j11 + j12 + j13$, $j12 + j22 + j32$, $j13 + j23 + j33$. The procedure makes use of the procedure *SJS* [Algorithm 260, 6-*j* symbols, *Comm. ACM 8* (Aug. 1965), 492], for calculating 6-*j* symbols;

```
begin integer k, kmin, kmax;
  real NJ;
  if J11 + J21 < J31 ∨ abs(J11−J21) > J31 ∨ J11 + J21 +
    J31 ≠ 2 × ((J11+J21+J31)÷2)
  ∨ J21 + J22 < J23 ∨ abs(J21−J22) > J23 ∨ J21 + J22 +
    J23 ≠ 2 × ((J21+J22+J23)÷2)
  ∨ J31 + J32 < J33 ∨ abs(J31−J32) > J33 ∨ J31 + J32 +
    J33 ≠ 2 × ((J31+J32+J33)÷2)
  ∨ J11 + J12 < J13 ∨ abs(J11−J12) > J13 ∨ J11 + J12 +
    J13 ≠ 2 × ((J11+J12+J13)÷2)
  ∨ J12 + J22 < J32 ∨ abs(J12−J22) > J32 ∨ J12 + J22 +
    J32 ≠ 2 × ((J12+J22+J32)÷2)
  ∨ J13 + J23 < J33 ∨ abs(J13−J23) > J33 ∨ J13 + J23 +
    J33 ≠ 2 × ((J13+J23+J33)÷2)
  then NJS := 0 else
  begin NJ := 0;
    kmin := abs(J21−J32);
    if kmin < abs(J11−J33) then kmin := abs(J11−J33);
    if kmin < abs(J12−J23) then kmin := abs(J12−J23);
      kmax := J21 + J32;
    if kmax > J11 + J33 then kmax := J11 + J33;
    if kmax > J12 + J23 then kmax := J12 + J23;
      for k := kmin step 2 until kmax do
    NJ := NJ + (if k=2×(k÷2) then 1 else −1) × (k+1) ×
      SJS(J11, J21, J31, J32, J33, k, factorial) ×
      SJS(J12, J22, J32, J21, k, J23, factorial) ×
      SJS(J13, J23, J33, k, J11, J12, factorial);
    NJS := NJ
  end
end NJS
```

ALGORITHM 262
NUMBER OF RESTRICTED PARTITIONS OF N [A1]

J. K. S. McKAY (Recd. 7 Dec. 1964 and 9 Mar. 1965)
Computer Unit, University of Edinburgh, Scotland

```
procedure set (p, N); integer N; integer array p;
comment The number of partitions of n with parts less than
  or equal to m is set in p[n, m] for all n, m such that N ≥ n ≥
  m ≥ 0.
```

REFERENCES:

1. GUPTA, H., GWYTHER, C. E., AND MILLER, J. C. P. Tables of partitions. In *Royal Society Mathematical Tables, vol. 4*, Cambridge U. Press, 1958.
2. HARDY, G. H., AND WRIGHT, E. M. *The Theory of Numbers*. Ch. 19, 4th ed., Clarendon Press, Oxford, 1960;

```
begin integer m, n;
  p[0, 0] := 1;
  for n := 1 step 1 until N do
  begin p[n, 0] := 0;
    for m := 1 step 1 until n do
      p[n, m] := p[n, m−1] +
        p[n−m, if n−m<m then n−m else m]
  end
end set
```

ALGORITHM 263
PARTITION GENERATOR [A1]

J. K. S. McKAY (Recd. 7 Dec. 1964 and 9 Mar. 1965)
Computer Unit, University of Edinburgh, Scotland.

```
procedure generate (p, N, position, ptn, length);
  integer array p, ptn; integer N, length, position;
comment The partitions of N may be mapped in their natural
  order, 1 − 1, onto the consecutive integers from 0 to P(N)−1
  where P(N)(=p[N, N]) is the number of unrestricted partitions
  of N. The array p is set by the procedure set [Algorithm 262,
  Number of Restricted Partitions of N, Comm. ACM 8 (Aug.
  1965), 493]. On entry position contains the integer into which
  the partition is mapped. On exit length contains the number of
  parts and ptn[1: length] contains the parts of the partition in
  descending order.
```

REFERENCE:

1. LITTLEWOOD, D. E. *The Theory of Group Characters*. Ch. 5, 2nd ed., Clarendon Press, Oxford, 1958;

```
begin integer m, n, psn;
  n := N; psn := position; length := 0;
A: length := length + 1; m := 1;
B: if p[n, m] < psn then begin m := m + 1; go to B end else
   if p[n, m] > psn then
C: begin
     ptn[length] := m; psn := psn − p[n, m−1]; n := n − m;
     if n ≠ 0 then go to A; go to D
   end
   else m := m + 1; go to C;
D: end generate
```

ALGORITHM 264
MAP OF PARTITIONS INTO INTEGERS [A1]

J. K. S. McKAY (Recd. 7 Dec. 1964 and 9 Mar. 1965)
Computer Unit, University of Edinburgh, Scotland

```
integer procedure place(p, n, ptn); value n;
  integer array p, ptn; integer n;
comment place is the inverse of the procedure generate [Al-
  gorithm 263, Partition Generator, Comm. ACM 8 (Aug. 1965),
  493]. The array p is set by the procedure set [Algorithm 262,
  Number of Restricted Partitions of N, Comm. ACM 8 (Aug.
  1965), 493]. The procedure produces the integer into which
  the partition of n, stored in descending order of parts in ptn[1]
  onwards, is mapped;
begin integer j, d;
  d := 0;
  if n = 0 then go to B;
  j : = 0;
A: j := j + 1; d := p[n, ptn[j]−1] + d; n := n − ptn[j];
  if n ≠ 0 then go to A;
B: place := d
end place
```