(*Description of Algorithm 392 continued from p. 568.*)

The correct values for the apex are

$$X = 1.6144; \quad Y = 1.1580;$$

$$U = 9.2057; \quad V = 4.0312.$$

Using 81 datum points on the initial curve but not applying extrapolation, the computed values were

$$X = 1.5889; \quad Y = 1.1418;$$

$$U = 9.0441; \quad V = 3.7319.$$

Thus extrapolation significantly improved the results.

By plotting the characteristic grid points in the X-Y plane, one sees that the characteristics become more parallel near the apex. Thus the above problem is ill conditioned. If the initial curve is chosen as $Y = 0$, $1 \leq X \leq 2$, the problem becomes so ill conditioned that the method fails for 81 datum points on the initial curve.

Example of use. In the following listing *TEST CH* sets up the initial data and makes the necessary calls to *CHARAC* to solve *Example* (II) for 81 initial datum points. *CH COEF* computes the coefficients $A_1 = 1 - U^2$, $A_2 = -UV$, $A_3 = -UV$, $A_4 = 1 - V^2$, $H_1 = -4U \exp(2X)$, $B_1 = 0$, $B_2 = 1$, $B_3 = -1$, $B_4 = 0$, $H_2 = 0$ as determined from (5).

REFERENCES:

1. FORSYTHE, G. E., AND W. R. WASOW. *Finite-Difference Methods for Partial Differential Equations*. Wiley, New York, 1960, p. 64.
2. BULIRSCH, R., AND J. STOER. Fehlerabschätzungen und Extrapolation mit Rationalen Funktionen bei Verfahren vom Richardson-Typus. *Num. Math. 6* (1964), 413–427.
3. JEFFREY, A., AND T. TANIUTI. *Non-Linear Wave Propagation.* Academic Press, New York, 1964.

ALGORITHM 393
SPECIAL SERIES SUMMATION WITH ARBITRARY PRECISION [C6]

S. KAMAL ABDALI* (Recd. 23 June 1969 and 9 Mar. 1970)
University of Wisconsin, Department of Computer Sciences, Madison, WI 35706

* This work was done while the author was at the University of Montreal, Montreal, Canada.

KEY WORDS AND PHRASES: function evaluation, series summation, approximation
CR CATEGORIES: 5.12, 5.13

```
procedure series (places, terms, base, digit, sgn, numerator, de-
  nominator, num0, denom0); value places, terms, base; integer
  places, terms, base, sgn, num0, denom0; integer array digit;
  integer procedure numerator, denominator;
comment Programs for very precise summation of series are
  conventionally written in machine language and employ multi-
  precision routines to perform arithmetic on especially defined
  multiword registers. The present algorithm requires only integer
  arithmetic and can be implemented in any algebraic language.
  It is applicable to series in which the ratios of successive terms
  can be expressed as quotients of given integers or integer func-
  tions of term positions.
    The sum of a given series is computed to a given number of
  places, places, in a specified base for representation, base. The
  number of terms needed, terms, should be calculated outside the
  procedure. Procedures numerator and denominator are to be
  obtained from the fraction ith term/(i-1)-th term, expressed as
  a ratio of two integer functions of i. (That fraction should prefer-
  ably be reduced to its lowest terms.) num0 and denom0 are the
  integer numerator and denominator of the 0th term. The out-
  puts of the procedure are the sign of the result, sgn, the integer
  part, digit [0], and the digits of the fractional part, digit [1], ··· ,
  digit [places].
    For example, one way to compute sin 0.6 = .6 - .6^3/3! + .6^5/5!
  - ··· correct to 1000 decimal places is to call series with the
  parameter values: terms = 226, num0 = 3, denom0 = 5, (and
  since ith term/(i-1)th term = -.6^2/2i(2i+1)) numerator(i) =
  -9 and denominator(i) = 50i(2i+1). By taking base = 100000
  and places = 200, five decimal digits of the result will be obtained
  per word of the array digit.
    The use of a large base (and, consequently, smaller places)
  results in faster computation, as the number of operations is
  proportional to (places×terms) for large values of terms and
  places. However, the intermediate products (base×num[i]×
  coef[i]) (and coef[i] can almost equal denom[i]) should not ex-
  ceed the largest number representable by an integer variable.
  Also within this limit should be the product of base and the in-
  teger portion of the result;
begin
  integer i, j, k, l; integer array num[-1:terms], denom,
    coef[0:terms];
  comment Express the series by the expression
```

$$\frac{n_0}{d_0}\left(c_0 + \frac{n_1}{d_1}\left(c_1 + \cdots + \frac{n_t}{d_t}(c_t)\cdots\right)\right) \qquad (1)$$

```
    where n_i and d_i are positive and c_i are ±1. (For short, n, d, c
    and t in (1) stand for num, denom, coef and terms, respectively);
  num[-1] := 1; num[0] := abs(num0); denom[0] := abs-
  (denom0); coef[0] := sign(num0) × sign(denom0);
  for j := 1 step 1 until terms do
  begin
    k := numerator(j); l := denominator(j); num[j] := abs(k);
    denom[j] := abs(l); coef[j] := coef[j-1] × sign(k) × sign(l)
  end;
  comment Calculate digits one at a step by extracting the in-
    teger part of base × (1) and restoring the fractional part in
    form (1);
  for i := 1 step 1 until places do
  begin
    l := 0;
    for j := terms step -1 until 0 do
    begin
      k := num[j] × (coef[j]×base+l); l := k ÷ denom[j];
      coef[j] := k - l × denom[j]; num[j] := num[j-1]
    end j;
    digit[i] := l
  end i;
  comment Some digits may be negative or larger than base in
    absolute value. Process the array digit to obtain true base
    representation;
  l := 0;
  for i := places step -1 until 1 do
  begin
    k := digit[i] + l; l := k ÷ base; digit[i] := k - base × l;
    if digit[i] < 0 then
    begin digit[i] := digit[i] + base; l := l - 1 end
  end;
  digit[0] := l; sgn := sign(l);
  if l < 0 then
  begin
    digit[0] := -l - 1; digit[places] := digit[places] - 1;
    for i := 1 step 1 until places do digit[i] := base - 1 - digit[i]
  end
end series
```