MATHEMATICS OF COMPUTATION
VOLUME 39, NUMBER 159
JULY 1982, PAGES 133-145

Some New Convergence Acceleration Methods*

By Claude Brezinski

Abstract. The E-algorithm is a general extrapolation method which includes most of the sequence transformations actually known. Some new convergence acceleration methods are derived from the E-algorithm by applying the so-called θ-procedure. The algorithms thus obtained are studied. Some theoretical results are proved and numerical examples are given.

1. Introduction. The general idea for accelerating the convergence of a slowly convergent sequence is to transform it into another sequence which will, under certain assumptions, converge faster to the same limit. Among the most famous sequence transformations are linear summation processes, Richardson polynomial extrapolation, Aitken's Δ^2 process [1] and Shanks' transformation [15] (Wynn's ε-algorithm [17]). An extensive study of these methods can be found in [3] while numerical examples, applications and FORTRAN subroutines are given in [4].

Each of these methods usually works in some particular cases since a universal method accelerating the convergence of wide classes of sequences cannot exist [10]. However numerical examples [11], [16] and also some theoretical results [6], [8] reveal that the best sequence transformations (that is those working in most of the examples) seem to be Levin's u and v transforms [14] and the θ-algorithm [2]. The θ-algorithm is a method derived from the rule of the ε-algorithm by a procedure which will be recalled in the next section.

Recently a general sequence transformation as well as the corresponding recursive algorithm was obtained by Håvie [13] and Brezinski [5]. This transformation, called the E-transformation, includes most of the convergence acceleration methods actually known. Thus it seemed interesting to apply to the E-algorithm the same procedure which was used to derive the θ-algorithm from the ε-algorithm and to study the new convergence acceleration methods obtained in that way. It is the aim of this paper.

2. The Procedure θ. Let us first begin by some reminiscences about Shanks' transformation and the ε-algorithm.

Let (S_n) be a sequence of real or complex numbers. Shanks' transformation consists in transforming (S_n) into a set of sequences $\{(e_k(S_n))\}$ given by

$$e_k(S_n) = H_{k+1}(S_n)/H_k(\Delta^2 S_n),$$

Received November 25, 1980; revised October 13, 1981.
1980 *Mathematics Subject Classification.* Primary 65B05.
Key words and phrases. Convergence acceleration, sequence transformation, extrapolation.
* Work partly supported by NATO Research Grant 027-81.

0025-5718/82/0000-0076/$04.25

where

$$H_k(u_n) = \begin{vmatrix} u_n & u_{n+1} & \cdots & u_{n+k-1} \\ u_{n+1} & u_{n+2} & \cdots & u_{n+k} \\ \cdots & \cdots & \cdots & \cdots \\ u_{n+k-1} & u_{n+k} & \cdots & u_{n+2k-2} \end{vmatrix}$$

and $\Delta^2 S_n = S_{n+2} - 2S_{n+1} + S_n$.

The computation of the determinants involved in this transformation can be avoided by using Wynn's ε-algorithm [17]

$$\varepsilon_{-1}^{(n)} = 0, \quad \varepsilon_0^{(n)} = S_n, \qquad n = 0, 1, \ldots,$$

$$\varepsilon_{k+1}^{(n)} = \varepsilon_{k-1}^{(n+1)} + \left[\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}\right]^{-1}, \qquad n, k = 0, 1, \ldots.$$

Wynn proved that $\varepsilon_{2k}^{(n)} = e_k(S_n)$. Thus the only interesting quantities are those with an even lower index; quantities with an odd lower index are intermediate results.

Let us now explain how the θ-algorithm was derived from the ε-algorithm. We set $D_k^{(n)} = [\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}]^{-1}$. Thus

$$\varepsilon_{2k+2}^{(n)} = \varepsilon_{2k}^{(n+1)} + D_{2k+1}^{(n)}. \tag{1}$$

The sequence $(\varepsilon_{2k+2}^{(n)})_{n\geqslant 0}$ will be said to converge faster than the sequence $(\varepsilon_{2k}^{(n+1)})_{n\geqslant 0}$ if

$$\Delta\varepsilon_{2k+2}^{(n)} = o\left(\Delta\varepsilon_{2k}^{(n+1)}\right),$$

where Δ operates on the upper index n ($\Delta u_n = u_{n+1} - u_n$). Thus a necessary and sufficient condition for $(\varepsilon_{2k+2}^{(n)})_{n\geqslant 0}$ to converge faster than $(\varepsilon_{2k}^{(n+1)})_{n\geqslant 0}$ is that

$$\lim_{n\to\infty} \Delta D_{2k+1}^{(n)} / \Delta\varepsilon_{2k}^{(n+1)} = -1.$$

If this condition is not satisfied, then a parameter w_k can be introduced in the algorithm (i.e., $D_{2k+1}^{(n)}$ will be replaced by $w_k D_{2k+1}^{(n)}$). If w_k is chosen such that

$$w_k = -\lim_{n\to\infty} \Delta\varepsilon_{2k}^{(n+1)} / \Delta D_{2k+1}^{(n)},$$

then the convergence will be accelerated. This new algorithm is the so-called γ-algorithm. In practice, of course, this choice is difficult to operate since it involves the computation of a limit. Thus w_k will be replaced by

$$w_k^{(n)} = -\Delta\varepsilon_{2k}^{(n+1)} / \Delta D_{2k+1}^{(n)}.$$

This new algorithm is called the θ-algorithm and its rules are

$$\theta_{-1}^{(n)} = 0, \quad \theta_0^{(n)} = S_n, \qquad n = 0, 1, \ldots,$$

$$\theta_{2k+2}^{(n)} = \theta_{2k}^{(n+1)} - \frac{\Delta\theta_{2k}^{(n+1)}}{\Delta D_{2k+1}^{(n)}} D_{2k+1}^{(n)}, \qquad n, k = 0, 1, \ldots, \tag{2}$$

where

$$D_k^{(n)} = \left[\theta_k^{(n+1)} - \theta_k^{(n)}\right]^{-1}, \quad \theta_{2k+1}^{(n)} = \theta_{2k-1}^{(n+1)} + D_{2k}^{(n)}.$$

Thus the θ-algorithm is obtained from the ε-algorithm by replacing the rule (1) by the rule (2).

Let us now formalize this way of transforming the rule of an algorithm. Let S be the set of sequences of real or complex numbers, and let S' be the subset of S of

sequences (b_n) such that $\forall n, \Delta b_n \neq 0$. We consider the applications f, g, and h from $S \times S'$ into S defined by

$$f\colon ((a_n),(b_n)) \mapsto f((a_n),(b_n)) = \left(a_n - \frac{\Delta a_n}{\Delta b_n} b_n\right),$$

$$g\colon ((a_n),(b_n)) \mapsto g((a_n),(b_n)) = \left(-\frac{\Delta a_n}{\Delta b_n} b_n\right),$$

$$h\colon ((a_n),(b_n)) \mapsto h((a_n),(b_n)) = \left(-\frac{\Delta a_n}{\Delta b_n} b_{n+1}\right).$$

Obviously

$$f((a_n),(b_n)) = (a_n) + g((a_n),(b_n)) = (a_{n+1}) + h((a_n),(b_n)).$$

Imperfectly we shall denote by $f(a_n, b_n)$ the nth term of the sequence $f((a_n),(b_n))$ and the same for g and h.

If, for some n, $\Delta b_n = 0$, then the corresponding member of the sequence under consideration does not exist. However, as stated by Wynn [18]: "Although special conditions can be imposed upon the numbers (b_n) to obviate this breakdown, in the exposition of a general theory in which no conditions are imposed upon the initial sequence, the results stated concern numbers that can be constructed".

Let us now consider an algorithm of the form

$$c_n = a_n + b_n. \tag{3}$$

We shall say that the procedure θ has been applied to this algorithm if the rule (3) is replaced by the rule

$$\theta(c_n) = f(a_n, b_n). \tag{4}$$

This algorithm will be called the θ-type algorithm associated with the algorithm (3).

Thus the θ-algorithm (2) is obtained by applying the procedure θ to the ε-algorithm (1). It is the θ-type algorithm associated with the ε-algorithm.

The application of the procedure θ can also be interpreted as a linear extrapolation at zero from the points (b_n, a_n) and (b_{n+1}, a_{n+1}) [12].

3. θ-Type Algorithms Associated with the E-Algorithm. Let us begin this section with a few words on the E-algorithm which is a general extrapolation method generalizing the Richardson polynomial extrapolation scheme. The basic idea of this sequence transformation is to assume that the given sequence (S_n) to be extrapolated behaves like

$$S_n = S + a_1 g_1(n) + \cdots + a_k g_k(n) \quad \forall n, \tag{5}$$

where the g_i's are given sequences, and to take S as the extrapolated value of (S_n). S is computed by solving the system

$$S_{n+i} = S + a_1 g_1(n+i) + \cdots + a_k g_k(n+i), \qquad i = 0,\ldots,k. \tag{6}$$

If the sequence (S_n) does not have the exact form (5) then the value of S obtained by solving (6) depends on n and k, and we shall denote it by $E_k(S_n)$. It is easy to check that $E_k(S_n)$ is given by a ratio of two determinants. This sequence-to-sequence transformation has been called the E-transformation, and it includes most of the sequence transformations actually known. For example, the choice $g_i(n) = \Delta S_{n+i-1}$

leads to Shanks' transformation, $g_i(n) = x_n^i$ gives Richardson extrapolation process, and $g_i(n) = x_n^{i-1}\Delta S_n/g(n)$ Levin's transformations.

A recursive algorithm, the E-algorithm, exists to avoid the computation of the determinants involved in the E-transformation. The rules of this algorithm are the following

$$E_0^{(n)} = S_n, \quad g_{0,i}^{(n)} = g_i(n), \qquad n = 0, 1, \ldots, \; i = 1, 2, \ldots,$$

$$E_k^{(n)} = \frac{E_{k-1}^{(n)} g_{k-1,k}^{(n+1)} - E_{k-1}^{(n+1)} g_{k-1,k}^{(n)}}{g_{k-1,k}^{(n+1)} - g_{k-1,k}^{(n)}},$$

$$g_{k,i}^{(n)} = \frac{g_{k-1,i}^{(n)} g_{k-1,k}^{(n+1)} - g_{k-1,i}^{(n+1)} g_{k-1,k}^{(n)}}{g_{k-1,k}^{(n+1)} - g_{k-1,k}^{(n)}}, \qquad k \geqslant 1, n \geqslant 0, i \geqslant k+1.$$

It has been proved that [5], [13]

$$E_k^{(n)} = E_k(S_n).$$

Using the notations of the preceding section, this algorithm can also be written

$$E_k^{(n)} = f\left(E_{k-1}^{(n)}, g_{k-1,k}^{(n)}\right), \quad \text{main rule,}$$

$$g_{k,i}^{(n)} = f\left(g_{k-1,i}^{(n)}, g_{k-1,k}^{(n)}\right), \quad \text{auxiliary rule.}$$

If we set

$$D_k^{(n)} = g\left(E_{k-1}^{(n)}, g_{k-1,k}^{(n)}\right), \quad \overline{D}_k^{(n)} = h\left(E_{k-1}^{(n)}, g_{k-1,k}^{(n)}\right),$$

$$D_{k,i}^{(n)} = g\left(g_{k-1,i}^{(n)}, g_{k-1,k}^{(n)}\right), \quad \overline{D}_{k,i}^{(n)} = h\left(g_{k-1,i}^{(n)}, g_{k-1,k}^{(n)}\right),$$

then the algorithm becomes (form 1)

$$E_k^{(n)} = E_{k-1}^{(n)} + D_k^{(n)}, \qquad \text{main rule,} \tag{7}$$

$$g_{k,i}^{(n)} = g_{k-1,i}^{(n)} + D_{k,i}^{(n)}, \qquad i > k, \quad \text{auxiliary rule,} \tag{8}$$

or (form 2)

$$E_k^{(n)} = E_{k-1}^{(n+1)} + \overline{D}_k^{(n)}, \qquad \text{main rule,} \tag{9}$$

$$g_{k,i}^{(n)} = g_{k-1,i}^{(n+1)} + \overline{D}_{k,i}^{(n)}, \qquad i > k, \quad \text{auxiliary rule.} \tag{10}$$

New convergence acceleration methods can be obtained by applying the procedure θ to the main rule and/or the auxiliary rule of the first and second forms of the E-algorithm. Let us now describe these new methods.

The T-Algorithm. Obtained by applying the procedure θ to the main rule (7).

$$T_0^{(n)} = S_n, \quad g_{0,i}^{(n)} = g_i(n), \quad T_k^{(n)} = f\left(T_{k-1}^{(n)}, D_k^{(n)}\right),$$

with

$$D_k^{(n)} = g\left(T_{k-1}^{(n)}, g_{k-1,k}^{(n)}\right), \quad g_{k,i}^{(n)} = f\left(g_{k-1,i}^{(n)}, g_{k-1,k}^{(n)}\right), \qquad i > k.$$

The t-Algorithm. Obtained by applying the procedure θ to the auxiliary rule (8).

$$t_0^{(n)} = S_n, \quad g_{0,i}^{(n)} = g_i(n), \quad t_k^{(n)} = f\left(t_{k-1}^{(n)}, g_{k-1,k}^{(n)}\right),$$

with

$$g_{k,i}^{(n)} = f\left(g_{k-1,i}^{(n+1)}, D_{k,i}^{(n)}\right), \qquad i > k,$$

$$D_{k,i}^{(n)} = h\left(g_{k-1,i}^{(n)}, g_{k-1,k}^{(n)}\right), \quad i > k.$$

The τ-Algorithm. Obtained by applying the procedure θ to the main rule (7) and to the auxiliary rule (8).

$$\tau_0^{(n)} = S_n, \quad g_{0,i}^{(n)} = g_i(n), \quad \tau_k^{(n)} = f\left(\tau_{k-1}^{(n)}, D_k^{(n)}\right),$$

with

$$\begin{aligned} D_k^{(n)} &= g\left(\tau_{k-1}^{(n)}, g_{k-1,k}^{(n)}\right), \\ g_{k,i}^{(n)} &= f\left(g_{k-1,i}^{(n)}, D_{k,i}^{(n)}\right), \qquad i > k, \\ D_{k,i}^{(n)} &= g\left(g_{k-1,i}^{(n)}, g_{k-1,k}^{(n)}\right), \qquad i > k. \end{aligned}$$

The S-Algorithm. Obtained by applying the procedure θ to the main rule (9).

$$S_0^{(n)} = S_n, \quad g_{0,i}^{(n)} = g_i(n), \quad S_k^{(n)} = f\left(S_{k-1}^{(n+1)}, D_k^{(n)}\right),$$

with

$$D_k^{(n)} = h\left(S_{k-1}^{(n)}, g_{k-1,k}^{(n)}\right), \quad g_{k,i}^{(n)} = f\left(g_{k-1,i}^{(n)}, g_{k-1,k}^{(n)}\right), \qquad i > k.$$

The s-Algorithm. Obtained by applying the procedure θ to the auxiliary rule (10).

$$s_0^{(n)} = S_n, \quad g_{0,i}^{(n)} = g_i(n), \quad s_k^{(n)} = f\left(s_{k-1}^{(n)}, g_{k-1,k}^{(n)}\right),$$

with

$$\begin{aligned} g_{k,i}^{(n)} &= f\left(g_{k-1,i}^{(n+1)}, D_{k,i}^{(n)}\right), \qquad i > k, \\ D_{k,i}^{(n)} &= h\left(g_{k-1,i}^{(n)}, g_{k-1,k}^{(n)}\right), \quad i > k. \end{aligned}$$

The σ-Algorithm. Obtained by applying the procedure θ to the main rule (9) and to the auxiliary rule (10).

$$\sigma_0^{(n)} = S_n, \quad g_{0,i}^{(n)} = g_i(n), \quad \sigma_k^{(n)} = f\left(\sigma_{k-1}^{(n+1)}, D_k^{(n)}\right),$$

with

$$\begin{aligned} D_k^{(n)} &= h\left(\sigma_{k-1}^{(n)}, g_{k-1,k}^{(n)}\right), \\ g_{k,i}^{(n)} &= f\left(g_{k-1,i}^{(n+1)}, D_{k,i}^{(n)}\right), \qquad i > k, \\ D_{k,i}^{(n)} &= h\left(g_{k-1,i}^{(n)}, g_{k-1,k}^{(n)}\right), \quad i > k. \end{aligned}$$

If we denote, respectively, by M_1 and M_2 the main rule of the first and second form of the E-algorithm and by A_1 and A_2 the auxiliary rules, then the preceding algorithms can be symbolically represented by

$$\begin{aligned} &T\text{-algorithm: } \theta(M_1) \vee A_1, \\ &t\text{-algorithm: } M_1 \vee \theta(A_1), \\ &\tau\text{-algorithm: } \theta(M_1) \vee \theta(A_1), \\ &S\text{-algorithm: } \theta(M_2) \vee A_2, \\ &s\text{-algorithm: } M_2 \vee \theta(A_2), \\ &\sigma\text{-algorithm: } \theta(M_2) \vee \theta(A_2). \end{aligned}$$

It is also possible to define and to study the following "crossing" algorithms: $\theta(M_1) \vee A_2$, $M_1 \vee \theta(A_2)$, $\theta(M_1) \vee \theta(A_2)$, $\theta(M_2) \vee A_1$, $M_2 \vee \theta(A_1)$, $\theta(M_2) \vee \theta(A_1)$.

As usual for convergence acceleration methods, in most of the cases the computational complexity of the preceding algorithms is much less than the complexity of computing the terms of the sequence to be transformed.

4. Properties. After defining these new algorithms, the first question which arises is to know which terms of the sequence (S_n) and of the sequences g_i are needed to implement these transformations. For the E-algorithm, this question was automatically solved by looking at the determinantal definition of the E-transformation. It is no more the case for the twelve preceding θ-type algorithms associated with the E-algorithm since, up to now, they are not yet defined by determinantal expressions; the only possible way of answering this question is to give a recursive proof by a direct look at the rules of the algorithms. One must first remark that the two rules of the E-algorithm have the form

$$c_n = a_n - \frac{\Delta a_n}{\Delta b_n} b_n = a_n + d_n \qquad \text{(form 1)},$$

$$c_n = a_{n+1} - \frac{\Delta a_n}{\Delta b_n} b_{n+1} = a_{n+1} + e_n \qquad \text{(form 2)}.$$

If we apply the procedure θ to these rules we get

$$\theta(c_n) = f(a_n, d_n) = a_n - \frac{\Delta a_n}{\Delta d_n} d_n \qquad \text{(form 1)},$$

$$\theta(c_n) = f(a_{n+1}, e_n) = a_{n+1} - \frac{\Delta a_{n+1}}{\Delta e_n} e_n \qquad \text{(form 2)}.$$

Thus, in both cases, the computation of $\theta(c_n)$ requires knowledge of a_n, a_{n+1}, a_{n+2}, b_n, b_{n+1} and b_{n+2}.

It is now easy to establish the following result ($\leftarrow$ means "depend(s) on"):

Property 1. *In the T- and S-algorithms*

$$g_{k,i}^{(n)} \leftarrow \begin{cases} (g_i(n+j),\ j = 0,\ldots,k), \\ (g_m(n+j),\ m = 1,\ldots,k \text{ and } j = 0,\ldots,k), \end{cases}$$

$$D_k^{(n)} \leftarrow \begin{cases} (S_{n+j},\ j = 0,\ldots,2k-1), \\ (g_m(n+j),\ m = 1,\ldots,k \text{ and } j = 0,\ldots,2k-m), \end{cases}$$

$$T_k^{(n)} \text{ and } S_k^{(n)} \leftarrow \begin{cases} (S_{n+j},\ j = 0,\ldots,2k), \\ (g_m(n+j),\ m = 1,\ldots,k \text{ and } j = 0,\ldots,2k+1-m). \end{cases}$$

In the t- and s-algorithms

$$g_{k,i}^{(n)} \leftarrow \begin{cases} (g_i(n+j),\ j = 0,\ldots,2k), \\ (g_m(n+j),\ m = 1,\ldots,k \text{ and } j = 0,\ldots,2k), \end{cases}$$

$$D_{k,i}^{(n)} \leftarrow \begin{cases} (g_i(n+j),\ j = 0,\ldots,2k-1), \\ (g_m(n+j),\ m = 1,\ldots,k \text{ and } j = 0,\ldots,2k-1), \end{cases}$$

$$t_k^{(n)} \text{ and } s_k^{(n)} \leftarrow \begin{cases} (S_{n+j},\ j = 0,\ldots,k), \\ (g_m(n+j),\ m = 1,\ldots,k \text{ and } j = 0,\ldots,2k-1). \end{cases}$$

In the τ- and σ-algorithms

$$g_{k,i}^{(n)} \leftarrow \begin{cases} (g_i(n+j),\ j=0,\ldots,2k), \\ (g_m(n+j),\ m=1,\ldots,k \text{ and } j=0,\ldots,2k), \end{cases}$$

$$D_{k,i}^{(n)} \leftarrow \begin{cases} (g_i(n+j),\ j=0,\ldots,2k-1), \\ (g_m(n+j),\ m=1,\ldots,k \text{ and } j=0,\ldots,2k-1), \end{cases}$$

$$D_k^{(n)} \leftarrow \begin{cases} (S_{n+j},\ j=0,\ldots,2k-1), \\ (g_m(n+j),\ m=1,\ldots,k \text{ and } j=0,\ldots,2k-1), \end{cases}$$

$$\tau_k^{(n)} \text{ and } \sigma_k^{(n)} \leftarrow \begin{cases} (S_{n+j},\ j=0,\ldots,2k), \\ (g_m(n+j),\ m=1,\ldots,k \text{ and } j=0,\ldots,2k). \end{cases}$$

Similar results could be obtained for the six remaining algorithms.

The second property to be studied is called quasi-linearity.

Property 2. *Let* $\{T_k^{(n)}\}$ *be the results obtained by applying the T-algorithm to the sequences* (S_n) *and* $(g_i(n))$. *The application of the T-algorithm to the sequences* $(aS_n + b)$ *and* $(b_i g_i(n))$, *where* $a, b_i \neq 0$ *produces the results* $\{aT_k^{(n)} + b\}$. *The same property holds for the eleven other θ-type algorithms associated with the E-algorithm.*

Proof. It immediately follows from

$$f((aa_n + b),(cb_n)) = af((a_n),(b_n)) + (b),$$
$$g((aa_n + b),(cb_n)) = ag((a_n),(b_n)),$$
$$h((aa_n + b),(cb_n)) = ah((a_n),(b_n)),$$

where (b) is the constant sequence whose terms are all equal to b. □

In the algebraic theory of a sequence transformation the most important point is the study of its kernel, that is, the set of sequences which are transformed into a constant sequence. When the transformation is defined by a ratio of two determinants as is, for example, the case for Shanks' transformation and Richardson process, the study of the kernel is, in general, easier than when the transformation is only defined by a recursive algorithm. Thus, for the θ-algorithm it was only possible to find the kernel of the transformation θ_2: $(S_n) \to (\theta_2^{(n)})$ [6]. The same trouble arises with the θ-type algorithms for which only the kernel of the transformation f is known. In my opinion it will only be possible to progress in this direction if a determinantal definition of the transformation is found (if it exists).

THEOREM 1. *Let us assume that* $\forall n,\ b_n \neq 0$. *A necessary and sufficient condition that* $\forall n, f(a_n, b_n) = S$ *is that*

$$\exists c \in \mathbf{C}: \forall n,\ a_n = S + cb_n.$$

Proof. The condition is sufficient. We have

$$f(a_n, b_n) = (b_{n+1}a_n - b_n a_{n+1})/\Delta b_n.$$

If the condition holds, then

$$f(a_n, b_n) = (b_{n+1}(S + cb_n) - b_n(S + cb_{n+1}))/\Delta b_n = S\Delta b_n/\Delta b_n = S$$

since $\Delta b_n \neq 0$.

Let us now prove that the condition is necessary. We assume that $\forall n, f(a_n, b_n) = S$. This means that $\forall n$, $\Delta b_n \neq 0$ since otherwise the ratio would be infinite or indefinite. Thus we have

$$S\Delta b_n = b_{n+1}a_n - b_n a_{n+1} \quad \text{or} \quad b_{n+1}(a_n - S) = b_n(a_{n+1} - S).$$

Since $\forall n$, $b_n \neq 0$, then

$$(a_n - S)/b_n = (a_{n+1} - S)/b_{n+1}.$$

Therefore, $\exists c \in \mathbf{C}$: $\forall n$, $(a_n - S)/b_n = c$ and the result is proved. □

The set of such sequences is called the kernel of f.

Remark 1. The condition $\forall n$, $b_n \neq 0$ is not needed to prove the sufficiency, but it is obligatory for the necessary condition as showed by the following example:

$$\begin{aligned} a_0 &= S + ab_0, \\ a_1 &= S + ab_1, \quad b_1 \neq 0, \\ a_2 &= S, \quad b_2 = 0, \\ a_3 &= S + cb_3, \quad b_3 \neq 0, \quad c \neq a, \\ a_4 &= S + cb_4. \end{aligned}$$

Then $f(a_i, b_i) = S$ for $i = 0, \ldots, 3$.

Remark 2. Let us assume that the sequence (a_n) converges. The condition $\forall n$, $f(a_n, b_n) = S$ does not imply that S is the limit of (a_n) as showed by the counterexample:

Let (a_n) converge to $S' \neq S$ and satisfy $\forall n$, $a_n \neq S$ and $\Delta a_n \neq 0$. Then $a_n = S + b_n$ with $b_n = a_n - S$, and $\forall n$, $f(a_n, b_n) = S$. Thus a converging sequence can be transformed into a sequence converging to a different limit. We shall say that the transformation is not quasi-regular [7]. This is, in particular, true for the E-transformation.

Remark 3. We have $f(a_n, b_n) = \Delta(a_n/b_n)/\Delta(1/b_n)$. The result of Theorem 1 can be compared with the result given by Cordellier for the θ-algorithm [6].

Let us now apply the preceding theorem to the θ-type algorithms when $k = 1$. We must first notice that $\forall n$, $E_1^{(n)} = t_1^{(n)} = s_1^{(n)}$, $T_1^{(n)} = \tau_1^{(n)}$, $s_1^{(n)} = \sigma_1^{(n)}$. We shall denote, respectively, by $\mathfrak{N}_{E_1}$, $\mathfrak{N}_{T_1}$ and $\mathfrak{N}_{S_1}$ the kernels of the transformations E_1: $(S_n) \to (E_1^{(n)})$, T_1: $(S_n) \to (T_1^{(n)})$ and S_1: $(S_n) \to (S_1^{(n)})$.

THEOREM 2.

$$\mathfrak{N}_{E_1} \subset \mathfrak{N}_{T_1}, \quad \mathfrak{N}_{E_1} \subset \mathfrak{N}_{S_1}, \quad \mathfrak{N}_{T_1} \neq \mathfrak{N}_{S_1}.$$

Proof. From Theorem 1, $\mathfrak{N}_{E_1}$ is the set of sequences such that $\forall n$, $g_1(n) \neq 0$, $\Delta g_1(n) \neq 0$, and $S_n - S = cg_1(n)$. $\mathfrak{N}_{T_1}$ is the set of sequences such that $\forall n$, $D_1^{(n)} \neq 0$, $\Delta D_1^{(n)} \neq 0$, and $S_n - S = aD_1^{(n)}$ with $D_1^{(n)} = -(\Delta S_n/\Delta g_1(n))g_1(n)$. If $(S_n) \in \mathfrak{N}_{E_1}$, then $\Delta S_n = c\Delta g_1(n)$, and thus $\Delta S_n/\Delta g_1(n) = c$ since $\Delta g_1(n) \neq 0$. It follows that $D_1^{(n)} = -cg_1(n)$, and therefore $\forall n$, $D_1^{(n)} \neq 0$, $\Delta D_1^{(n)} \neq 0$, and $S_n - S = aD_1^{(n)}$ with $a = -1$, which shows that $(S_n) \in \mathfrak{N}_{T_1}$ and $\mathfrak{N}_{E_1} \subset \mathfrak{N}_{T_1}$. From the example

$$S_n = \lambda^n, \quad g_1(n) = \mu^n, \quad \mu \neq \lambda, \quad \mu, \lambda \neq 1,$$

for which $E_1^{(n)} = \lambda^n(\mu - \lambda)/(\mu - 1)$ and $T_1^{(n)} = 0$, we see that the inclusion is strict.

An analogous proof will show the second inclusion. The kernels of T_1 and S_1 are not identical as showed by the examples:

$S_n = 1/(n+1)$, $g_1(n) = n$ then $(S_n) \in \mathcal{K}_{S_1}$ and $(S_n) \notin \mathcal{K}_{T_1}$,

$S_n = 1/(n+1)$, $g_1(n) = n+2$ then $(S_n) \in \mathcal{K}_{T_1}$ and $(S_n) \notin \mathcal{K}_{S_1}$. □

We shall now study the convergence and convergence acceleration conditions for the θ-type algorithms in order to explain their superiority on the E-algorithm.

A sufficient condition that $\lim_{n\to\infty} E_k^{(n)} = S$ is that $\lim_{n\to\infty} E_{k-1}^{(n)} = S$ and that $\exists \alpha < 1 < \beta$: $\forall n > N$, $g_{k-1,k}^{(n+1)}/g_{k-1,k}^{(n)} \notin [\alpha, \beta]$ [5, Theorem 4]. The same condition insures the convergence of $(T_k^{(n)})$ when n goes to infinity by replacing $g_{k-1,k}^{(n)}$ by $D_k^{(n)}$. But

$$\frac{D_k^{(n+1)}}{D_k^{(n)}} = \frac{\Delta T_{k-1}^{(n+1)}}{\Delta T_{k-1}^{(n)}} \frac{g_{k-1,k}^{(n+1)}/g_{k-1,k}^{(n)} - 1}{g_{k-1,k}^{(n+2)}/g_{k-1,k}^{(n+1)} - 1},$$

and thus we get

THEOREM 3. *A sufficient condition that* $\lim_{n\to\infty} T_k^{(n)} = S$ *is that* $\lim_{n\to\infty} T_{k-1}^{(n)} = S$, $\lim_{n\to\infty} g_{k-1,k}^{(n+1)}/g_{k-1,k}^{(n)} = b_k \neq 1$ *and that* $\exists \alpha < 1 < \beta$: $\forall n > N$, $\Delta T_{k-1}^{(n+1)}/\Delta T_{k-1}^{(n)} \notin [\alpha, \beta]$.

It must be noticed that these conditions are stronger than the convergence conditions for the E-algorithm but that they are only sufficient conditions. Similar results can also be obtained for the other θ-type algorithms.

Let us now look at convergence acceleration conditions. Let us assume that $\lim_{n\to\infty} g_{k-1,k}^{(n+1)}/g_{k-1,k}^{(n)} = b_k \neq 1$. Then a sufficient condition for $(E_k^{(n)})$ to converge faster than $(E_{k-1}^{(n)})$ is that [5, Theorem 6]

$$\lim_{n\to\infty} \left(E_{k-1}^{(n+1)} - S\right)/\left(E_{k-1}^{(n)} - S\right) = b_k.$$

This is a very strong condition which is only satisfied, in practice, for restricted classes of sequences [5, Theorem 7] or for a very appropriate choice of the auxiliary sequences g_i. For the T-algorithm the situation is much more interesting as showed by

THEOREM 4. *If* $\lim_{n\to\infty} g_{k-1,k}^{(n+1)}/g_{k-1,k}^{(n)} = b_k \neq 0, 1$ *and if*

$$\lim_{n\to\infty} \left(T_{k-1}^{(n+1)} - S\right)/\left(T_{k-1}^{(n)} - S\right) = c_k \neq 0, 1,$$

then $T_k^{(n)} - S = o(T_{k-1}^{(n)} - S)$.

Proof. From the rule of the algorithm we have

$$\frac{T_k^{(n)} - S}{T_{k-1}^{(n)} - S} = 1 - \left(\frac{T_{k-1}^{(n+1)} - S}{T_{k-1}^{(n)} - S} - 1\right) \Bigg/ \left(\frac{\Delta T_{k-1}^{(n+1)}}{\Delta T_{k-1}^{(n)}} \frac{g_{k-1,k}^{(n+1)}}{g_{k-1,k}^{(n)}} \frac{\Delta g_{k-1,k}^{(n)}}{\Delta g_{k-1,k}^{(n+1)}} - 1\right).$$

If the conditions of the theorem are satisfied, then, by a classical result,

$$\lim_{n\to\infty} \Delta g_{k-1,k}^{(n+1)}/\Delta g_{k-1,k}^{(n)} = b_k \quad \text{and} \quad \lim_{n\to\infty} \Delta T_{k-1}^{(n+1)}/\Delta T_{k-1}^{(n)} = c_k,$$

and the result immediately follows. □

As we see, the condition $c_k = b_k$ is no longer necessary as it was for the E-algorithm.

Remark 4. Usually, in practice, it is difficult to check if the second condition of the theorem is satisfied because it involves the unknown limit S. Using a result proved by Delahaye [9], we know that if

$$\lim_{n\to\infty} \Delta T_{k-1}^{(n+1)}/\Delta T_{k-1}^{(n)} = c_k \neq 0, -1, +1,$$

then

$$\lim_{n\to\infty} \left(T_{k-1}^{(n+1)} - S\right)/\left(T_{k-1}^{(n)} - S\right) = c_k,$$

and consequently we shall have $T_k^{(n)} - S = o(T_{k-1}^{(n)} - S)$.

As for Theorem 3, the conditions of Theorem 4 are only sufficient. Similar results can also be obtained for the other θ-type algorithms.

Let us give two examples to illustrate the preceding theorem:

$$S_n = n\lambda^n, \quad g_1(n) = \mu^n \quad \text{with } \lambda\mu \neq 0, |\lambda| < 1, |\mu| < 1 \text{ and } \lambda \neq \mu.$$

We have $g_1(n+1)/g_1(n) = \mu$ and $\lim_{n\to\infty} S_{n+1}/S_n = \lambda$. Thus the sequences $(E_1^{(n)})$ and $(T_1^{(n)})$ tend to zero. Since $\lambda \neq \mu$, $(E_1^{(n)})$ does not converge faster than (S_n):

$$E_1^{(n)} = \lambda^n(n(\mu - \lambda) - \lambda)/(\mu - 1), \quad \lim_{n\to\infty} E_1^{(n)}/S_n = (\mu - \lambda)/(\mu - 1).$$

The conditions of Theorem 4 are satisfied, and $(T_1^{(n)})$ converges faster than (S_n):

$$T_1^{(n)} = \lambda^{n+2}/\left(2\lambda(\lambda - 1) - n(1 - \lambda)^2\right).$$

$(T_1^{(n)})$ does not depend on μ. It can be said that the T_1-transformation corrects, in some sense, the bad choice which was made for the auxiliary sequence g_1 (a good choice is, of course, $g_1(n) = \lambda^n$).

If we now consider $S_n = 1/(n+1)$ with $g_1(n) = n + 2$, then $E_1^{(n)} = 2/(n+1)$ and $T_1^{(n)} = 0$. For this example the conditions of Theorem 4 are not satisfied, which shows that they are only sufficient.

5. Recursive Use of the Procedure θ. In Section 2 we saw that the procedure θ consists in replacing an algorithm of the form $c_n = a_n + b_n$ by an algorithm of the form

$$\theta(c_n) = f(a_n, b_n).$$

This new rule can be written as

$$\theta(c_n) = a_n + (f(a_n, b_n) - a_n),$$

and thus we can again apply the procedure θ to this rule to get

$$\theta^2(c_n) = \theta(\theta(c_n)) = f(a_n, f(a_n, b_n) - a_n) = f(a_n, \theta(c_n) - a_n).$$

This recursive use of the procedure θ can be continued, and we finally obtain the following algorithm

$$\theta^0(c_n) = f^0(a_n, b_n) = a_n + b_n,$$

$$\theta^k(c_n) = f^k(a_n, b_n) = f\left(a_n, f^{k-1}(a_n, b_n) - a_n\right)$$

$$= f\left(a_n, \theta^{k-1}(c_n) - a_n\right), \qquad k = 1, 2, \ldots.$$

The kth iterated application of the procedure θ consists in replacing the rule $c_n = a_n + b_n$ by $\theta^k(c_n) = f^k(a_n, b_n)$.

Heuristically this recursive use of the procedure θ is justified by the fact that if $\theta^{k-1}(c_n)$ is a good approximation of the limit S of the sequence (a_n), then $\theta^k(c_n)$ might be a better approximation of S. Hence if $b_n = S - a_n$, then $\forall n, f(a_n, S - a_n) = S$ if $\forall n, \Delta a_n \neq 0$.

This recursive use of the procedure θ can be applied to both forms of the E-algorithm and to the twelve θ-type algorithms associated with it as well.

6. Numerical Examples. Some numerical examples have been carried out on the IRIS 80 computer of the University of Lille in double precision with approximately 16 exact decimal digits.

For the sequence $S_n = (0.95)^{n+1}/(n+1)$ the following results have been obtained with the choice $g_i(n) = \Delta S_{n+i-1}$ which leads to Shanks' transformation for the E-algorithm.

T-Algorithm.

$(T_0^{(n)})$	$(T_1^{(n)})$	$(T_2^{(n)})$	$(T_3^{(n)})$	$(T_4^{(n)})$
0.95000000				
0.45125000				
0.28579167	0.05877243			
0.20362656	0.01932949			
0.15475619	0.00412200	−0.01399516		
0.12251532	−0.00242632	−0.01086415		
0.09976247	−0.00523243	−0.00854903	0.00473557	
0.08292755	−0.00626256	−0.00715902	−0.00395274	
0.07002771	−0.00641662	−0.00646418	−0.00548366	−0.00602525

The results with the other algorithms are not better than the preceding ones. However it must be noticed that they are better than the results obtained by the E-algorithm, for which we get

E-Algorithm.

$(E_1^{(n)})$	$(E_2^{(n)})$	$(E_3^{(n)})$	$(E_4^{(n)})$
0.20365202			
0.12257430			
0.08302371	0.07015522		
0.06000750	0.04522404		
0.04519982	0.03113841	0.02778396	
0.03503673	0.02236704	0.01826045	
0.02773985	0.01655228	0.01254054	0.01147972

Let us now consider the sequence $S_n = (-0.95)^{n+1}/(n+1)$ with the same choice for the g_i's. We shall compare the sequences (S_{3k}), $(E_k^{(k)})$, $(T_k^{(k-1)})$, $(S_k^{(k-1)})$, $(t_k^{(1)})$, $(s_k^{(1)})$, $(\tau_k^{(0)})$, and $(\sigma_k^{(0)})$ because the computation of their respective terms requires knowledge of the initial sequence up to the same term S_{3k}.

k	(S_{3k})	$(E_k^{(k)})$	$(T_k^{(k-1)})$	$(S_k^{(k-1)})$
1	0.20362656	0.00832445	−0.00471336	−0.00684002
2	−0.09976247	−0.00006298	−0.00000083	0.00002024
3	0.05987369	0.00000046	0.00000002	−0.00000005
4	−0.03948785	$< 10^{-8}$	$< 10^{-8}$	$< 10^{-8}$
5	0.02750792	$< 10^{-8}$	$< 10^{-8}$	$< 10^{-8}$
k	$(t_k^{(1)})$	$(s_k^{(1)})$	$(\tau_k^{(0)})$	$(\sigma_k^{(0)})$
1	0.00832445	0.00832444	−0.00471336	−0.00684002
2	−0.00016970	−0.00028390	0.00001704	0.00001055
3	0.00002136	0.00002510	−0.00000008	−0.00000034
4	−0.00000313	−0.00000363	0.00000003	−0.00000003
5	0.00000118	0.00000052	$< 10^{-8}$	$< 10^{-8}$

Let us now consider the sequence $S_n = (n + 1)/(n + 2)$, which is a logarithmic sequence and, thus, is difficult to accelerate. We get the following results with the choice $g_i(n) = \Delta S_{n+i-1}$:

k	(S_{3k})	$(E_k^{(k)})$	$(T_k^{(k-1)})$	$(S_k^{(k-1)})$
1	0.80000000	0.87500000	0.94444444	1.00000000
2	0.87500000	0.94444444	0.99900000	1.00000000
3	0.90909091	0.96875000	1.00000256	1.00000000
4	0.92857143	0.98000000	0.99999998	1.00000000
5	0.94117647	0.98611111	0.99999999	1.00000000
k	$(t_k^{(1)})$	$(s_k^{(1)})$	$(\tau_k^{(0)})$	$(\sigma_k^{(0)})$
1	0.87500000	0.87500000	0.94444444	1.00000000
2	0.93819444	0.92604167	0.99672084	1.00000000
3	0.96490310	0.94590545	0.99973882	1.00000000
4	0.96804451	0.98868613	0.99983712	1.00000000
5	1.04429718	rounding errors	0.99999770	1.00000000

We must remark that, for the preceding sequence, the conditions of Theorem 1 hold with $c = -2$ and that, $\forall n$, $S_1^{(n)} = \sigma_1^{(n)} = 1$, as can be checked from the numerical results.

To exemplify the numerical instability of the preceding algorithms let us give the numerical results obtained when programming the s-algorithm in single precision for the sequence $S_n = (-0.95)^{n+1}/(n + 1)$. We get

$$
\begin{aligned}
s_1^{(1)} &= \ \ 0.00832444,\\
s_2^{(1)} &= -0.00028397,\\
s_3^{(1)} &= \ \ 0.00002587,\\
s_4^{(1)} &= \ \ 0.00122677,\\
s_5^{(1)} &= \ \ 0.01454507.
\end{aligned}
$$

Acknowledgements. I would like to thank Mrs. B. Delannoy for programming the numerical examples.

Université des Sciences et Techniques de Lille
U.E.R. I.E.E.A.-M3
59655 Villeneuve d'Ascq Cedex, France

1. A. C. AITKEN, "On Bernoulli's numerical solution of algebraic equations," *Proc. Roy. Soc. Edinburgh*, v. 46, 1926, pp. 289–305.

2. C. BREZINSKI, "Accélération de suites à convergence logarithmique," *C. R. Acad. Sci. Paris Ser. A*, v. 273, 1971, pp. 727–730.

3. C. BREZINSKI, *Accélération de la Convergence en Analyse Numérique*, Lecture Notes in Math., vol. 584, Springer-Verlag, Heidelberg, 1977.

4. C. BREZINSKI, *Algorithmes d'Accélération de la Convergence. Etude Numérique*, Editions Technip, Paris, 1978.

5. C. BREZINSKI, "A general extrapolation algorithm," *Numer. Math.*, v. 35, 1980, pp. 175–187.

6. F. CORDELLIER, "Caractérisation des suites que la première étape du θ-algorithme transforme en suites constantes," *C. R. Acad. Sci. Paris Ser. A*, v. 284, 1977, pp. 389–392.

7. F. CORDELLIER, "Sur la régularité des procédés δ^2 d'Aitken et W de Lubkin," in *Padé Approximation and Its Applications* (L. Wuytack, ed.), Lecture Notes in Math., vol. 765, Springer-Verlag, Heidelberg, 1979, pp. 20–35.

8. F. CORDELLIER, *Analyse Numérique des Transformations de Suites et de Séries*, Thesis, University of Lille. (To appear.)

9.. J. P. DELAHAYE, "Liens entre la suite du rapport des erreurs et celle du rapport des différences," *C. R. Acad. Sci. Paris Ser. A*, v. 290, 1980, pp. 343–346.

10. J. P. DELAHAYE & B. GERMAIN-BONNE, "Résultats négatifs en accélération de la convergence," *Numer. Math.*, v. 35, 1980, pp. 443–457.

11. C. ESPINOZA, *Application de l'ε-Algorithme à des Suites Non Scalaires et Comparaison de quelques Résultats Numériques Obtenus avec les ε, ρ et θ-Algorithmes*, Mémoire de DEA, University of Lille, 1974. (Unpublished manuscript.)

12. B. GERMAIN-BONNE, *Estimation de la Limite des Suites et Formalisation de Procédés d'Accélération de Convergence*, Thesis, University of Lille, 1978.

13.. T. HÅVIE, "Generalized Neville type extrapolation schemes," *BIT*, v. 19, 1979, pp. 204–213.

14. D. LEVIN, "Development of non-linear transformations for improving convergence of sequences," *Internat. J. Comput. Math. (B)*, v. 3, 1973, pp. 371–388.

15. D. SHANKS "Non linear transformations of divergent and slowly convergent sequences," *J. Math. Phys.*, v. 34, 1955, pp. 1–42.

16. D. A. SMITH & W. F. FORD, "Acceleration of linear and logarithmic convergence," *SIAM J. Numer. Anal.*, v. 16, 1979, pp. 223–240.

17. P. WYNN "On a device for computing the $e_m(S_n)$ transformation," *MTAC*, v. 10. 1956, pp. 91–96.

18. P. WYNN, "Hierarchies of arrays and function sequences associated with the epsilon algorithm and its first confluent form," *Rend. Mat.* (4), v. 5, 1972, pp. 819–852.