A note on the generalised Euler transformation*

P. Wynn

Department of Mathematics, Louisiana State University in New Orleans

The generalised Euler transformation is a well-known device for accelerating the numerical convergence of infinite series. In practice the transformation is often applied to the infinite series remaining after the first m terms have been added together to form a partial sum. The other partial sums, obtained by taking the first m terms of the original series and the first r terms of the transformed remaining series form a double sequence of approximations to the sum or formal sum of the original series. The purpose of this note is to point out that the partial sums may be built up by means of a remarkably simple recursion.

The generalised Euler transformation

The generalised Euler transformation is a well-known device (see, for example, Hartree (1952) Ch. 12) for accelerating the numerical convergence of infinite series; it functions in the following way: given the series

$$\sum_{s=0}^{\infty} u_s \tag{1}$$

to be transformed, it is assumed that its terms behave like those of a geometric progression with ratio z, so that we may write

$$u_m = z^m v_m \qquad (m = 0, 1, \ldots) \tag{2}$$

where the quantities $\{v_m\}$ are approximately constant. If E is the displacement operator and Δ the difference operator relating to members of the sequence $\{v_m\}$, so that

$$E^s v_m = v_{m+s} \qquad (m = 0, 1, \ldots; s = 0, 1, \ldots) \tag{3}$$

$$\Delta = E - 1, \tag{4}$$

then we have, quite formally,

$$\begin{aligned} \sum_{s=0}^{\infty} u_s &= \sum_{s=0}^{\infty} z^s v_s \\ &= \sum_{s=0}^{\infty} (zE)^s v_o \\ &= \left\{\frac{1}{1 - zE}\right\} v_o \\ &= \frac{1}{1 - z} \sum_{s=0}^{\infty} \left(\frac{z}{1 - z}\right)^s \Delta^s v_o \,. \end{aligned} \tag{5}$$

It is to be expected that the successive differences $\{\Delta^s v_o\}$ will decrease sharply in magnitude, and that the series (5) will converge numerically with greater rapidity than the original series (1).

Some results concerning the convergence of the series (5) are given in Cherry (1950); others may be deduced from the theory presented in van Wijngaarden. An extensive convergence theory for the special case in which $z = -1$ (formula (5) is then simply called the Euler transformation) is given in Hardy (1963) Ch. IV.

The delayed form of the generalised Euler transformation

In practice it is often wise to delay application of the transformation (5): to add together the first m terms in the partial sum $\sum_{s=0}^{m-1} u_s$ in the usual way, and transform the remaining infinite series $\sum_{s=0}^{\infty} u_{m+s}$ by a formula similar to (5): one obtains the generalised Euler transformation in its delayed form

$$\sum_{s=0}^{\infty} u_s = \sum_{s=0}^{m-1} u_s + \frac{z^m}{1 - z} \sum_{s=0}^{\infty} \left(\frac{z}{1 - z}\right)^s \Delta^s v_m \,. \tag{6}$$

It frequently occurs that the successive differences $\{\Delta^s v_m\}$ (where m is a fixed finite positive integer) decrease in magnitude far more sharply than do the differences $\{\Delta^s v_o\}$; i.e. the numerical convergence of the infinite series in (6) is more rapid than that of (5).

It is clear that the partial sums

$$S_m^{(r)} = \sum_{s=0}^{m-1} u_s + \frac{z^m}{1 - z} \sum_{s=0}^{r-1} \left(\frac{z}{1 - z}\right)^s \Delta^s v_m \qquad (r = 0, 1, \ldots; m = 0, 1, \ldots) \tag{7}$$

form a double sequence of approximations to the sum or formal sum of the original series (1); they may be placed in the following two dimensional array:

$$\begin{array}{lllll} S_0^{(0)} & & & & \\ S_1^{(0)} & S_0^{(1)} & & & \\ S_2^{(0)} & S_1^{(1)} & S_0^{(2)} & & \\ \cdot & \cdot & \cdot & & \\ \cdot & \cdot & \cdot & & \\ \cdot & \cdot & \cdot & & \\ S_{m-1}^{(0)} & S_{m-2}^{(1)} & S_{m-3}^{(2)} & \ldots & S_0^{(m-1)} \\ S_m^{(0)} & S_{m-1}^{(1)} & S_{m-2}^{(2)} & \ldots & S_1^{(m-1)} \quad S_0^{(m)} \end{array}$$

Fig. 1

For the sake of clarity, we point out that the successive partial sums of the series (1) are to be found in the first column of this array, and the successive partial sums of the series (5) in the first diagonal.

*Technical Report No. 42, April 1970.

An algorithm

The main purpose of this note is to point out that the partial sums of formula (7) may be built up by means of a remarkably simple recursion.

Theorem. *The quantities* $\{S_m^{(r)}\}$ *given by equation* (7) *obey the recursion*

$$S_m^{(r+1)} = p\,S_{m+1}^{(r)} + q\,S_m^{(r)} \quad (r = 0, 1, \ldots; m = 0, 1, \ldots) \tag{8}$$

where

$$p = \frac{1}{1-z}, \qquad q = -\frac{z}{1-z}. \tag{9}$$

Proof. We have first of all

$$S_m^{(0)} = \sum_{s=0}^{m-1} z^s v_s \quad (m = 0, 1, \ldots) \tag{10}$$

and

$$\begin{aligned} S_m^{(1)} &= \sum_{s=0}^{m-1} z^s v_s + \frac{z^m}{1-z} v_m \\ &= S_m^{(0)} + \frac{1}{1-z}(S_{m+1}^{(0)} - S_m^{(0)}) \\ &= p\,S_{m+1}^{(0)} + q\,S_m^{(0)}, \quad (m = 0, 1, \ldots) \end{aligned} \tag{11}$$

so that equation (8) is true for $r = 0$, $m = 0, 1, \ldots$.

Assume that equation (8) is true for when r is replaced by the integers $0, 1, \ldots, r-1$, and $m = 0, 1, \ldots$. We know that

$$\Delta^r v_m = \Delta^{r-1} v_{m+1} - \Delta^{r-1} v_m \; (r = 1, 2, \ldots; m = 0, 1, \ldots) \tag{12}$$

or, in terms of the members of the sequence $\{S_m^{(r)}\}$

$$\begin{aligned} \frac{(1-z)^{r+1}}{z^{m+r}}\{S_m^{(r+1)} - S_m^{(r)}\} &= \frac{(1-z)^r}{z^{m+r}}\{S_{m+1}^{(r)} - S_{m+1}^{(r-1)}\} \\ &\quad - \frac{(1-z)^r}{z^{m+r-1}}\{S_m^{(r)} - S_m^{(r-1)}\}, \quad (m = 0, 1, \ldots) \end{aligned} \tag{13}$$

i.e.

$$\begin{aligned} S_m^{(r+1)} &= S_m^{(r)} + \frac{1}{1-z} S_{m+1}^{(r)} - \frac{1}{1-z} S_{m+1}^{(r-1)} - \frac{z}{1-z} S_m^{(r)} \\ &\qquad + \frac{z}{1-z} S_m^{(r-1)} \\ &= \frac{1}{1-z} S_{m+1}^{(r)} - \frac{z}{1-z} S_m^{(r)} + \\ &\qquad \left\{ S_m^{(r)} - \frac{1}{1-z} S_{m+1}^{(r-1)} + \frac{z}{1-z} S_m^{(r-1)} \right\} \\ &= p\,S_{m+1}^{(r)} + q\,S_m^{(r)}. \quad (m = 0, 1, \ldots) \end{aligned} \tag{14}$$

It follows that formula (8) is generally true.

We remark that the quantities referred to in equation (8) occupy the following relative positions in the array of **Fig. 1.**

$$\begin{matrix} S_m^{(r)} \\ S_{m+1}^{(r)}\ S_m^{(r+1)} \end{matrix}$$

Fig. 2

When $z = -1$, recursion (8) is well known; indeed it is used in the ALGOL 60 report (Backus *et al.*, 1960) to illustrate the use of this language.

The determination of the parameter z

In many cases the most suitable choice of the parameter z is clear from an explicit formula for the terms $\{u_m\}$ occurring in formulae (2). In others the choice may be determined by analysis or by numerical estimation.

Concerning the cases in which z is determined by analysis, we remark that linear difference equations with polynomial coefficients of certain types in the independent variable m have solutions, involving exponential functions and inverse factorials, of the form

$$\sum_{\nu=0}^{h} z_\nu^m \sum_{\tau=0}^{\infty} v_\tau^{(\nu)} \prod_{i=1}^{\tau} (\sigma_\nu + m + i)^{-1} \quad (m = 0, 1, \ldots)$$

where $\{z_\nu\}$, $\{\sigma_\nu\}$ are fixed constants and the $\{v_\tau^{(\nu)}\}$ are fixed coefficients (see, for example Nörlund (1937) and Milne-Thomson (1951). If the terms $\{u_m\}$ of a series to be transformed satisfy such a difference equation and, in the notation of the above expression, one of the numbers $\{z_\nu\}$, $z_{\nu'}$ say, is of largest modulus, we may take $z = z_{\nu'}$ in formulae (6). Again if the terms $\{u_m\}$ have a representation of the form

$$u_m = \left\{ \int_a^b \delta^m d\alpha(\delta) \right\} x^m \quad (m = 0, 1, \ldots)$$

where $\alpha(\delta)$ is a bounded non-decreasing function of the real variable δ in the range $-\infty < a \leq \zeta \pm b < \infty$ and x is fixed finite complex number, it may also be possible to determine a suitable choice of z: if $|b| > |a|$ ($|a| > |b|$) and $\alpha(\delta)$ has points of increase in an arbitrarily small left (right) neighbourhood of the point $b(a)$, we may take $z = bx(z = ax)$.

Numerical estimation of the parameter z is perhaps best carried out by applying an extrapolated limit technique, such as the ε-algorithm (Wynn, 1956a) or the ρ-algorithm (Wynn, 1956b). (ALGOL programmes for implementing the former are given in Wynn (1962) and Wynn (1966); they may be adapted by trivial modification for implementing the second) to the sequence $\frac{u_{m+1}}{u_m}$ $(m = 0, 1, \ldots)$.

The value of the parameter z determined by the methods suggested in the preceding paragraphs is $\lim_{m=\infty} \frac{u_{m+1}}{u_m}$. For the sake of completeness we remark that the Euler transformation may also be applied with resulting improvement in numerical

Table 1

m/r	0	1	2	3	4	5	6	7	8	9
1	+1·0	+0·3333								
2	0·0	0·6667	0·4444							
3	+1·3333	0·4444	0·5926	0·4938						
4	−0·6667	0·6667	0·5185	0·5679	0·5185					
5	+2·5333	0·4000	0·5778	0·5383	0·5580	0·5317				
6	−2·8000	0·7556	0·5185	0·5580	0·5449	0·5536	0·5390			
7	+6·3429	0·2476	0·5862	0·5411	0·5524	0·5474	0·5515	0·5432		
8	−9·6571	+1·0095	0·5016	0·5580	0·5467	0·5505	0·5484	0·5505	0·5456	
9	+18·7873	−0·1757	0·6145	0·5392	0·5518	0·5484	0·5498	0·5489	0·5500	0·5471

convergence to many series, for example to $\sum_{m=0}^{\infty}(-1)^m m!\,\lambda^{-m-1}$ $(0 < \lambda < \infty)$, for which this limit does not exist.

A numerical example

The results of the above theorem are illustrated in **Table 1** which relates to the transformation of the series (7) when

$$u_s = (s+1)^{-1}x^s \qquad (s = 0, 1, \ldots) \tag{15}$$

and $x = -2$; we have taken, in the notation of equation (2),

$$z = x, \qquad v_s = (s+1)^{-1} \qquad (s = 0, 1, \ldots). \tag{16}$$

The reader with a facility for mental arithmetic will quickly note that the members of Table 1 satisfy the recursion

$$S_m^{(r+1)} = \tfrac{1}{3}S_{m+1}^{(r)} + \tfrac{2}{3}S_m^{(r)} \quad (r = 0, 1, \ldots; m = 0, 1, \ldots)$$

The value of $S_0^{(0)}$ (which in the notation of equation (7) is always zero) has been omitted.

The terms (15) are those of the (divergent) power series expansion of the function $-x^{-1}\ln(1-x)$, whose value when $x = -2{\cdot}0$ is 0·549 306 to six decimal places. It can be seen that the best approximation ($S_2^{(7)}$) in Table 1 to the formal sum of the original series is not to be found in the leading diagonal. We mention in passing that considerably better results are to be found in a slightly extended version of Table 1: for example, $S_{13}^{(4)} = 0{\cdot}549\,305$.

ALGOL programs

We give two programs which exploit the algorithmic relationships (8) in the application of the generalised Euler transformation. The first displays a small initial section of Fig. 1 and is to be used in a provisional inquiry as to the efficacy of the transformation in any particular case; the second is for actual use in computing the transformed sum of a given series.

In both cases, the transformed sums of Fig. 1 are computed row by row. Assuming that the row stretching from $S_m^{(0)}$ to $S_0^{(m)}$ has been computed; the new term u_m is evaluated and equation (8) is used to evaluate the row of partial sums stretching from $S_{m+1}^{(0)}$ to $S_0^{(m+1)}$ progressively from left to right. Equation (8) is used in the form

$$S_{m-r}^{(r+1)} = p\,S_{m-r+1}^{(r)} + q\,S_{m-r}^{(r)}. \quad (r = 0, 1, \ldots, m) \tag{18}$$

A program for display

Since the number of transformed sums computed in a preliminary investigation is relatively small, no great wastage of storage space in a computer occurs if the transformed sums $\{S_m^{(r)}\}$ are computed as members of what is effectively a triangular array and stored as such.

In order to bring the concept of a triangular array within the facilities offered by ALGOL, the members of the array are mapped onto a vector; it is assumed that the indices (*m dash*, *r dash*) relating to an array of functions such as $S_m^{(r')}$ run from a minimum, *O*, to a maximum, *dim*. The members of the array are stored as a vector according to the following scheme:

(Since, in the notation of equation (7), $S_0^{(0)}$ is always zero, there is no need to store this number, and in the ALGOL program to be given the storage location for this number (namely 0) is not reserved.) The required mapping is carried out by means of the following general purpose

```
integer procedure triangular array(dim, m dash, r dash);
value dim,m dash,r dash;
integer dim,m dash,r dash;
  triangular array := m dash + r dash × dim − (r dash ×
    (r dash − 3)) ÷ 2;
```

A second general procedure, of which use is made while printing the results, is the

```
integer procedure lesser of(first, second);
value first, second;
integer first, second;
  lesser of := (if first < second then first else second);
```

Use is made of a global integer *m*, which plays the role of the suffix *m* in the variable v_m. Its value is adjusted by the display procedure immediately before the evaluation of each v_m.

The display procedure may now be given. Its input parameters are

z: a real variable occurring in equations (2), (7), (8) and (9).
v: a real variable playing the role of v_m in equations (2) and (7).
m max: an integer. The transformed sums $\{S_m^{(r)}\}$ are derived from the terms $u_m (m = 0, 1, \ldots, m\ max)$ of the original series.
col: an integer. It is assumed that the output typewriter prints *col* numbers to a line. The results are printed out in a form similar to that of Table 1 cut into vertical strips of *col* columns.

```
procedure Display Generalised Euler (z, v, m max, col);
value z, m max, col;
integer m max, col;
real z, v;
begin integer m max plus 1;
  m max plus 1 := m max + 1;
  begin integer procedure s scheme (m, r);
      integer m, r;
        s scheme := triangular array (m max plus 1, m, r);
    begin Boolean m equals zero;
      integer m plus 1, r, m minus r, r anfang;
      real p, q, z to power m, u;
      real array S[1: s scheme (0, m max plus 1)];
      comment See equations (9);
      p := 1·0/(1·0 − z); q := −z × p;
      for m := 0 step 1 until m max do
      begin m equals zero := (m = 0); m plus 1 := m + 1;
        z to power m := (if m equals zero then 1·0
                         else z × z to power m);
        comment See equation (2);
        u := z to power m × v;
        comment S_m^(0) is computed, see equation (10);
        S[m plus 1] := (if m equals zero then 0·0
          else S[m]) + u;
        comment S_m^(1) is computed, see equation (11);
        S[m max plus 1 + m plus 1] := (if m equals zero
          then 0·0 else S[m]) + p × u;
```

m \ r	0	1	2 . . .	$dim - 1$	dim
0	0				
1	1	$dim + 1$			
2	2	$dim + 2$	$2dim + 1$		
:	:	:	:		
$dim - 1$	$dim - 1$	$2dim - 1$	$3dim - 2$ . . .	$((dim + 4) \times (dim - 1)) \div 2$	
dim	dim	$2dim$	$3dim - 1$ . . .	$((dim \times (dim + 3)) \div 2) - 1$	$(dim \times (dim + 3)) \div 2$

Fig. 3

```
    for r := 1 step 1 until m do
    begin m minus r := m − r;
      comment See equation (18);
      S[s scheme (m minus r, r + 1)] :=
        p × S[s scheme (m minus r + 1, r)] + q ×
          S[s scheme (m minus r, r)];
    end computing a row of transformed sums
  end evaluating all the terms:
  for r anfang := 0 step col until m max plus 1 do
  begin comment NLCR operates the New Line Carriage
        Return mechanism of the typewriter;
    NLCR; for m := (if r anfang = 0 then 1 else 0)
        step 1 until m max plus 1 − r anfang do
    begin NLCR; for r := r anfang step 1 until
        (r anfang + lesser of (m, col − 1)) do
      print (S[s scheme (m − r + r anfang, r)])
    end printing out a vertical strip of the S-array
  end printing out the entire S-array
 end the block in which the size of the S-array is declared
end the block in which the procedure s scheme operates
end Display Generalised Euler;
```

In order to illustrate the use of this procedure, we introduce the real procedure ln v which computes the quantities $v_m = (m+1)^{-1}$ $(m = 0, 1, \ldots)$;

```
real procedure ln v;
  ln v: = 1·0/(m + 1);
```

Use of this procedure is, of course, hardly necessary in the present example; the formal parameter v of Display Generalised Euler may be replaced by the arithmetic expression 1·0/$(m+1)$. However, in a more complicated case, if not in this, it is aesthetically more pleasing to make use of such a procedure.

A complete program which uses the above procedure is the following:

```
begin integer m, max m, col;
  real z;
  comment This comment should be replaced by the above
      procedures
    lesser of, triangular array, Display Generalised Euler and
      ln v;
  read (z, max m, col); print (z, −(ln(1·0 − z))/z);
  Display Generalised Euler (z, ln v, max m, col)
end
```

The results of Table 1, cut into two vertical strips of five columns, may be produced by use of this programme with the input parameters

$z = -2{\cdot}0$, *max m* = 8, *col* = 5.

A program for use

Two main problems must be faced in the design of a program which actually computes the transformed sum of a given series. They relate to the economical use of computer storage and a criterion for terminating the computations.

Although the transformed sums $\{S_m^{(r)}\}$ may be placed in a two dimensional array, it is possible and (when computing a large number of these sums) it is desirable to arrange that the intermediate results required for the continuation of the computation should occupy a one dimensional array in the storage. After the computation of $S_{m-r}^{(r)}$ $(r = 0, 1, \ldots, m)$, this vector (we call it l), stretches along this row of transformed sums, l_r containing $S_{m-r}^{(r)}$ $(r = 0, 1, \ldots, m)$; a new term u_m is computed, and progressively the vector l is pushed down by one step in the S-array, so as to contain $S_{m-r+1}^{(r)}$ $(r = 0, 1, \ldots, m+1)$. The mechanism for accomplishing this requires two auxiliary storage spaces, called *aux* 0 and *aux* 1.

In **Fig. 4** this process is illustrated at an intermediate stage.

We assume that the intended contents $(S_{m-r+1}^{(r)})$ of l_r have been computed and lie in *aux* 0, and that the intended contents

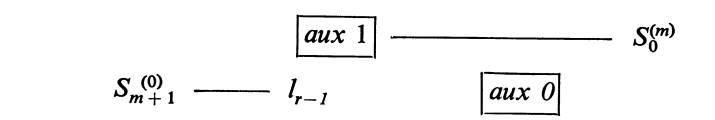

Fig. 4

$(S_{m-r+2}^{(r-1)})$ of l_{r-1} have also been computed and lie in *aux* 1. The first step in the computation is to replace the actual contents $(S_{m-r+1}^{(r-1)})$ of l_{r-1} by those of *aux* 1, and then those of *aux* 1 by those of *aux* 0. The new contents of *aux* 0 are now computed from those of *aux* 0 and l_r ($S_{m-r+1}^{(r)}$ and $S_{m-r}^{(r)}$ respectively), the suffix is increased by unity, and the cycle is repeated.

Clearly special steps must be taken to prepare the above mechanism; furthermore the process must be terminated by transferring the contents of *aux* 1 and *aux* 0 to the appropriate positions of the vector l.

The decision as to which of the transformed sums is to be accepted as a suitable approximation to the sum or formal sum of the original series is based upon the following strategy which has proved to be satisfactory in many practical examples: the relative differences of succeeding pairs of transformed sums $S_{m-r}^{(r)}$ and $S_{m-r+1}^{(r+1)}$ (lying adjacently on a row of Fig. 4) are examined. When the relative differences of *tim* such consecutive pairs are all less in absolute magnitude than a prescribed quantity, δ say, than the computation is terminated; the second member of the last pair is accepted as the required approximation. The positive integer *tim*, and the small positive real quantity δ are prescribed by the user.

Use is again made of a global integer m, which plays the same role as before.

The program for use may now be given; its formal parameters are

z, v: real variables, whose function is as described for the display procedure

relative error: a real variable whose value, δ, is prescribed by the user. Its significance is described above

the very end: a positive integer prescribed by the user. The components l_r of Fig. 4 may have indices r running from 0 to *the very end*

tim: a positive integer whose role is described above

storage not exceeded: a Boolean variable; if the computations have been terminated according to the criterion described above, before all the positions of the vector l have been used, then this variable is given the value true, otherwise the value false.

transformed sum: a real variable; if without exceeding the storage capacity of the vector l, for some m and r

$$\text{abs}\left(1 - \frac{S_{m-r-r'+1}^{(r+r'-1)}}{S_{m-r-r'}^{(r+r')}}\right) \leq \delta \quad (r' = 1, 2, \ldots, tim)$$

then *transformed sum* is given the value $S_{m-r-tim}^{(r+tim)}$.

```
procedure Generalised Euler (z, v, relative error, the very end,
    tim, storage not exceeded, transformed sum);
value z, relative error, the very end, tim;
Boolean storage not exceeded;
integer the very end, tim;
real z, v, relative error, transformed sum;
begin integer number of times agreement has been reached, r;
  real p, q, u, z to power m, aux 0, aux 1;
  real array l[0: the very end];
  comment See equations (9);
  p := 1·0/(1·0 − z); q := −z × p;
  comment The row of transformed sums is prepared by
    inserting S0(1) and S1(0);
  m := 0; l[0] := u := v; l[1] := p × u; z to power
    m := 1·0;
```

```
compute new term: m := m + 1;
  if m = the very end then
    begin comment The computation has failed due to
                  insufficient storage space;
      storage not exceeded := false; goto all is over
    end;
  comment See equation (2);
  z to power m := z × z to power m; u := z to power m × v;
  comment The mechanism of Fig. 4 is prepared;
  aux 1 := l[0] + u; aux 0: = l[0] + p × u;
  number of times agreement has been reached := 0;
  for r := 1 step 1 until m do
  begin comment The mechanism of Fig. 4 now functions;
    l[r − 1] := aux 1; aux 1 := aux 0 ;
    comment See equation (8);
    aux 0 := p × aux 1 + q × l[r];
    if abs (1·0 − aux 1/aux 0) ≤ relative error then
      begin comment Provisional agreement has been reached;
        number of times agreement has been reached :=
          number of times agreement has been reached + 1;
        if number of times agreement has been reached = tim
        then
          begin transformed sum := aux 0; storage not
                exceeded := true;
            goto all is over
          end
      end else
        begin comment If provisional agreement is not
                      sustained counting must begin anew;
          number of times agreement has been reached := 0
        end
    end r, working across a line of Fig. 4;
    comment The mechanism of Fig. 4 is terminated;
    l[m] := aux 1; l[m + 1] := aux 0; goto compute new
                                          term;
all is over:
end Generalised Euler;
```

This procedure may be used in the following complete program:

```
begin Boolean successful;
  integer m, tim, available storage;
  real z, delta, result;
  comment This comment must be replaced by the
    procedure Generalised Euler and ln v;
  read (z, delta, tim, available storage);
  print (z, delta, tim, available storage);
  Generalised Euler (z, ln v, delta, available storage, tim,
    successful, result);
  if successful then print (result)
end
```

The reader may care to experiment with the above program using the input variables

$z = -2{\cdot}0$, delta $= 10^{-4}$, *tim* $= 3$, *available storage* $= 100$.

References

BACKUS, T. W. *et al*, (1960). Report on the Algorithmic Language ALGOL 60, *Num. Math.*, Vol. 2, p. 106.

CHERRY, T. M. (1950). Summation of Slowly Convergent Series, *Proc. Camb. Phil., Soc.*, Vol. 46, pp. 436-449.

HARDY, G. H. (1963). *Divergent Series*, Oxford.

HARTREE, D. R. (1952). *Numerical Analysis*, Oxford.

MILNE-THOMSON, L. M. (1951). *The Calculus of Finite Differences*, Macmillan, London.

NORLAND, N. E. (1937). *Vorlesungen über Differenzenrechnung*, Springer, Berlin.

VAN WIJNGAARDEN, A. A Transformation of Formal Series, *Proc. Kon. Ned. Akad. v. Wet., Ser A*, Vol. 56.

WYNN, P. (1956a). On a Device for Computing the $e_m(S_n)$ Transformation, *Maths. of Comp.*, Vol. 10.

WYNN, P. (1956b). On a Procrustean Technique for the Numerical Transformation of Slowly Convergent Sequences and Series, *Proc. Camb. Phil. Soc.*, Vol. 52. pp. 663-671.

WYNN, P. (1966). An Arsenal of Algol Procedures for the Evaluation of Continued Fractions and for Effecting the Epsilon Agorithm, *Rev. Franç. de Trait. de l'Inf.* (*Chiffres*), Vol. 9, pp. 327-362.

WYNN, P. (1962). An Arsenal of Algol Procedures for Complex Arithmetic, *Nord. Tids. for Inf. Beh.* (*BIT*), Vol. 4, pp. 231-255.