SIAM J. SCI. STAT. COMPUT.
Vol. 9, No. 5, September 1988

IMPLEMENTATION OF THE TOPOLOGICAL ε-ALGORITHM*

ROGER C. E. TAN†

Abstract. A recent survey paper of D. A. Smith, W. F. Ford, and A. Sidi [*SIAM Review*, 29 (1987), pp. 199–233] found the topological ε-algorithm (TEA) to compare very unfavourably with certain other methods for the acceleration of vector sequences. It is suggested that this poor performance of the TEA is due to an error in implementation. When this error is removed the method is found to perform at least as well as the other methods in most situations.

Key words. topological ε-algorithm, vector ε-algorithm, scalar ε-algorithm, vector sequences

AMS(MOS) subject classifications. 65B05, 65F10

1. Introduction. Smith, Ford, and Sidi [9] have published a valuable comparison of numerical algorithms for five extrapolation methods for accelerating the convergence of vector sequences. These five extrapolation methods may be classified into two families, the *polynomial methods* and the *epsilon algorithms.* They are exact methods for the sequence of vectors generated by $\mathbf{x}_{n+1} = F(\mathbf{x}_n)$ if F is a fixed-point linear generator and they usually converge quadratically if F is nonlinear. Unfortunately implementation in [9] of one of these algorithms, the topological ε-algorithm (TEA) of Brezinski [4], appears to contain some critical errors and this appears to have resulted in an incorrect conclusion being reached on the practical usefulness of this algorithm. It was reported in [9], that the TEA "has no practical application," "its primary role seems to be as a theoretical bridge between the two families of methods." Although stability and convergence of the TEA were established in [8], the TEA was compared very unfavourably in [8], [9] with the vector ε-algorithm (VEA) and the scalar ε-algorithm (SEA).

In this paper these errors are corrected and the TEA is shown to perform well on all the numerical examples considered in [9]. Moreover, the use of the TEA on certain sequences produced in numerical solution of Laplace's equation is also shown to be very effective. Several alternative (correct) implementations of the TEA are compared and an explicit program listing, using the package MATLAB [7], is also given.

In [9] the authors stated that they were not aware of any reported success with the TEA, but it has subsequently been implemented with complete success in [13], where it is used to improve dramatically the numerical performance of an iterative method for computing derivatives of eigenvalues and eigenvectors of parameter-dependent matrices. This problem arises in several applications in physics and engineering (see [1], [2], and [12] for references). Some alternative procedures for this problem were considered in [10]–[12] and in general, when appropriately implemented, the TEA performed at least as well as these other methods. The implementation in [13] appears to be the first successful implementation of the TEA.

2. The topological ε-algorithm. Four different versions of the TEA are given in [13]. Two of them involve the usual inner product $(\cdot, \cdot)$ which satisfies $(\alpha a, \beta b) = \alpha\bar{\beta}(a, b)$ for all $\alpha, \beta \in \mathbf{C}$, the other two involve the bilinear form $\langle \cdot, \cdot \rangle$ which satisfies

* Received by the editors January 21, 1987; accepted for publication (in revised form) January 6, 1988.

Editor's Note: Refer to the December issue of *SIAM Review*, 30 (1988), for corrections to the Smith, Ford, and Sidi article, "Extrapolation Methods for Vector Sequences," that was originally published in *SIAM Review*, 29 (1987), pp. 199–233.

† Mathematics Department, La Trobe University, Bundoora, Victoria 3083, Australia.

$\langle \alpha a, \beta b\rangle = \alpha\beta\langle a, b\rangle$ for all $\alpha, \beta \in \mathbf{C}$ while $\langle a, b\rangle = (a, b)$ whenever a and b are real vectors. We are interested in accelerating a given vector sequence $\{\mathbf{x}_n\}$ by generating other sequences $\{\varepsilon_{2m}^{(n)}\}$ with respect to m that will typically converge faster than the original sequence. Moreover $\{\varepsilon_{2m}^{(n)}\}$ may converge to a meaningful limit when $\{\mathbf{x}_n\}$ does not.

All algorithms begin with

$$\varepsilon_{-1}^{(n)} = 0, \quad \varepsilon_0^{(n)} = \mathbf{x}_n, \quad n = 0, 1, \cdots. \tag{1}$$

From this they generate the following sequences.

ALGORITHM 1.

$$\varepsilon_{2m+1}^{(n)} = \varepsilon_{2m-1}^{(n+1)} + \frac{\mathbf{y}}{\langle \mathbf{y}, \Delta\varepsilon_{2m}^{(n)}\rangle}, \quad \varepsilon_{2m+2}^{(n)} = \varepsilon_{2m}^{(n+1)} + \frac{\Delta\varepsilon_{2m}^{(n)}}{\langle \Delta\varepsilon_{2m+1}^{(n)}, \Delta\varepsilon_{2m}^{(n)}\rangle}, \qquad m, n = 0, 1, \cdots. \tag{2}$$

ALGORITHM 2.

$$\varepsilon_{2m+1}^{(n)} = \varepsilon_{2m-1}^{(n+1)} + \frac{\mathbf{y}}{\langle \mathbf{y}, \Delta\varepsilon_{2m}^{(n)}\rangle}, \quad \varepsilon_{2m+2}^{(n)} = \varepsilon_{2m}^{(n+1)} + \frac{\Delta\varepsilon_{2m}^{(n+1)}}{\langle \Delta\varepsilon_{2m+1}^{(n)}, \Delta\varepsilon_{2m}^{(n+1)}\rangle}, \qquad m, n = 0, 1, \cdots. \tag{3}$$

ALGORITHM 3.

$$\varepsilon_{2m+1}^{(n)} = \varepsilon_{2m-1}^{(n+1)} + \frac{\mathbf{y}}{(\mathbf{y}, \Delta\varepsilon_{2m}^{(n)})}, \quad \varepsilon_{2m+2}^{(n)} = \varepsilon_{2m}^{(n+1)} + \frac{\Delta\varepsilon_{2m}^{(n)}}{(\Delta\varepsilon_{2m}^{(n)}, \Delta\varepsilon_{2m+1}^{(n)})}, \qquad m, n = 0, 1, \cdots. \tag{4}$$

ALGORITHM 4.

$$\varepsilon_{2m+1}^{(n)} = \varepsilon_{2m-1}^{(n+1)} + \frac{\mathbf{y}}{(\mathbf{y}, \Delta\varepsilon_{2m}^{(n)})}, \quad \varepsilon_{2m+2}^{(n)} = \varepsilon_{2m}^{(n+1)} + \frac{\Delta\varepsilon_{2m}^{(n+1)}}{(\Delta\varepsilon_{2m}^{(n+1)}, \Delta\varepsilon_{2m+1}^{(n)})}, \qquad m, n = 0, 1, \cdots. \tag{5}$$

Here Δ denotes the forward difference operator acting on the superscript. Clearly, in the real case, Algorithms 1 and 3 are identical and Algorithms 2 and 4 are also identical. The analysis in [4] refers to Algorithms 1 and 2 but similar proofs yield similar results for Algorithms 3 and 4 (see [13]). The algorithm in [9] involves inner products and does not use the appropriate form for the complex case but, as the examples in [9] are real, this does not explain the failure of the TEA reported in [9]. The complex case is discussed in more detail in [13] and some of the numerical examples given there are complex.

Theorems 105 and 100 of [4] show that if Algorithm 1 is applied to a sequence $\{\mathbf{x}_n\}$ of vectors satisfying the recurrence relation $\sum_{i=0}^{k} a_i(\mathbf{x}_{n+i} - \mathbf{s}) = 0$ for all $n \in \mathbf{N}$, where $a_i \in \mathbf{C}$ and $\sum_{i=0}^{k} a_i \neq 0$, then, for any fixed arbitrary vector $\mathbf{y}$, such that all the indicated quantities exist, then

$$\varepsilon_{2k}^{(n)} = e_k(\mathbf{x}_n) = \mathbf{s},$$

and

$$\varepsilon_{2k+1}^{(n)} = [e_k(\Delta\mathbf{x}_n)]^{-1} = \frac{\mathbf{y}}{\langle \mathbf{y}, e_k(\Delta\mathbf{x}_n)\rangle},$$

where

$$e_k(\mathbf{x}_n) = \frac{\begin{vmatrix} \mathbf{x}_n & \mathbf{x}_{n+1} & \cdots & \mathbf{x}_{n+k} \\ \langle \mathbf{y}, \Delta\mathbf{x}_n \rangle & \langle \mathbf{y}, \Delta\mathbf{x}_{n+1} \rangle & \cdots & \langle \mathbf{y}, \Delta\mathbf{x}_{n+k} \rangle \\ \cdots & \cdots & \cdots & \cdots \\ \langle \mathbf{y}, \Delta\mathbf{x}_{n+k-1} \rangle & \cdots & \cdots & \langle \mathbf{y}, \Delta\mathbf{x}_{n+2k-1} \rangle \end{vmatrix}}{\begin{vmatrix} 1 & 1 & \cdots & 1 \\ \langle \mathbf{y}, \Delta\mathbf{x}_n \rangle & \langle \mathbf{y}, \Delta\mathbf{x}_{n+1} \rangle & \cdots & \langle \mathbf{y}, \Delta\mathbf{x}_{n+k} \rangle \\ \cdots & \cdots & \cdots & \cdots \\ \langle \mathbf{y}, \Delta\mathbf{x}_{n+k-1} \rangle & \cdots & \cdots & \langle \mathbf{y}, \Delta\mathbf{x}_{n+2k-1} \rangle \end{vmatrix}}. \tag{6}$$

The transformed sequence $e_k(\mathbf{x}_n)$, $k = 1, 2, \cdots$, plays an important role in the theory of the TEA. In the linear case where $F(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$ for some nondefective $N \times N$ matrix $\mathbf{A}$, it is well known that for a given $\mathbf{x}_0$, the vectors $\mathbf{x}_n$ generated by $\mathbf{x}_{n+1} = F(\mathbf{x}_n)$, $n = 0, 1, \cdots$, have error given by

$$\mathbf{x}_n - \mathbf{s} = \sum_{i=1}^{k} \gamma_i \lambda_i^n \mathbf{v}_i, \qquad n = 0, 1, \cdots, \tag{7}$$

where $\mathbf{s}$ is the exact solution of $\mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{b}$, γ_i are scalars, λ_i and $\mathbf{v}_i$ are the eigenvalues and corresponding eigenvectors of the matrix $\mathbf{A}$, and k is the number of distinct eigenvalues ($k \leqq N$). From this, it can be deduced by Theorem 2.2 of [11] that there exist scalars $c_0, c_1, \cdots, c_k$ such that

$$\sum_{i=0}^{k} c_i \mathbf{x}_{n+i} = \left(\sum_{i=0}^{k} c_i \right) \mathbf{s}. \tag{8}$$

Hence by Theorems 105 and 100 of [4], $\varepsilon_{2k}^{(n)} = e_k(\mathbf{x}_n) = \mathbf{s}$. Theorem 110 of [4] gives a similar result for Algorithm 2, with the vectors in the first row of the numerator of (6) replaced by $\mathbf{x}_{n+k}, \mathbf{x}_{n+k+1}, \cdots, \mathbf{x}_{n+2k}$.

In the scalar case ($N = 1$), the classical ε-algorithm (SEA) [15] yields the scalar sequence $e_k(x_n)$ (with $\mathbf{y} = 1$) in (6). If the errors of the scalar sequence satisfy $x_n - s = \sum_{i=1}^{k} \gamma_i \lambda_i^n$, then, $\varepsilon_{2k}^{(n)} = e_k(x_n) = s$ also and for this algorithm convergence has been proved in quite general circumstances [16]. Convergence of the TEA in the linear case is established in [8]. Although in the vector case the theory of the TEA for nonlinear F is less well developed, the error analysis may be treated similarly to the VEA in [9]. However, Wynn [15] originally developed the classical ε-algorithm as an efficient means for computing the $e_k(x_n)$ when $N = 1$ and Brezinski [3]–[5] followed along similar lines in developing the TEA as an efficient means for computing the $e_k(\mathbf{x}_n)$ in the general vector case by an appropriate interpretation of "inverse" of a vector with his definition for inverse of an ordered pair. Wynn, on the other hand, defined the inverse of a vector as the Samelson inverse. Although both these extensions yield $\varepsilon_{2k}^{(n)} = e_k(\mathbf{x}_n) = \mathbf{s}$, if $\{\mathbf{x}_n\}$ satisfies the recurrence relation in (8), the case where c_i in (8) are complex numbers has only recently been proved by Graves-Morris [6]. It appears to be unsuccessful implementation of the TEA that lead the authors in [9] to claim that the VEA is "a much more successful extension" of the SEA, and "the most useful member" of the *epsilon* family, although the results for the TEA are valid for all $c_i \in \mathbf{C}$ [4, p. 173]. Because of the central theoretical role of $e_k(\mathbf{x}_n)$, any implementation of the TEA which fails to compute the $e_k(\mathbf{x}_n)$ must be suspect. It is shown below that the implementation of the TEA given in [9] does not in fact compute the $e_k(\mathbf{x}_n)$ correctly, although the four algorithms given here do. This appears to be why the four

algorithms given here all gave excellent numerical results, whereas that given in [9] failed even to achieve convergence.

In [9] the TEA is given as (1) and

$$(9)\quad \varepsilon_{2m+1}^{(n)} = \varepsilon_{2m-1}^{(n)} + \frac{\mathbf{y}}{(\mathbf{y}, \Delta\varepsilon_{2m}^{(n)})}, \quad \varepsilon_{2m+2}^{(n)} = \varepsilon_{2m}^{(n)} + \frac{\Delta\varepsilon_{2m}^{(n)}}{(\Delta\varepsilon_{2m+1}^{(n)}, \Delta\varepsilon_{2m}^{(n)})}, \qquad m, n = 0, 1, \cdots.$$

This differs from (4) in three respects. The first of these is important only in the complex case: $(\Delta\varepsilon_{2m+1}^{(n)}, \Delta\varepsilon_{2m}^{(n)})$ should be $(\Delta\varepsilon_{2m}^{(n)}, \Delta\varepsilon_{2m+1}^{(n)})$ in (9). Here, as usual, the inner product is defined so that $(\alpha a, \beta b) = \alpha\bar{\beta}(a, b)$, for all $\alpha, \beta \in \mathbf{C}$. This mistake cannot be corrected merely by altering the notation for the inner product as in [8], [9] so that $(\alpha a, \beta b) = \bar{\alpha}\beta(a, b)$, $\alpha, \beta \in \mathbf{C}$. If this less usual notation were used then $(\mathbf{y}, \Delta\varepsilon_{2m}^{(n)})$ would have to be replaced by $(\Delta\varepsilon_{2m}^{(n)}, \mathbf{y})$ in (9) instead. See [13] for more details.

The other two corrections are important in both the real and the complex cases. The first vectors on the right-hand sides of (9) should be $\varepsilon_{2m-1}^{(n+1)}$ and $\varepsilon_{2m}^{(n+1)}$ instead of $\varepsilon_{2m-1}^{(n)}$ and $\varepsilon_{2m}^{(n)}$, respectively. It is not hard to show that $\varepsilon_{2k}^{(n)}$ generated by (1) and (9), as given in [9], does not satisfy (6) even in the real case. We shall show this by a simple example. When we take $k = 1$, then (6) gives

$$(10)\qquad e_1(\mathbf{x}_n) = \frac{(\mathbf{y}, \Delta\mathbf{x}_{n+1})\mathbf{x}_n - (\mathbf{y}, \Delta\mathbf{x}_n)\mathbf{x}_{n+1}}{(\mathbf{y}, \Delta\mathbf{x}_{n+1}) - (\mathbf{y}, \Delta\mathbf{x}_n)}.$$

Now, from (1) and (9),

$$\varepsilon_1^{(n)} = \frac{\mathbf{y}}{(\mathbf{y}, \Delta\mathbf{x}_n)},$$

$$\varepsilon_2^{(n)} = \varepsilon_0^{(n)} + \frac{\Delta\varepsilon_0^{(n)}}{(\varepsilon_1^{(n+1)} - \varepsilon_1^{(n)}, \Delta\mathbf{x}_n)}$$

$$= \mathbf{x}_n + \frac{\Delta\mathbf{x}_n}{\left(\dfrac{\mathbf{y}}{(\mathbf{y}, \Delta\mathbf{x}_{n+1})}, \Delta\mathbf{x}_n\right) - \left(\dfrac{\mathbf{y}}{(\mathbf{y}, \Delta\mathbf{x}_n)}, \Delta\mathbf{x}_n\right)},$$

that is,

$$\varepsilon_2^{(n)} = \mathbf{x}_n + \frac{(\mathbf{y}, \Delta\mathbf{x}_{n+1})\Delta\mathbf{x}_n}{(\mathbf{y}, \Delta\mathbf{x}_n) - (\mathbf{y}, \Delta\mathbf{x}_{n+1})}$$

$$\neq e_1(\mathbf{x}_n) \quad \text{in general.}$$

For example if we take the case $\mathbf{y} = (1, 1)^T$, $\mathbf{x}_n = (0, 0)^T$, $\mathbf{x}_{n+1} = (1, 1)^T$, $\mathbf{x}_{n+2} = (1.5, 2)^T$, then $\varepsilon_2^{(n)} = (3, 3)^T \neq (4, 4)^T = e_1(\mathbf{x}_n)$.

Note that the difference between Algorithms 1 and 3, on the one hand, and Algorithms 2 and 4, on the other hand, is that $e_k(\mathbf{x}_n)$ is computed as $\sum_{i=0}^{k} a_i\mathbf{x}_{i+n}$ in Algorithms 1 and 3 and as $\sum_{i=0}^{k} a_i\mathbf{x}_{n+k+i}$ in Algorithms 2 and 4. That is, Algorithms 1 and 3 use a linear combination of the first consecutive $k+1$ vectors, whereas Algorithms 2 and 4 use a linear combination of the most recently computed $k+1$ vectors. Our numerical examples indicate that Algorithms 2 and 4 nearly always give a better result than Algorithms 1 and 3. This is so whether the sequences generated are convergent or divergent. In [13] it was shown, even when the given sequence $\{\mathbf{x}_n\}$ is divergent, that the TEA can be very effective especially when used in conjunction with a refinement procedure. This involves taking $\varepsilon_{2k}^{(0)}$ computed (with roundoff) by the TEA as a new starting vector $\mathbf{x}_0$ and then computing a new set of vectors $\mathbf{x}_1, \cdots, \mathbf{x}_{2k}$ and again

applying the same TEA. A similar procedure was used in [9] where the refinements are called second and subsequent "cycles" of the process.

Even for sequences of vectors arising from nonlinear problems, the TEA generally converges quadratically. Its performance is at least as good as other ε-family members VEA and SEA if Algorithm 2 or 4 is used. It is unfortunate that the excellent theoretical work of [8], [9] is marred by some critical mistakes in the numerical implementation of the TEA. Since there has been some controversy regarding this algorithm, it seems useful to give an explicit programme listing. Such a listing, written in MATLAB [7] command, using the software package MATLAB is given below.

```
//Topological ε-algorithm for Example 8 of § 3
for j = 1:n, ep1(j, 1) = x(j); end;
//Input variables
A = ⟨3.9, −3.7, 2.4, −0.6; 2.4, −2.0, 2.2, −0.6; 2.4, −3.6, 4.1, −0.9; 2.8, −5.2, ···
4.8, −0.4⟩;
b = −0.75 * ⟨1; 1; 1; 1⟩;
//Iterative scheme
for i = 1:2 * k, ···
Q = −0.25 * ⟨x(1) ** 2; x(2) ** 2; x(3) ** 2; x(4) ** 2⟩; ···
x = A * x + Q + b; ···
for j = 1:n, ep1(j, i + 1) = x(j); end; ···
end;
//Input the arbitrary vector y
y = ⟨1; 2; 3; 4⟩;
//Set ε_{-1}^{(r)} = 0, for r = 0, 1, ···, 2k
ep4 = eye(n, 2 * k + 1); for i = 1:n, ep4(i, i) = 0.0; end;
//Generate odd and even columns of the TEA
m = 2 * k;
for w = 1:m, ···
for i = 1:m, ···
dum(:, i) = ep1(:, i + 1) − ep1(:, i); ···
num = dum(:, i)' * y; ···
ep2(:, i) = ep4(:, i + 1) + y/num; ···
end; ···
m = m − 1; ···
for i = 1:m, ···
dum1(:, i) = ep2(:, i + 1) − ep2(:, i); ···
num1 = dum1(:, i)' * dum(:, i + 1); ···
ep3(:, i) = ep1(:, i + 1) + dum(:, i + 1)/num1; ···
end; ···
m = m − 1; clear ep4; ep4 = ep2; clear ep1; ep1 = ep3; ···
end;
//Extract the computed ε_{2k}^{(0)}
for i = 1:n, x(i) = ep3(i, 1);
//Print the computed ε_{2k}^{(0)}
x
```

There are a few comments to be made here. This program is Algorithm 4 written for Example 8 of [9]. This example has

$$F(\mathbf{x}) = \mathbf{b} + \mathbf{A}\mathbf{x} + Q(\mathbf{x})$$

with

$$\mathbf{A} = \begin{bmatrix} 3.9 & -3.7 & 2.4 & -0.6 \\ 2.4 & -2.0 & 2.2 & -0.6 \\ 2.4 & -3.6 & 4.1 & -0.9 \\ 2.8 & -5.2 & 4.8 & -0.4 \end{bmatrix},$$

$$\mathbf{b} = -0.75(1, 1, 1, 1)^t,$$

$$Q(\mathbf{x}) = -0.25(x_1^2, x_2^2, x_3^2, x_4^2)^t,$$

$$\mathbf{x}_0 = 1.5(1, 1, 1, 1)^t.$$

The TEA will converge to the vector $(3, 3, 3, 3)^t$ from the given starting point $\mathbf{x}_0$. (See [9] for more details of this example.) For Algorithm 3, we must replace $dum(:, i+1)$ in lines 28 and 29 of this program by $dum(:, i)$. Before running this program, the following command must be executed. $n = 4$; $k = 4$; *for* $i = 1 : n$, $x(i) = 1.5$;. Here n refers to the dimension of the matrix $\mathbf{A}$ and k $(k \leqq n)$ corresponds to that used in the text. Every refinement cycle involves repeated running of this program. A specific choice of $\mathbf{y}$ was made here so that any skeptical readers with access to the MATLAB package on a mainframe or microcomputer should easily be able to reproduce these results. Obvious modifications may be made to this program to run the other examples in [9].

3. Numerical results and comments. In this section, all the numerical examples given in [9] are used to test the viability of the four different versions of the TEA given in § 2. Since the vector sequences generated by these examples are real, Algorithms 1 and 3 are identical and so are Algorithms 2 and 4. Table 1 displays numerical results corresponding to Examples 1-8 of [9]. The numbers in Table 1 denote the number of significant digits accuracy as measured by the number SD, defined, as in [9], by

$$\mathrm{SD} = -\log_{10}(\|\mathrm{Solution} - \mathrm{Answer}\|_\infty / \|\mathrm{Answer}\|_\infty).$$

In Table 1, the second, third, fourth, etc. "cycles" are the first, second, third, etc. "refinements" produced by the method described in § 2. The auxillary vectors $\mathbf{y}$ in all the examples are generated randomly with the entries having a uniform distribution, except for Example 8 where, as indicated in the above program, $\mathbf{y}$ is chosen as $(1, 2, 3, 4)^t$. The authors in [9] have reported that they "seldom find any significant acceleration of convergence with the TEA" for the $\mathbf{y}$ chosen, including at random. In [13], we found that if the sequence of vectors is generated from $\mathbf{x}_{n+1} = \mathbf{A}\mathbf{x}_n + \mathbf{b}$ and if $\mathbf{b}$ is known, then $\mathbf{y}$ chosen as the vector $\mathbf{b}$ always performed slightly better than the one chosen randomly. In all examples carried out here, we found that Algorithm 2 (4) always performed significantly better than Algorithm 1 (3), although Algorithm 3 seems to be cited most of the time in papers published regarding the TEA.

In all numerical tests we found changes in the choice of $\mathbf{y}$ to alter the number of correct significant figures in the result by at most two. For both linear and nonlinear problems Algorithm 2 (4) performed much better than Algorithm 1 (3) without refinement, but with refinement both Algorithms 2 and 1 proved so accurate that the difference became slight, especially in Examples 3, 4, and 5. In general, Algorithm 2 (4) was also found to be more accurate than the VEA. However, as in [9], use of (9) failed to produce convergence.

TEA was also applied to the sequences produced from solving the following problem using the Gauss-Seidel method:

$$u_{xx} + u_{yy} = 0,$$

with boundary conditions $u(0, y) = \cos(\pi y)$, $u(1, y) = e^\pi \cos(\pi y)$, $u(x, 0) = e^{\pi x}$, and

TABLE 1

Table 1 displays the number of significant figures accuracy for Examples 1-8 of [9].

Example 1

$k=4$	1st cycle			2nd cycle			3rd cycle		
No. of base iterations	4	6	8	12	14	16	20	22	24
Algorithm 3	0.6	0.7	4.5	4.5	4.5	4.6	4.6	4.6	7.2
Algorithm 4	0.6	0.7	4.5	4.5	4.5	7.3	7.3	7.3	7.3

Example 2

$k=4$	1st cycle			2nd cycle			3rd cycle		
No. of base iterations	4	6	8	12	14	16	20	22	24
Algorithm 3	−0.1	2.2	11.1	11.3	11.7	13.5	13.7	13.9	14.4
Algorithm 4	0.5	4.7	12.9	13.7	16.0	15.6	15.7	16.0	16.0

Example 3

$k=8,\ c_2=0.4,\ c_3=0.2,\ c_4=0.1,\ c_5=0.001$

	Algorithm 3			Algorithm 4		
c_1	1st cycle	2nd cycle	3rd cycle	1st cycle	2nd cycle	3rd cycle
0.9	5.7	12.7	12.7	12.2	15.8	14.6
0.99	4.6	10.0	13.1	10.1	12.3	13.1
0.999	3.6	10.4	10.5	8.2	9.3	10.6
0.9999	2.6	8.8	8.2	6.0	8.1	8.5
0.99999	1.5	6.2	6.9	3.6	4.5	6.6

Example 4. Case (a)

$k=4,\ c_1=0.4,\ c_2=0.2,\ c_3=1.0,\ c_4=0.1,\ c_5=0.01$

No. of cycles	1st	2nd	3rd	4th	5th	6th
Algorithm 3	0.4	5.0	11.8	13.6	14.4	14.8
Algorithm 4	1.8	7.7	13.3	14.7	15.1	16.0

Example 4. Case (b)

$k=6,\ c_1=0.4,\ c_2=0.2,\ c_3=1.0,\ c_4=0.1,\ c_5=0.01$

No. of cycles	1st	2nd	3rd	4th
Algorithm 3	1.3	10.9	14.1	14.0
Algorithm 4	4.5	14.9	14.6	15.2

TABLE 1—continued

Example 4. Case (c)

$k=8,\ c_1=0.4,\ c_2=0.2,\ c_3=1.0,\ c_4=0.1,\ c_5=0.01$			
No. of cycles	1st	2nd	3rd
Algorithm 3	5.3	13.0	14.5
Algorithm 4	12.1	15.4	13.8

Example 5

$k=8,\ c_1=0.001,\ c_2=0.4,\ c_3=0.2,\ c_4=0.1,\ c_5=0.9$					
Algorithm 3			Algorithm 4		
1st cycle	2nd cycle	3rd cycle	1st cycle	2nd cycle	3rd cycle
6.7	13.1	15.3	14.1	15.8	16.0

Example 6

$k=4$	1st cycle			2nd cycle			3rd cycle		
No. of base iterations	4	6	8	12	14	16	20	22	24
Algorithm 3	0.8	2.2	2.3	3.2	6.2	7.3	7.6	8.2	13.7
Algorithm 4	1.1	2.8	3.8	6.9	9.5	11.2	11.4	14.4	14.8

Example 7. Case (a)

$k=3$	1st cycle		2nd cycle		3rd cycle		4th cycle	
No. of base iterations	4	6	10	12	16	18	22	24
Algorithm 3	0.9	0.9	1.8	2.5	4.0	5.9	8.8	11.9
Algorithm 4	1.0	0.9	2.0	2.5	4.4	5.8	9.1	11.9

Example 7. Case (b)

$k=4$	1st cycle			2nd cycle			3rd cycle		
No. of base iterations	4	6	8	12	14	16	20	22	24
Algorithm 3	0.9	0.8	1.5	1.4	3.1	4.6	6.9	10.0	9.9
Algorithm 4	1.0	0.9	2.7	4.2	6.1	5.9	10.1	11.5	12.6

Example 8

$k=4$	1st cycle			2nd cycle			3rd cycle		
No. of base iterations	4	6	8	12	14	16	20	22	24
Algorithm 3	0.2	−0.1	0.8	2.5	4.2	6.4	13.9	15.0	15.5
Algorithm 4	0.2	−0.1	0.8	2.5	4.2	6.4	14.1	15.1	15.7

$u(x, 1) = -e^{\pi x}$, using the well-known 5-point formula with stepsize $h = 1/9$. The solution obtained by the TEA with only 64 iterations without any refinement for the 64×64 linear system using Algorithm 4, achieved 14 significant digits agreement with the one obtained when solving the linear system by Gaussian elimination with partial pivoting. This shows it is not necessary for the number of base iterations to be $2N$, where N is the dimension of the vector space.

Although the TEA worked very well for all the examples described here and for all those described in [13], for some of which the original sequence $\{\mathbf{x}_n\}$ is divergent, a word of caution is in order. Stability problems can arise when quantities whose differences are required are large or close together and when the original sequence is divergent, especially when the dimension of the vectors is large, though other methods may also encounter difficulties in these circumstances. All the problems considered in [9] have small dimensions and computation used double precision here and in [9]. Larger dimensions and single precision would provide a more severe test of all methods. For the problems considered in [13], a method has recently been developed [14] for solving the given problem without involving divergent sequences, and if such devices are available they are likely to reduce difficulties. Also it should be mentioned that for linear problems there are other methods of conjugate gradient-type which are just as optimal as the methods considered in [9] and are also generally more stable. Nevertheless, despite these reservations, the TEA is clearly far better than suggested in [9].

All computations in this section are carried out on the Vax 11/780 computer in double precision using the software package MATLAB effectively to about 16 significant digits.

Professor D. A. Smith kindly sent me some comments on an earlier draft of this paper. He mentioned that the numerical results of [9] were in fact obtained some five years ago. He stated that the erroneous conclusions reached in [9] were not the result of equation (9), which was a separate typographical error, but of another programming error. After learning of the work reported here and in [13] he quickly located this error. When this error was corrected he also found that the TEA did work at least in some sense on all the eight numerical examples of [9], although there were many differences, most of them minor, between his revised numerical results and those reported here. This difference is due, in part, to the fact that his computations were done on a different machine and this makes precise comparison of the results for the different methods difficult. The remaining difference is probably due to the fact that he made a different random choice of $\mathbf{y}$. Significantly, for Example 8, for which we used the same $\mathbf{y}$ (given in the text), our results were essentially the same. Professor Smith stated that in general he still found the TEA to perform less well than the VEA. However, it should be noted that his calculations used only Algorithm 3, which I found to be significantly less accurate than Algorithm 4. It appears that further testing is required before definitive statements can be made about the relative performance of the methods.

Acknowledgments. The author is grateful to Dr. A. L. Andrew for many helpful discussions and to Professors A. Sidi and D. A. Smith for providing a prepublication copy of reference [9]. He also thanks Professor D. A. Smith for some comments on an earlier draft of this paper and a referee for some helpful suggestions.

REFERENCES

[1] H. M. ADELMAN AND R. T. HAFTKA, *Sensitivity analysis of discrete structural systems*, AIAA J., 24 (1986), pp. 823–832.

[2] A. L. ANDREW, *Iterative computation of derivatives of eigenvalues and eigenvectors*, J. Inst. Math. Appl., 24 (1979), pp. 209–218.
[3] C. BREZINSKI, *Généralizations de la transformation de Shanks, de la table de Padé, et de l'ε-algorithme.* Calcolo, 12 (1975), pp. 317–360.
[4] ———, *Accélération de la Convergence en Analyse Numérique*, Lecture Notes in Mathematics 584, Springer-Verlag, Berlin, New York, 1977.
[5] ———, *Padé-type approximation and general orthogonal polynomials*, ISNM 50, Birkhäuser-Verlag, Basel, Boston, Stuttgart, 1980.
[6] P. R. GRAVES-MORRIS, *Vector valued rational interpolants* I, Numer. Math., 42 (1983), pp. 331–348.
[7] C. MOLER, *Matlab Users' Guide*, Department of Computer Science, University of New Mexico, Albuquerque, NM, 1980.
[8] A. SIDI, W. F. FORD, AND D. A. SMITH, *Acceleration of convergence of vector sequences*, SIAM J. Numer. Anal., 23 (1986), pp. 178–196.
[9] D. A. SMITH, W. F. FORD, AND A. SIDI, *Extrapolation methods for vector sequences*, SIAM Rev., 29 (1987), pp. 199–233.
[10] R. C. E. TAN, *Accelerating the convergence of an iterative method for derivatives of eigensystems*, J. Comput. Phys., 67 (1986), pp. 230–235.
[11] ———, *Computing derivatives of eigensystems by the vector ε-algorithm*, IMA J. Numer. Anal., 7 (1987), pp. 485–494.
[12] ———, *An extrapolation method for computing derivatives of eigensystems*, Internat. J. Comput. Math., 22 (1987), pp. 63–73.
[13] ———, *Computing derivatives of eigensystems by the topological ε-algorithm*, Appl. Numer. Math., 3 (1987), pp. 539–550.
[14] R. C. E. TAN AND A. L. ANDREW, *Computing derivatives of eigenvalues and eigenvectors by simultaneous iteration*, IMA J. Numer. Anal., to appear.
[15] P. WYNN, *On a device for computing $e_m(S_n)$ transformation*, Math. Tables Aids Computation, 10 (1956), pp. 91–96.
[16] ———, *On the convergence and stability of the epsilon algorithm*, SIAM J. Numer. Anal., 3 (1966), pp. 91–122.