STICHTING
MATHEMATISCH CENTRUM
2e BOERHAAVESTRAAT 49
AMSTERDAM

MR 59

Singular Rules For Certain Non-Linear Algorithms

P. Wynn

1963

BIT 3 (1963), 175–195

SINGULAR RULES FOR CERTAIN NON-LINEAR ALGORITHMS*

P. WYNN

Abstract.

Certain simple non-linear algorithms which have useful application in numerical analysis have recently been introduced. When using the formulae of these algorithms it may sometimes occur that a quantity is numerically ill-determined. As a result of this and because of the way in which the algorithmic formulae are used, all quantities lying in a certain sector become ill-determined. By considering the ε-, ϱ-, and $q-d$-algorithms in detail it is shown how this misfortune may be overcome. ALGOL programmes are developed to show how the derived formulae may be mechanized.

Introduction.

Recently a number of algorithms, which have important application in Numerical Analysis, have been developed.

These algorithms relate members of an array of functions or numbers, the general member of which may be denoted for the purpose of exposition by $\varphi_s^{(m)}$. The array may be displayed as follows:

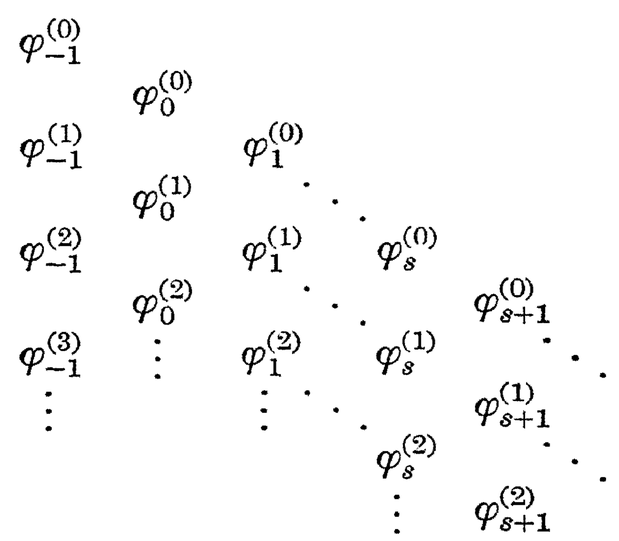


Diagram I

so that the superscript m denotes a diagonal and the suffix s a column.

---

* Communication MR59 of the Computation Department of the Mathematical Centre, Amsterdam.

All the algorithms considered are lozenge diagram algorithms, that is, each quantity $\varphi_s^{(m)}$ is derived from a relationship of the form

$$\theta_s^{(m)}\{\varphi_s^{(m)}, \varphi_{s-1}^{(m)}, \varphi_{s-1}^{(m+1)}, \varphi_{s-2}^{(m+1)}\} = 0 \tag{1}$$

which affects quantities lying at the corners of a lozenge in the described array. In certain of the algorithms considered here, the form of the functional relationship $\theta_s^{(m)}\{\ldots\}$ is independent of s, whilst in others $\theta_s^{(m)}\{\ldots\}$ differs according to whether s is even or odd.

The algorithmic relationship (1) may be used in two ways. In the forward application of the algorithm, (1) is solved for $\varphi_s^{(m)}$ and the φ-array is built up from left to right. In the progressive application of the algorithm, (1) is solved for $\varphi_{s-1}^{(m+1)}$ and the φ-array is constructed downwards.

Now it may occur that in certain singular cases, (1) results in an ill-determined or even an indeterminate value of $\varphi_s^{(m)}$ or $\varphi_{s-1}^{(m+1)}$. In view of the manner in which the φ-array is built up, this misfortune is propagated throughout a whole sector. It is the purpose of this note to show how this difficulty may be overcome.

The ε-Algorithm [1]

In order to facilitate the discussion an example, which is perhaps the simplest of its kind, will immediately be given.

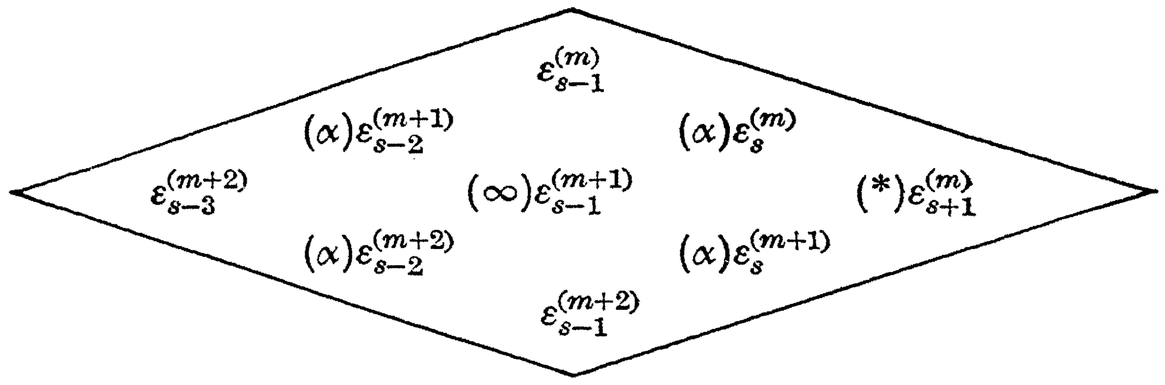
The ε-algorithm relationships are

$$(\varepsilon_{s+1}^{(m)} - \varepsilon_{s-1}^{(m+1)})(\varepsilon_s^{(m+1)} - \varepsilon_s^{(m)}) = 1 \tag{2}$$

which in the forward form are

$$\varepsilon_{s+1}^{(m)} = \varepsilon_{s-1}^{(m+1)} + \frac{1}{\varepsilon_s^{(m+1)} - \varepsilon_s^{(m)}}. \tag{3}$$

Now suppose that, quite fortuitously, $\varepsilon_{s-2}^{(m+2)}$ and $\varepsilon_{s-2}^{(m+1)}$ are both equal to α. It can immediately be seen that $\varepsilon_{s-1}^{(m+1)}$ becomes formally infinite, $\varepsilon_s^{(m)}$ and $\varepsilon_s^{(m+1)}$ are equal to α, and $\varepsilon_{s+1}^{(m)}$ is indeterminate. Further quantities in a sector whose vertex is at $\varepsilon_{s+1}^{(m)}$ remain undetermined. The state of affairs is as follows.



The situation may however be retrieved by multiple appeal to relationships (3). There follows

$$\begin{aligned}
\varepsilon_{s+1}^{(m)} &= \varepsilon_{s-1}^{(m+1)} + \cfrac{1}{\varepsilon_{s-1}^{(m+2)} + \cfrac{1}{\varepsilon_{s-1}^{(m+2)} - \varepsilon_{s-1}^{(m+1)}} - \varepsilon_{s-2}^{(m+1)} - \cfrac{1}{\varepsilon_{s-1}^{(m+1)} - \varepsilon_{s-1}^{(m)}}} \\
&= \varepsilon_{s-1}^{(m+1)} + \cfrac{1}{\cfrac{1}{\varepsilon_{s-1}^{(m+1)} - \varepsilon_{s-3}^{(m+2)}} + \cfrac{1}{\varepsilon_{s-1}^{(m+2)} - \varepsilon_{s-1}^{(m+1)}} - \cfrac{1}{\varepsilon_{s-1}^{(m+1)} - \varepsilon_{s-1}^{(m)}}} \\
&= \varepsilon_{s-1}^{(m+1)} - \cfrac{\varepsilon_{s-1}^{(m+1)}}{1 - \cfrac{\varepsilon_{s-3}^{(m+2)}}{\varepsilon_{s-1}^{(m+1)} - \varepsilon_{s-3}^{(m+2)}} + \cfrac{\varepsilon_{s-1}^{(m+2)}}{\varepsilon_{s-1}^{(m+1)} - \varepsilon_{s-1}^{(m+2)}} + \cfrac{\varepsilon_{s-1}^{(m)}}{\varepsilon_{s-1}^{(m+1)} - \varepsilon_{s-1}^{(m)}}} \\
&= \cfrac{\cfrac{\varepsilon_{s-1}^{(m+2)}}{1 - \varepsilon_{s-1}^{(m+2)}\varepsilon_{s-1}^{(m+1)^{-1}}} + \cfrac{\varepsilon_{s-1}^{(m)}}{1 - \varepsilon_{s-1}^{(m)}\varepsilon_{s-1}^{(m+1)^{-1}}} - \cfrac{\varepsilon_{s-3}^{(m+2)}}{1 - \varepsilon_{s-3}^{(m+2)}\varepsilon_{s-1}^{(m+1)^{-1}}}}{1 - \cfrac{\varepsilon_{s-3}^{(m+2)}}{\varepsilon_{s-1}^{(m+1)} - \varepsilon_{s-3}^{(m+2)}} + \cfrac{\varepsilon_{s-1}^{(m+2)}}{\varepsilon_{s-1}^{(m+1)} - \varepsilon_{s-1}^{(m+2)}} + \cfrac{\varepsilon_{s-1}^{(m)}}{\varepsilon_{s-1}^{(m+1)} - \varepsilon_{s-1}^{(m)}}}
\end{aligned} \tag{4}$$

or writing

$$a = \varepsilon_{s-1}^{(m+2)}(1 - \varepsilon_{s-1}^{(m+2)}\varepsilon_{s-1}^{(m+1)^{-1}})^{-1} + \varepsilon_{s-1}^{(m)}(1 - \varepsilon_{s-1}^{(m)}\varepsilon_{s-1}^{(m+1)^{-1}})^{-1} - \varepsilon_{s-3}^{(m+2)}(1 - \varepsilon_{s-3}^{(m+2)}\varepsilon_{s-1}^{(m+1)^{-1}})^{-1} \tag{5}$$

$$\varepsilon_{s+1}^{(m)} = a(1 + a\varepsilon_{s-1}^{(m+1)^{-1}})^{-1} . \tag{6}$$

It will be observed that if $\varepsilon_{s-2}^{(m+1)}$ and $\varepsilon_{s-2}^{(m+2)}$ are nearly equal, then $\varepsilon_{s-1}^{(m+1)}$ is large and ill-determined, but $\varepsilon_s{}^{(m)}$ and $\varepsilon_s{}^{(m+1)}$ derived from (3), and $\varepsilon_{s+1}^{(m)}$ derived from (6), are quite well determined. When $\varepsilon_{s-2}^{(m+1)}$ and $\varepsilon_{s-2}^{(m+2)}$ are exactly equal, the singular rule for the ε-algorithm becomes quite simply

$$\varepsilon_{s+1}^{(m)} = \varepsilon_{s-1}^{(m+2)} + \varepsilon_{s-1}^{(m)} - \varepsilon_{s-3}^{(m+2)} . \tag{7}$$

Any number of pathological examples may be constructed to illustrate this phenomenon; it arises quite naturally however when the ε-algorithm is used to transform the slowly convergent series $\sum_{s=0}^{\infty} u_s$, in which, for some r,

$$u_r = u_{r+1} .$$

The initial conditions for the ε-algorithm in this case are

$$\varepsilon_{-1}^{(m)} = 0 \quad \varepsilon_0{}^{(m)} = \sum_{s=0}^{m-1} u_s \quad m = 1, 2, \ldots ; \ \varepsilon_0{}^{(0)} = 0 . \tag{8}$$

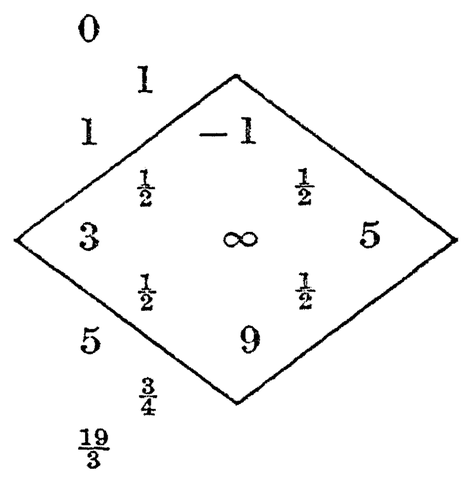
Consider the ε-scheme so derived from the series

$$u_s = x^s/s! \quad s = 0, 1, \ldots \tag{9}$$

Its initial members are as follows:

$$\begin{array}{ccccc}
 & 0 & & & \\
0 & & 1 & & 1 \\
 & 1 & & \dfrac{1}{1-x} & \\
0 & & x^{-1} & & \dfrac{2-4x+x^2}{-x^2} \\
 & 1+x & & \dfrac{2+x}{2-x} & \\
0 & & 2x^{-2} & & -\dfrac{2}{x^3}\{6-6x+x^2\} \qquad \dfrac{2(3+x)}{6-4x+x^2} \\
 & 1+x+\dfrac{x^2}{2} & & \dfrac{6+4x+x^2}{6-2x} & \\
0 & & 6x^{-3} & & \\
 & 1+x+\dfrac{x^2}{2}+\dfrac{x^3}{6} & & &
\end{array}$$

When $x=2$ the ε-scheme which may be constructed by use of (3) and (6), and verified from the above array, is:



The method of treatment to be adopted when the φ-array is constructed in the downward direction may be illustrated by the

ε-Algorithm: Progressive Form.

The progressive form of the ε-algorithm relationships is

$$\varepsilon_s^{(m+1)} = \varepsilon_s^{(m)} + \frac{1}{\varepsilon_{s+1}^{(m)} - \varepsilon_{s-1}^{(m+1)}}. \tag{10}$$

When $\varepsilon_{s-1}^{(m+1)}$ and $\varepsilon_{s+1}^{(m)}$ are approximately equal the state of affairs is as follows

$$\begin{array}{ccccc}
 & & \varepsilon_s^{(m)} & & \\
 & (\alpha)\varepsilon_{s-1}^{(m+1)} & & (\alpha)\varepsilon_{s+1}^{(m)} & \\
\varepsilon_{s-2}^{(m+2)} & & (\infty)\varepsilon_s^{(m+1)} & & \varepsilon_{s+2}^{(m)} \\
 & (\alpha)\varepsilon_{s-1}^{(m+2)} & & (\alpha)\varepsilon_{s+1}^{(m+1)} & \\
 & & (*)\varepsilon_s^{(m+2)} & &
\end{array}$$

Writing

$$A = \varepsilon_{s+2}^{(m)}(1-\varepsilon_{s+2}^{(m)}\varepsilon_s^{(m+1)^{-1}})^{-1}+\varepsilon_{s-2}^{(m+2)}(1-\varepsilon_{s-2}^{(m+2)}\varepsilon_s^{(m+1)^{-1}})^{-1}-\varepsilon_s^{(m)}(1-\varepsilon_s^{(m)}\varepsilon_s^{(m+1)^{-1}})^{-1}, \quad (11)$$

there follows

$$\varepsilon_s^{(m+2)} = A(1+A\varepsilon_s^{(m+1)^{-1}})^{-1}. \quad (12)$$

When $\varepsilon_{s-1}^{(m+1)}$ and $\varepsilon_{s+1}^{(m)}$ are exactly equal

$$\varepsilon_s^{(m+2)} = \varepsilon_{s-2}^{(m+2)}+\varepsilon_{s+2}^{(m)}-\varepsilon_s^{(m)}. \quad (13)$$

A further algorithm which is similar in form to the ε-algorithm is

The ϱ-Algorithm [2].

Quantities $\varrho_s^{(m)}$ are related by

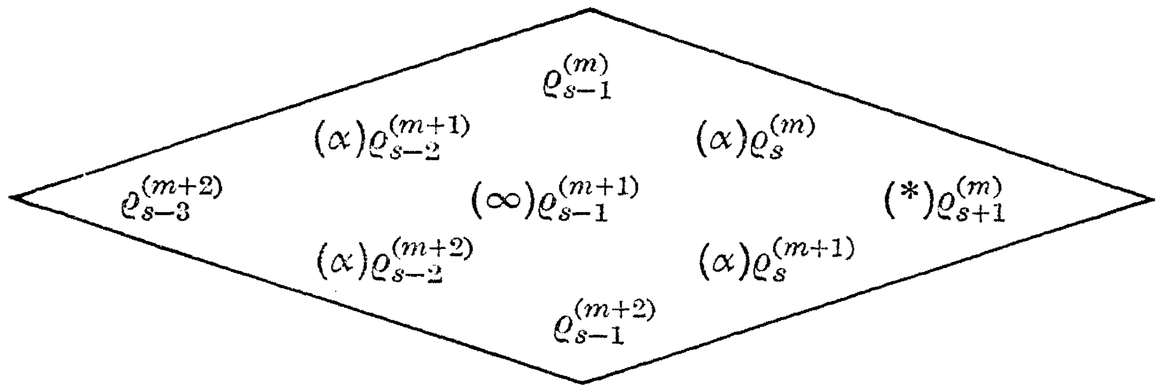
$$(\varrho_{s+1}^{(m)}-\varrho_{s-1}^{(m+1)})(\varrho_s^{(m+1)}-\varrho_s^{(m)}) = s+1. \quad (14)$$

Forward Form:

$\varrho_{s+1}^{(m)}$ is determined from

$$\varrho_{s+1}^{(m)} = \varrho_{s-1}^{(m+1)}+\frac{s+1}{\varrho_s^{(m+1)}-\varrho_s^{(m)}}. \quad (15)$$

Misfortune occurs when $\varrho_{s-2}^{(m+1)}$ and $\varrho_{s-2}^{(m+2)}$ are nearly equal:



Compute first

$$r = s\varrho_{s-1}^{(m)}(1-\varrho_{s-1}^{(m)}\varrho_{s-1}^{(m+1)^{-1}})^{-1}+s\varrho_{s-1}^{(m+2)}(1-\varrho_{s-1}^{(m+2)}\varrho_{s-1}^{(m+1)^{-1}})^{-1} - (\mathrm{s}-1)\varrho_{s-3}^{(m+2)})(1-\varrho_{s-3}^{(m+2)}\varrho_{s-1}^{(m+1)^{-1}})^{-1}, \quad (16)$$

when

$$\varrho_{s+2}^{(m)} = \frac{r}{s+1+r\varrho_{s-1}^{(m+1)^{-1}}}. \quad (17)$$

When $\varrho_{s-2}^{(m+1)}=\varrho_{s-2}^{(m+2)}$, then

$$\varrho_{s+1}^{(m)} = (s+1)^{-1}\{s\varrho_{s-1}^{(m)}+s\varrho_{s-1}^{(m+2)}-(s-1)\varrho_{s-3}^{(m+2)}\} \quad (18)$$

Note:

In the theory of the ϱ-algorithm it is shown that if quantities $\varrho_s^{(m)}$ are constructed by use of (15) from the initial conditions

$$\varrho_{-1}^{(m)} = 0,\ \varrho_0^{(m)} = \frac{\sum_{s=0}^{s=n} a_s m^s}{\sum_{s=0}^{s=n} a_s' m^s} \qquad m = 0, 1, \ldots \tag{19}$$

then

$$\varrho_{2n}^{(m)} = a_n/a_n' \qquad m = 0, 1, \ldots \tag{20}$$

Consideration of the case in which one of the initial values $\varrho_0^{(m)}$ is infinite has hitherto been impossible, but this is now made possible by application of the singular rule

$$\text{if } \varrho_0^{(m)} = \infty, \text{ then } \varrho_2^{(m-1)} = \tfrac{1}{2}(\varrho_0^{(m-1)} + \varrho_0^{(m+1)}) . \tag{21}$$

Progressive Form:

$\varrho_s^{(m+1)}$ is determined from

$$\varrho_s^{(m+1)} = \varrho_s^{(m)} + \frac{s+1}{\varrho_{s+1}^{(m)} - \varrho_{s-1}^{(m+1)}} . \tag{22}$$

Misfortune occurs when $\varrho_{s-1}^{(m+1)}$ and $\varrho_{s+1}^{(m)}$ are approximately equal.

$$\begin{array}{ccccc} & & \varrho_s^{(m)} & & \\ & (\alpha)\varrho_{s-1}^{(m+1)} & & (\alpha)\varrho_{s+1}^{(m)} & \\ \varrho_{s-2}^{(m+2)} & & (\infty)\varrho_s^{(m+1)} & & \varrho_{s+2}^{(m)} \\ & (\alpha)\varrho_{s-1}^{(m+2)} & & (\alpha)\varrho_{s+1}^{(m+1)} & \\ & & (*)\varrho_s^{(m+2)} & & \end{array}$$

Compute first

$$\begin{aligned} R = {} & (s+2)\varrho_{s+2}^{(m)}(1 - \varrho_{s+2}^{(m)}\varrho_s^{(m+1)^{-1}})^{-1} + s\varrho_{s-2}^{(m+2)}(1 - \varrho_{s-1}^{(m+2)}\varrho_s^{(m+1)^{-1}})^{-1} \\ & - (s+1)\varrho_s^{(m)}(1 - \varrho_s^{(m)}\varrho_s^{(m+1)^{-1}})^{-1} \end{aligned} \tag{23}$$

when

$$\varrho_s^{(m+2)} = \frac{R}{s+1-R\varrho_s^{(m+1)^{-1}}}; \tag{24}$$

when $\varrho_{s-1}^{(m+1)} = \varrho_{s+1}^{(m)}$,

$$\varrho_s^{(m+2)} = (s+1)^{-1}\{(s+2)\varrho_{s+2}^{(m)} + s\varrho_{s-2}^{(m+2)} - (s+1)\varrho_s^{(m)}\} . \tag{25}$$

The two algorithms considered contain only one algorithmic relationship. Lozenge diagram algorithms exist however in which two relationships are used to construct two sets of quantities which stand in the even and odd columns respectively of the φ-array. Typical of such an algorithm is

The q-d Algorithm [3].

The $q-d$ algorithm relationships are

$$e_s^{(m)} + q_s^{(m)} = e_{s-1}^{(m+1)} + q_s^{(m+1)}, \quad e_s^{(m)} q_{s+1}^{(m)} = e_s^{(m+1)} q_s^{(m+1)} . \tag{26}$$

Forward Form:

The q and e functions are determined from

$$e_s^{(m)} = e_{s-1}^{(m+1)} + q_s^{(m+1)} - q_s^{(m)}, \quad q_{s+1}^{(m)} = e_s^{(m+1)} q_s^{(m+1)} e_s^{(m)-1} . \tag{27}$$

Misfortune occurs when fortuitously $e_s^{(m)}$ is small or zero; the distribution of singular values is then

$$\begin{array}{ccccccc} & & e_{s+1}^{(m-2)} & & & & \\ & (0)q_{s+1}^{(m-1)} & & (*)q_{s+2}^{(m-2)} & & & \\ (0)e_s^{(m)} & & (\infty)e_{s+1}^{(m-1)} & & (*)e_{s+2}^{(m-2)} & & \\ & (\infty)q_{s+1}^{(m)} & & (\infty)q_{s+2}^{(m-1)} & & (*)q_{s+3}^{(m-2)} & \\ & & (\infty)e_{s+1}^{(m)} & & (0)e_{s+2}^{(m-1)} & & \\ & & & (0)q_{s+2}^{(m)} & & (*)q_{s+3}^{(m-1)} & \end{array}$$

Write successively

$$b = e_s^{(m)} - q_{s+1}^{(m-1)}, \; c = q_s^{(m+1)} e_s^{(m+1)}, \; d = q_{s+1}^{(m+1)} + e_s^{(m+1)}; \tag{28}$$

then

$$q_{s+2}^{(m-2)} = (e_s^{(m)} b + c) q_s^{(m)} e_{s+1}^{(m-2)-1} e_s^{(m-1)-1} , \tag{29}$$

$$e_{s+2}^{(m-2)} = b - q_{s+2}^{(m-2)} + (d+b)(1 + q_s^{(m)-1} b)^{-1}; \tag{30}$$

$$e = q_{s+1}^{(m+1)} e_{s+1}^{(m+1)},$$

$$f = e + (d q_{s+1}^{(m)-1} - 1)(d e_s^{(m)} - c)(1 - q_s^{(m)} e_s^{(m-1)-1})(b q_{s+1}^{(m)-1} + 1) , \tag{31}$$

$$q_{s+3}^{(m-2)} = (b q_{s+1}^{(m)-1} + 1) f e_{s+2}^{(m-2)-1} , \tag{32}$$

$$q_{s+3}^{(m+1)} = e_{s+2}^{(m)} e f^{-1} . \tag{33}$$

Progressive Form:

The q and e functions are determined from

$$q_s^{(m+1)} = e_s^{(m)} + q_s^{(m)} - e_{s-1}^{(m+1)}, \quad e_s^{(m+1)} = q_{s+1}^{(m)} e_s^{(m)} q_s^{(m+1)-1} . \tag{34}$$

Misfortune occurs when $q_s^{(m)}$ is very small.

$$\begin{array}{ccccccc} & & (0)q_s^{(m)} & & & & \\ & (0)e_{s-1}^{(m+1)} & & (\infty)e_s^{(m)} & & & \\ q_{s-1}^{(m+2)} & & (\infty)q_s^{(m+1)} & & (\infty)q_{s+1}^{(m)} & & \\ & (*)e_{s-1}^{(m+2)} & & (\infty)e_s^{(m+1)} & & (0)e_{s+1}^{(m)} & \\ & & (*)q_s^{(m+2)} & & (0)q_{s+1}^{(m+1)} & & \\ & & & (*)e_s^{(m+2)} & & (*)e_{s+1}^{(m+1)} & \end{array}$$

Write

$$B = q_s^{(m)} - e_{s-1}^{(m+1)}, \; C = e_s^{(m-1)} q_{s+1}^{(m-1)}, \; D = q_{s+1}^{(m-1)} + e_{s+1}^{(m-1)}; \tag{35}$$

then

$$e_{s-1}^{(m+2)} = e_{s-1}^{(m)} q_{s-1}^{(m+1)-1} q_{s-1}^{(m+2)} (q_s^{(m)} B + C) , \tag{36}$$

$$q_s^{(m+2)} = B - e_{s-1}^{(m+2)} + C(B+D)(C + q_s^{(m)} B)^{-1}; \tag{37}$$

further

$$E = C + (De_s^{(m)-1} - 1)(Be_s^{(m)-1} + 1)^{-1}\{DB - Cq_{s-1}^{(m+1)-1}(e_s^{(m)} - q_{s-1}^{(m+1)})\}\,, \quad (38)$$

when

$$e_s^{(m+2)} = q_s^{(m+2)-1}(1 + Be_s^{(m)-1})E\,, \quad e_{s+1}^{(m+1)} = q_{s+2}^{(m)}e_{s+1}^{(m-1)}q_{s+1}^{(m-1)}E^{-1}\,. \quad (39)$$

Further Algorithms.

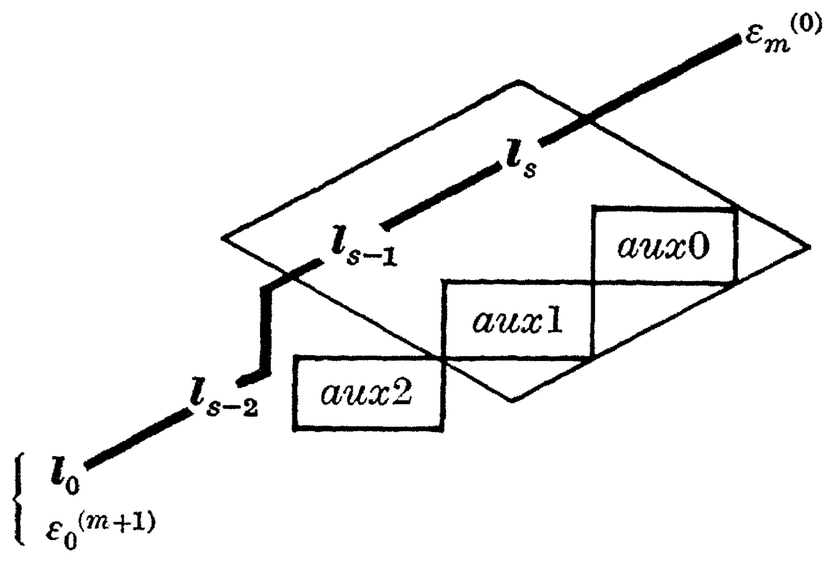
There are a number of furhter lozenge diagram algorithms extant in the literature ([4] p. 6, [4] p. 13, [4] p. 14). Singular cases arise, and may be dealt with, in a manner analogous to those which have been considered here.

ALGOL Programmes.

It is perfectly clear that the various special formulae which have either been given or may be derived are of no use unless it is shown how they may be used. In order to accomplish this the progressive form of the ε-algorithm is considered in detail. Extension of the stratagems employed to other lozenge algorithms will then be a mere exercise in transliteration.

Normal ε-Algorithm Procedures.

Firstly we consider how the quantities of the ε-array may be computed. The fact that we are concerned with a lozenge algorithm means that we are required to store, not a two dimensional array as in Diagram I, but a vector $\boldsymbol{l}$ of quantities which, in the notation of the ε-array, stretches along the backward diagonal from $\varepsilon_0^{(m)}$ to $\varepsilon_m^{(0)}$:



We arrive with a new quantity $\varepsilon_0^{(m+1)}$ and in stages push the vector $\boldsymbol{l}$ down one place so as to extend from $\varepsilon_0^{(m+1)}$ to $\varepsilon_{m+1}^{(0)}$. The process requires three auxiliary storage spaces $aux0$, $aux1$ and $aux2$. In the diagram the process of displacing the vector is not complete; $\boldsymbol{l}$ contains quantities lying along the thick line. The contents of l_{s-1}, l_s, $aux0$ and $aux1$ form a lozenge; the contents of $aux0$ are computed from those of l_{s-1}, l_s and $aux1$; the contents of l_{s-1} are replaced by those of $aux2$, those of $aux2$ by those of $aux1$, those of $aux1$ by those of $aux0$; the value of s is increased by unity, and the process is repeated.

Two ALGOL ε-algorithm procedures will now be given. The first displays the even order ε-array which may be derived from the terms u_m $(m=0,1,\ldots,mmax)$ of the series $\sum_{r=0}^{\infty}u_r$. The initial values $\varepsilon_0^{(m)}$ are put equal to the partial sums $\sum_{r=0}^{m-1}u_r$ of the series. During the computation of the quantities of the ε-array by means of the vector $\boldsymbol{l}$ as just described, the even suffix ε-quantities are mapped onto a display vector $display_{ms}$, and (subsequent to the computation of the complete ε-array) printed out in the form of a triangular array (just as in Diagram I with the odd order columns removed) in strips of *col* columns. The procedure uses a non-locally declared integer m indicating the suffix of the terms u_m.

The inputs to the procedure are an integer *mmax* indicating the suffix of the last term, a real procedure *term* which computes the terms of the series to be transformed as a function of m, and an integer *col* indicating the number of columns which can be accommodated per page upon the output typewriter. The procedure is as follows:

```
procedure display epsilon algorithm (mmax, term, col);
value mmax, col; integer mmax, col; real term;
begin integer s, sanfang, twommax;
      real aux0, aux1, aux2;
      array l[0: mmax+1], display[1: ((mmax+1)×(mmax+5))÷4];
      l[0] := 0.0; twommax := 2×mmax;
      for m := 0 step 1 until mmax do
      begin aux0 := term; aux1 := aux0+l[0];
            for s := 0 step 1 until m do
            begin aux0 := 1.0/(if s = 0 then aux0 else aux1−l[s]);
                  if s ≠ 0 then
begin aux0 := aux0+l[s−1];
      l[s−1] := aux2
end;
if even(s) then
display[(s×(twommax−s+2))÷4+m+1] := aux1;
```

```
                aux2 := aux1; aux1 := aux0
        end;
        l[m] := aux2; l[m+1] := aux1;
        if ¬even(m) then
                display[((m+1)×(twommax−m+5))÷4] := aux1
    end;
    for sanfang := 0 step 2×col until mmax+1 do
    begin NLCR; for m := 1 step 1 until mmax+1−sanfang÷2 do
            begin NLCR;
                    for s := sanfang step 2 until sanfang+2×(col−1) do
                    begin if (s÷2 ≦ m) ∧ (m ≦ mmax+1−(s÷2))
                            then
                            print (display[(s×(twommax+4−s))÷4+m])
                    end
            end
    end
end;
```

The second ALGOL procedure does not display the even order ε-array. It persists in the computation of the ε-array, using the backward diagonal $\boldsymbol{l}$ discussed above, until either if m is odd —$\varepsilon_m^{(0)}$ and $\varepsilon_{m-2}^{(2)}$, or if m is even $-\varepsilon_{m-1}^{(1)}$ and $\varepsilon_{m-3}^{(3)}$, agree to within a prescribed relative accuracy. It again uses a non-locally declared integer m indicating the suffix of the term of the series to be transformed. The inputs to this procedure are the real procedure *term* as before, and an integer *available storage* indicating how long the vector $\boldsymbol{l}$ may become without exceeding the storage capacity available. The outputs from the procedure are the real result and a boolean variable which indicates whether the available storage has been exceeded. The procedure is as follows:

```
procedure epsilon algorithm (result, term, relative accuracy,
                                        available storage, storage not exceeded);
value relative accuracy, available storage;
integer available storage;
real result, relative accuracy, term;
boolean storage not exceeded;
begin integer s; real aux0, aux1, aux2;
array l[0: available storage];
l[0] := 0.0; m := 0;
EPSALG: aux0 := term; aux1 := aux0+l[0]; s := 0;
EPSLOOP: aux0 := 1.0/(if s = 0 then aux0 else aux1−l[s]);
```

```
if s ≠ 0 then
begin aux0 := aux0 + l[s − 1]; l[s − 1] := aux2
end;
        aux2 := aux1; aux1 := aux0; s := s + 1;
        if m ≧ s then goto EPSLOOP;
        l[m] := aux2; l[m + 1] := aux1;
        if m > 0 then
        begin if ⌝even(m) then aux2 := l[m − 1] else
                begin aux1 := aux2; aux2 := l[m − 2]
                end;
                aux2 := aux2/aux1;
                if abs(1.0 − aux2) < relative accuracy then
                begin storage not exceeded := true;
                        result := aux1; goto END
                end
        end
        m := m + 1; if m < available storage then goto EPSALG
        else storage not exceeded := false;
        END:
end;
```

Both procedures make use of the

```
boolean procedure even(integer);
value integer; integer integer;
even := (integer = (integer ÷ 2) × 2);
```

which has the value **true** if the input integer is even, and **false** if it is not.

An Example.

In order to test these procedures on another occasion and to clarify matters on this, two complete programmes are given illustrating the application of the ε-algorithm to the series

$$\sum_{m=0}^{\infty} (-1)^m/(m+1) = ln(2) \doteq 0.69314 .$$

The terms of this series are computed by means of the

```
real procedure ln2term;
        ln2term := (if even (m) then 1.0 else −1.0)/(m + 1);
```

The application of the ε-algorithm to the above series is displayed by means of the following complete programme

```
begin integer m;
        comment This comment must be replaced by the procedures
```

```
        display epsilon algorithm, even and ln2term;
        display epsilon algorithm (5, ln2term, 6)
end
```

It produces the following array:

+1.00000	+0.66667		
0.50000	.70000	+0.69231	
.83333	.69047	.69333	+0.69312
.58333	.69444	+0.69309	
.78333	+0.69242		
+0.61667			

The computation of $\ln(2)$ to a relative accuracy of 10^{-9} by means of application of the ε-algorithm to the above series, assuming that there is room in the computer store for a further 2000 real numbers, is as follows:

```
begin integer m; real check; boolean successful;
        comment This comment must be replaced by the procedures
        epsilon algorithm, even and ln2term;
        epsilon algorithm (check, ln2term, +1.0_10 - 9, 2000, successful);
        if successful then print (check)
end
```

A Single Point of Instability.

We now discuss how the singular rules which have been derived for the forward form of the ε-algorithm may be placed within the framework of the programmes which have been described above. We recall that if cancellation occurs in the subtraction $\varepsilon_{s-2}^{(m+1)} - \varepsilon_{s-2}^{(m+2)}$ then the singular rules affect quantities occurring in the following large lozenge:

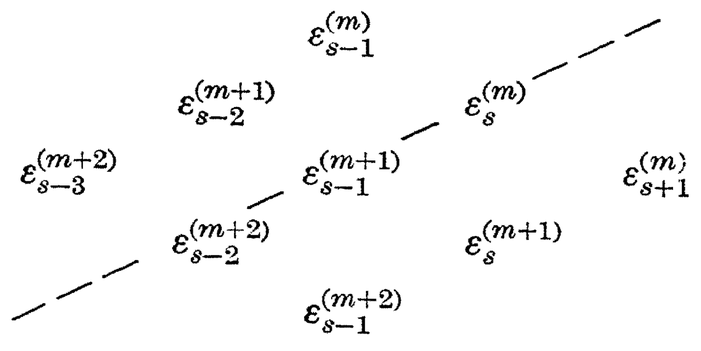


Diagram II

If

$$A = \varepsilon_{s-3}^{(m+2)}(1 - \varepsilon_{s-3}^{(m+2)}\varepsilon_{s-1}^{(m+1)^{-1}})^{-1} \tag{40}$$

$$B = \varepsilon_{s-1}^{(m)}(1 - \varepsilon_{s}^{(m)}\varepsilon_{s-1}^{(m+1)^{-1}})^{-1} \tag{41}$$

$$D = \varepsilon_{s-1}^{(m+2)}(1-\varepsilon_{s-1}^{(m+2)}\varepsilon_{s-1}^{(m+1)^{-1}})^{-1} \tag{42}$$

and

$$a = D+B-A \tag{43}$$

then

$$\varepsilon_{s+1}^{(m)} = C = a(1+a\varepsilon_{s-1}^{(m+1)^{-1}})^{-1} \tag{44}$$

We may assert that cancellation takes place if, in the subtraction $\varepsilon_{s-2}^{(m+2)}-\varepsilon_{s-2}^{(m+1)}$, a loss of f decimal figures takes place. Thus for the singular ε-algorithm procedure we must provide as input a real parameter (called *permissible degree of cancellation* in the following procedures) equal in magnitude to 10^{-f}.

Suppose that we are in the process of computing quantities lying on the backward diagonal through $\varepsilon_{s-2}^{(m+2)}$, $\varepsilon_{s-1}^{(m+1)}$, $\varepsilon_s^{(m)}$ and discover that cancellation occurs in the formation of $\varepsilon_{s-2}^{(m+2)}-\varepsilon_{s-2}^{(m+1)}$, (i.e. within the definition just made, that $|(\varepsilon_{s-2}^{(m+2)}-\varepsilon_{s-2}^{(m+1)})/\varepsilon_{s-2}^{(m+1)}| <$ *permissible degree of cancellation*). Immediately we note the current value of s ($s1$ say) and thus (it will later be apparent that we do not need to record the current value of m) have a record of the position in the ε-array of the point of cancellation. We know that we are about to compute A from (40), and indeed we compute and store it. We know that we are about to compute B, and after the normal administration of the ε-algorithm programme we do indeed compute and store B. We now proceed to the end of the vector l and start again at the other end. When the current value of s is equal to $s1+1$ then we know that we are about to compute D. This we do. We now know that we shall compute the next value of the ε-array, not by means of the normal ε-algorithm relationships, as has hitherto always been the case, but by means of the singular relationship (44). This is done, and the ill-effects of cancellation have been overcome.

Multiple Points of Instability.

So much for the case in which cancellation takes place at one point only. We should like, however, to apply the above procedure in the case in which there are a number of points in the ε-array at which cancellation takes place. We could, a posteriori at least, by scanning the ε-array backward diagonal by backward diagonal, construct a list of such points of cancellation.

As we scan the ε-array, backward diagonal by backward diagonal, we have as it were a current list of singular points which have not yet completely been dealt with. When the backward diagonal l meets a large lozenge of Diagram II for the second time, i.e. a new point of cancellation is encountered, the new singular point takes its place at the end of the current list; when the backward diagonal l leaves a large lozenge

for the third time, i.e. a new singular value of ε has been computed, the old singular point is removed from the beginning of the current list.

The integer $s1$ and the real quantities A and B of the preceding paragraphs become members of arrays. In the terminology of the ALGOL report, these arrays are dynamic own arrays. Indeed they are dynamic even unto the point of vanishing: this means that they are declared in a block, access to which is dependent upon the values of a logical expression (*local instability present* ∨ *new point of instability*).

As soon as we have discovered that cancellation takes place in the formation of $\varepsilon_{s-2}^{(m+2)} - \varepsilon_{s-2}^{(m+1)}$ we examine whether, first local instability is present, and secondly if the current value of s is the same as that stored in the $s1$-array at the position corresponding to the beginning of the list. If this is so we know immediately that $\varepsilon_{s-2}^{(m)}$ and $\varepsilon_{s-2}^{(m+1)}$ are approximately equal; but the singular rules given in this paper relate only to isolated points of cancellation so that we know immediately that our programme is inadequate for the case in hand. In this case therefore we immediately leave the procedure with the value of a boolean variable called *non isolated singularity* put equal to **true**.

If we are not in the presence of a non-isolated singularity however we increase the numerical value of the *end of list* by unity.

We now proceed as outlined earlier, store the current value of s in the array $s1$, the relevant values of the real quantities given by expressions (40) and (41) in the arrays A and B.

We continue the normal ε-algorithm process until the current value of s is equal to that ocurring in the $s1$ array at the position corresponding to the *begin of list*. We now know that we are about to compute D. This we do, and shortly after compute a new value of ε by means the singular rule.

Having done this we increase the numerical value of *begin of list* by unity. If now the *begin of list* exceeds the end of list we know that there is no *local instability present* (i.e. this boolean variable takes the value false). This means that we no longer have to compare the current value of s with some member of the $s1$ array and thus, apart from having to test for cancellation at each stage, the singular procedure may be made to run along the same lines as the normal ε-algorithm procedure.

There are two further details to be mentioned. The first is that formulae (40) to (44) are only usable on the assumption that the magnitude of $\varepsilon_{s-1}^{(m+1)}$ is greater than that of $\varepsilon_{s-1}^{(m)}$, $\varepsilon_{s-1}^{(m+2)}$ and $\varepsilon_{s-3}^{(m+2)}$. However $\varepsilon_{s-1}^{(m)}$ could only be large if $\varepsilon_{s-2}^{(m)}$ and $\varepsilon_{s-2}^{(m+1)}$ are approximately equal; this implies the presence of a non-isolated singularity which is dealt with by the procedure; similarly for $\varepsilon_{s-1}^{(m+2)}$. The quantity $\varepsilon_{s-3}^{(m+2)}$ can only be large if $\varepsilon_{s-4}^{(m+2)}$

and $\varepsilon_{s-4}^{(m+3)}$ are approximately equal, which corresponds to the application of a singular rule at a point displaced two columns to the left in the ε-array; this will already have been dealt with by the procedure.

The second detail is this: In the notation of Diagram 1 if the value of $m+1$ is equal to zero then the possible cancellation in the formation of $\varepsilon_{s-2}^{(m+2)} - \varepsilon_{s-2}^{(m+1)}$ need not be considered, since $\varepsilon_{s-2}^{(m)}$ lies outside the array.

A Segment of Programme.

We now summarize the reasoning of the preceding paragraphs in the form of a segment of an ALGOL programme. As a preliminary there are two details. In the ε-algorithm procedures given above, certain special operations were carried out if the value of s was non-zero; in the modified procedures further special operations are carried out if s is non-zero, thus a new boolean variable (*s non zero*) is introduced. Secondly, the box *aux*0 is used for two purposes: as before for containing the new value $\varepsilon_{s+1}^{(m)}$, and for containing the difference $\varepsilon_s^{(m+1)} - \varepsilon_s^{(m)}$ (i.e. $aux1 - l[s]$ or *term*).

In the above procedures the inner ε-algorithm loop contained the following assignments

*aux*0 := 1.0/(**if** $s = 0$ **then** *aux*0 **else** $aux1 - l[s]$);
if $s = 0$ **then**
begin $aux0 := aux0 + l[s-1]$; $l[s-1] := aux2$
end

In the singular ε-algorithm procedures we introduce the

procedure *normal aux* 0;
*aux*0 := 1.0/*aux*0 + (**if** *s nonzero* **then** $l[s-1]$ **else** 0.0);
and replace the above assignments by the following segment:
s non zero := $(s \neq 0)$; **if** *s non zero* **then** $aux0 := aux1 - l[s]$;
if $s < m \wedge \neg C$ *about to be computed*
$\wedge$ abs(*aux*0/$l[s]$) < *permissible degree of cancellation* **then**
begin *new point of instability* := **true**;
end of list := *end of list* + 1
end;
if *local instability present* $\vee$ *new point of instability* **then**
begin real a; **own real** D;
own boolean A *about to be computed*, B *about to be computed*;
own integer array $s1$ [*begin of list*: *end of list*];
own real array A, B[*begin of list*: *end of list*];
if *new point of instability* **then**

```
begin if local instability present then
        begin if s = s1[begin of list] then
                begin non isolated singularity := true;
                        goto END
                end
        end;
        local instability present := A about to be computed := true;
        new point of instability := false;
        s1[end of list] := s
end;
if C about to be computed then
begin a := D+B[begin of list]-A[begin of list];
        aux0 := a/(1.0+a/l[s-1]);
        C about to be computed := false;
        begin of list := begin of list+1;
        if begin of list > end of list then
        local instability present := false
end
else
normal aux0;
if local instability present then
        begin if A about to be computed then
                begin A[end of list] :=
                        if s non zero then l[s-1]/(1.0-l[s-1]/aux0)
                                        else 0.0;
                        A about to be computed := false;
                        B about to be computed := true
                end
                else
                if B about to be computed then
                begin B[end of list] := l[s]/(1.0-l[s]/aux1);
                        B about to be computed := false
                end
                else
                if s non zero ∧ s = s1[begin of list]+1 then
                begin D := aux1/(1.0-aux1/l[s]);
                                C about to be computed := true
                end
        end
end
else
normal aux0;
if s non zero then l[s-1] := aux2;
```

N.B. It is assumed that the computer operates perfectly naturally with numbers which are formally infinite; e.g. that the division of a finite real number by a formally infinite number produces zero, and that the machine does not stop because the capacity of the floating point representation has been exceeded.

Singular ε-Algorithm Procedures.

Finally we give two singular versions of the ε-algorithm procedures given earlier. The first, which displays the resultant even column ε-array, is:

```
procedure display singular epsilon algorithm (mmax, term, col, permissible
degree of cancellation, non isolated singularity);
value mmax, col, permissible degree of cancellation;
integer mmax, col; real term, permissible degree of cancellation;
boolean non isolated singularity;
begin integer s, sanfang, twommax, begin of list, end of list;
      real aux0, aux1, aux2;
      boolean local instability present, s non zero, new point of instability,
      C about to be computed;
      array l[0: mmax+1], display[1: ((mmax+1)×(mmax+5))÷4];
      procedure normal aux0;
      aux0 := 1.0/aux0+(if s non zero then l[s−1] else 0.0);
      non isolated singularity := local instability present :=
      new point of instability := C about to be computed := false;
      begin of list := 1; end of list := m := 0;
      l[0] := 0.0; twommax := 2×mmax;
      for m := 0 step 1 until mmax do
      begin aux0 := term; aux1 := aux0+l[0];
            for s := 0 step 1 until m do
            begin comment This comment must be replaced by the
                  above segment of programme;
                  if even(s) then
                  display[((s×(twommax−s+2))÷4+m+1] := aux1;
                  aux2 := aux1; aux1 := aux0
            end;
            l[m] := aux2; l[m+1] := aux1;
            if ⌝even(m) then
            display[((m+1)×(twommax−m+5))÷4] := aux1
      end;
      for sanfang := 0 step 2×col until mmax+1 do
```

```
begin NLCR; for m := 1 step 1 until mmax + 1 − sanfang ÷ 2 do
        begin NLCR; for s := sanfang step 2 until
                                    sanfang + 2 × (col − 1) do
            begin if (s ÷ 2 ≦ m) ∧ (m ≦ mmax + 1 − (s ÷ 2)) then
                                    print(display[(s × (twommax + 4 − s)) ÷ 4 + m])
            end
        end
    end;
    END:
end;
```

The second, which is to be used for computing the transformed sum of a slowly convergent series is as follows:

```
procedure singular epsilon algorithm (result, term, relative accuracy,
available storage, storage not exceeded, permissible degree of cancellation,
non isolated singularity);
value relative accuracy, available storage,
permissible degree of cancellation;
integer available storage;
real result, permissible degree of cancellation, relative accuracy, term;
boolean storage not exceeded, non isolated singularity;
    real aux0, aux1, aux2;
    boolean local instability present, s non zero,
    new point of instability, C about to be computed;
    array l[0: available storage];
    procedure normal aux0;
    aux0 := 1.0/aux0 + (if s non zero then l[s − 1] else 0.0);
    non isolated singularity := local instability present :=
    new point of instability := C about to be computed := false;
    begin of list := 1; end of list := m := 0; l[0] := 0.0;
    EPSALG: aux0 := term; aux1 := aux0 + l[0]; s := 0;
    SUBTRACTION: comment This comment must be replaced by
    the above segment of programme;
    aux2 := aux1; aux1 := aux0;
    s := s + 1; if s ≦ m then goto SUBTRACTION;
    l[m] := aux2; l[m + 1] := aux1;
    if m > 0 then
    begin if ¬even(m) then aux2 := l[m − 1] else
        begin aux1 := aux2; aux2 := l[m − 2]
        end;
        aux2 := aux2/aux1;
```

```
        if abs(1.0−aux2) < relative accuracy then
        begin storage not exceeded := true;
              result := aux1; goto END
        end;
        m := m+1; if m < available storage then goto EPSALG
        else storage not exceeded := false;
        END:
end;
```

An Example.

Use of these procedures may be illustrated by the application of the ε-algorithm to the exponential series. The terms in the latter series are computed by means of the

```
real procedure expterm(x); value x; real x;
begin own real u;
      expterm := u := (if m = 0 then 1.0 else x × u/m)
end;
```

A complete programme, which displays the application of the singular ε-algorithm to the exponential series is as follows:

```
begin integer m, n1; real x, t1; boolean something wrong;
      comment This comment must be replaced by the procedures
      display singular epsilon algorithm, even and expterm;
      n1 := read; t1 := read; x := read;
      print(n1); print(t1); print(x);
      display singular algorithm
      (n1, expterm(x), 6, t1, something wrong);
      if something wrong then print (1)
end
```

With $n1 = 3$, $x = 2$ and values of the other parameters taken to suit the reader's machine hardware, the entries in the even order columns of Table I may be reproduced.

A similar programme for the computation of $\exp(x)$ by application of the ε-algorithm may be constructed.

The reader may care to construct programmes for the application of the ε-algorithm to the exponential series in which no provision has been made for effecting singular rules, and to examine the results produced.

Extension to Further Algorithms.

It will be appreciated that the above strategic concepts may be brought to bear when programming singular versions of all the algorithms of this paper; the assignment statements may well be a little more complicated but the general design of the programmes will be the same.

Modification for Translators not Dealing with Own Variables.

Since many ALGOL translators do not include the facility of dynamic own arrays (or indeed own variables of any kind) it is in order to indicate how the above singular ε-algorithm procedures must be modified for use with these limited translators.

The real variables a and D must be declared together with $aux0$, $aux1$ and $aux2$. The arrays $s1$, A and B must be declared at the same place in the programme as the array l. With regard to the dimensions of the arrays $s1$, A and B, a restriction must be introduced. We provide as an input parameter to the singular ε-algorithm procedures an integer (*number of cases*) indicating the length of the current list of points of local instability as described earlier. The arrays $s1$, A and B are declared as for example:

integer array $s1[1:$ *number of cases*$]$;

Nevertheless these arrays are used as if they are cyclic in form with the addressing proceeding in the order $1, 2, \ldots,$ *number of cases* -1, *number of cases*, $1, 2, \ldots$. This method of addressing is achieved by modulus (*number of cases*) integer arithmetic, carried out by the following

integer procedure *dynamic* (*address*); **value** *address*;
integer *address*;
dynamic := *address* $-$ ((*address* $-$ 1) $\div$ *number of cases*) $\times$ *number of cases*;

A reference to $s1$[*begin of list*], for example, is now replaced by a reference to $s1$[*begin marker*], where

begin marker := *dynamic* (*begin of list*);

After the emergence of a new point of local instability the value of end of list is increased by unity, as before. If *end of list* $-$ *begin of list* $=$ *number of cases*, then we know that the number of singular points with which we are currently dealing exceeds the number allowed for. In this case we immediately leave the procedure with the value of a boolean variable called *too many points of instability* (which must also be included in the parameter list of the singular ε-algorithm procedure) put equal to **true**.

Conclusion.

The existence of singular cases considered in this note would be interpreted, when effecting the algorithm, as chronic instability. The results of this note show how this may be overcome. It is possible to extend the method to deal with singular cases of a more involved type, for example as would occur in the ε-algorithm when $\varepsilon_s^{(m)} = \varepsilon_s^{(m+1)} = \varepsilon_s^{(m+2)}$.

It may occur however that near singular cases are distributed uniformly throughout the application of the algorithm, and this sort of instability, which indicates some organic property of the example to which the algorithm is being applied, cannot be overcome by the methods described.

Acknowledgement.

The ALGOL procedures of this paper have been tested on an ALGOL translator for the $X1$ constructed by J. Nederkoorn and J. J. van de Laarschot.

REFERENCES

1. Wynn, P., *On a Device for Computing the $e_m(S_n)$ Transformation*, MTAC 10, 91–96 (1956).
2. Wynn, P., *On a Procrustean Technique for the Numerical Transformation of Slowly Convergent Sequences and Series*, Proc. Cambr. Phil. Soc. 52, 665–671.
3. Rutishauser, H., *Der Quotienten-Differenzen-Algorithmus*, Birkhauser Verlag, Basel/Stuttgart 1957.
4. Bauer, F. L., *The g-Algorithm*, J. Soc. Indust. Appl. Math., Vol. 8, No 1, 1960, 1–17.

MATHEMATISCH CENTRUM
AMSTERDAM