Applied Numerical Mathematics 4 (1988) 477–489
North-Holland

RECURSIVE ALGORITHMS FOR VECTOR EXTRAPOLATION METHODS

William F. FORD
Computer Services Division, NASA Lewis Research Center, Cleveland, OH 44135, U.S.A.

Avram SIDI
Computer Science Department, Technion, Israel Institute of Technology, Haifa 32000, Israel

In this work we devise three classes of recursion relations that can be used for implementing some extrapolation methods for vector sequences. One class of recursion relations can be used to implement methods like the modified minimal polynomial extrapolation and the topological epsilon algorithm, another allows implementation of methods like minimal polynomial and reduced rank extrapolation, while the remaining class can be employed in the implementation of the vector E-algorithm. Operation counts and storage requirements for these methods are also discussed, and some related techniques for special applications are also presented. Included are methods for the rapid evaluation of the vector E-algorithm.

1. Introduction

Let μ_n^m, $m,n = 0, 1, 2, \ldots$, be given complex numbers, and let b_n, $n = 0, 1, 2, \ldots$, be an arbitrary sequence of either complex numbers or vectors in a vector space B defined over the field of complex numbers. We define $G_k^{n,m}$ to be the determinant

$$G_k^{n,m} = \begin{vmatrix} \mu_n^m & \mu_{n+1}^m & \cdots & \mu_{n+k-1}^m \\ \mu_n^{m+1} & \mu_{n+1}^{m+1} & \cdots & \mu_{n+k-1}^{m+1} \\ \vdots & \vdots & & \vdots \\ \mu_n^{m+k-1} & \mu_{n+1}^{m+k-1} & \cdots & \mu_{n+k-1}^{m+k-1} \end{vmatrix} \tag{1.1}$$

and $f_k^{n,m}(b)$ to be the determinant

$$f_k^{n,m}(b) = \begin{vmatrix} b_n & b_{n+1} & \cdots & b_{n+k} \\ \mu_n^m & \mu_{n+1}^m & \cdots & \mu_{n+k}^m \\ \vdots & \vdots & & \vdots \\ \mu_n^{m+k-1} & \mu_{n+1}^{m+k-1} & \cdots & \mu_{n+k}^{m+k-1} \end{vmatrix}. \tag{1.2}$$

When b_n are scalars the determinant $f_k^{n,m}(b)$ is also a scalar, and when b_n are vectors then $f_k^{n,m}(b)$ is to be interpreted as the vector $\sum_{j=0}^k \nu_j b_{n+j}$ that results from expansion of the determinant with respect to its first row. Finally, we define $G_0^{n,m} = 1$ and $f_0^{n,m}(b) = b_n$.

0168-9274/88/$3.50 © 1988, Elsevier Science Publishers B.V. (North-Holland)

In this work we shall develop recursive algorithms for computing the quantities

$$S_k^{n,m}(b) = f_k^{n,m}(b)/f_k^{n,m}(I), \tag{1.3}$$

where $I_n = 1$ for all n, as well as the quantities

$$T_k^{n,m}(b) = f_k^{n,m}(b)/G_k^{n+1,m}. \tag{1.4}$$

Note that since

$$S_k^{n,m}(b) = T_k^{n,m}(b)/T_k^{n,m}(I), \tag{1.5}$$

$S_k^{n,m}(b)$ may be obtained from $T_k^{n,m}(b)$. Therefore, if $S_k^{n,m}(b)$ is not needed for all values of n and k, $T_k^{n,m}(b)$ may be computed recursively and (1.5) used to obtain the desired values of $S_k^{n,m}(b)$. This strategy requires fewer arithmetic operations than direct recursive computation of $S_k^{n,m}(b)$, as will be discussed later.

Quantities such as $S_k^{n,m}$ arise, for example, when one applies extrapolation (or equivalently, convergence acceleration) techniques to a vector sequence x_n, $n = 0, 1, 2, \ldots$, in B, whose limit or antilimit s is being sought, the approximations to s being $S_k^{n,m}(x)$, and the scalars μ_n^m depending on the extrapolation technique being used. Let $u_n = \Delta x_n = x_{n+1} - x_n$, $n = 0,1,\ldots$. For the modified minimal polynomial extrapolation of Sidi, Ford, and Smith [12], the approximations to s are $S_k^{n,0}(x)$ with $\mu_n^m = Q_{m+1}(u_n)$, where $Q_1, Q_2, \ldots$, are fixed linear functionals over the space B. For the topological epsilon algorithm of Brezinski [2] the approximations to s are $S_k^{n,0}(x)$ with $\mu_n^m = Q(u_{m+n})$, where Q is a linear functional over the space B. For the minimal polynomial extrapolation method of Cabay and Jackson [5] and the reduced rank extrapolation method of Eddy [6] and Mešina [8] the approximations to s are $S_k^{n,n}(x)$ with $\mu_n^m = (u_m, u_n)$ and $\mu_n^m = (w_m, u_n)$ respectively, where B is an inner product space, $(\cdot,\cdot)$ represents the inner product, and $w_n = \Delta u_n$ (see [9]). The last three of these methods, as well as the scalar and vector epsilon algorithms of Wynn [14, 15], were reviewed and tested numerically by Smith, Ford, and Sidi [13]. In addition, the convergence and stability of all four methods were analyzed for a class of vector sequences x_n, $n = 0, 1, 2, \ldots$, that includes, for example, those sequences that arise in the iterative solution of linear systems. The relevant results for $S_k^{n,0}(x)$ for the first two methods, and for $S_k^{n,n}(x)$ for the last two methods, may be found in [9,11,12] (in all cases with k fixed and $n \to \infty$). The theoretical results of [9,11,12] were also verified by numerical examples in [11,12]. (Of course, one can also consider fixing n and increasing k in all these extrapolation techniques. A convergence analysis for such an approach is provided by Sidi [10] for minimal polynomial and reduced rank extrapolation, where the connection between these extrapolation methods and well known Krylov subspace methods is also explored.

From the discussion above it is clear that there are two separate problems of interest:

(1) Recursive computation of $S_k^{n,m}(b)$ and $T_k^{n,m}(b)$ when m is fixed.

(2) Recursive computation of $S_k^{n,m}(b)$ and $T_k^{n,m}(b)$ when m varies with n as $m = n +$ a fixed integer.

In at least one application the following problem also turns out to be of interest:

(3) Recursive computation of $S_k^{n,m}(b)$ and $T_k^{n,m}(b)$ when n is fixed and m varies.

Problem (1) will be treated in Section 2, where we shall also consider related algorithms proposed by Brezinski [4] and Ford and Sidi [7]. Problems (2) and (3) will be treated in Sections

3 and 4 respectively. Special applications covering also the vector E-algorithm of Brezinski [3] will be discussed in Section 5.

2. Recursions for fixed m

Since m is kept fixed we shall denote $S_k^{n,m}(b)$ and $T_k^{n,m}(b)$ by $S_k^n(b)$ and $T_k^n(b)$ respectively.

Theorem 2.1. *$S_k^n(b)$ and $T_k^n(b)$ satisfy the three-term recursion relations*

$$S_k^n(b) = \frac{S_{k-1}^n(b) - c_k^n S_{k-1}^{n+1}(b)}{1 - c_k^n} \tag{2.1}$$

and

$$T_k^n(b) = T_{k-1}^n(b) - d_k^n T_{k-1}^{n+1}(b), \tag{2.2}$$

where

$$c_k^n = S_{k-1}^n(\mu^{m+k-1})/S_{k-1}^{n+1}(\mu^{m+k-1}) \tag{2.3}$$

and

$$d_k^n = T_{k-1}^n(\mu^{m+k-1})/T_{k-1}^{n+1}(\mu^{m+k-1}), \tag{2.4}$$

and μ^j denotes the sequence μ_n^j, $n = 0, 1, 2, \ldots$, in the same way that b denotes the sequence b_n, $n = 0, 1, 2, \ldots$.

Proof. Applying Sylvester's determinant identity (see [1, p. 23]) to $f_k^{n,m}(b)$ with respect to the first and last rows and first and last columns, we obtain

$$f_k^{n,m}(b)G_{k-1}^{n+1,m} = f_{k-1}^{n,m}(b)G_k^{n+1,m} - f_{k-1}^{n+1,m}(b)G_k^{n,m}. \tag{2.5}$$

Invoking (1.4) and (2.4), we see that (2.2) follows. Next, we replace b in (2.5) by I, and divide (2.5) by the new identity for $f_k^{n,m}(I)$. Finally, we invoke (1.3) and use the fact that $G_k^{n,m} = (-1)^{k-1} f_{k-1}^{n,m}(\mu^{m+k-1})$ to obtain (2.1). □

Note that (2.1) and (2.2) can also be written as

$$S_k^n(b) = \frac{\begin{vmatrix} S_{k-1}^n(b) & S_{k-1}^{n+1}(b) \\ S_{k-1}^n(\mu^{m+k-1}) & S_{k-1}^{n+1}(\mu^{m+k-1}) \end{vmatrix}}{\begin{vmatrix} 1 & 1 \\ S_{k-1}^n(\mu^{m+k-1}) & S_{k-1}^{n+1}(\mu^{m+k-1}) \end{vmatrix}} \tag{2.6}$$

and

$$T_k^n(b) = \frac{\begin{vmatrix} T_{k-1}^n(b) & T_{k-1}^{n+1}(b) \\ T_{k-1}^n(\mu^{m+k-1}) & T_{k-1}^{n+1}(\mu^{m+k-1}) \end{vmatrix}}{T_{k-1}^{n+1}(\mu^{m+k-1})} \tag{2.7}$$

Using Theorem 2.1, S_k^n and T_k^n can be computed by Algorithm 2.2. Without loss of generality we set $m = 0$. We assume that x_n, $0 \leq n \leq K$, are introduced one by one along with the appropriate μ_q^p. We also assume that whenever x_n and/or μ_n^p are introduced, automatically

$T_0^n(x) = x_n$ and/or $T_0^n(\mu^p) = \mu_n^p$. Finally, d_k^n, $T_k^n(b)$, and $S_k^n(b)$ are to be computed using (2.4), (2.2), and (1.5) respectively.

Algorithm 2.2 (for $m = 0$).

```
{read x_0};
for l := 1 to K do
begin
  {read x_l, μ_i^{l-1}, μ_l^i, 0 ⩽ i ⩽ l − 1};
  for p := 1 to l − 1 do for k := 1 to p do compute T_k^{p-k}(μ^{l-1});
  for k := 1 to l do
  begin
    n := l − k; compute d_k^n,
    for i := k to l − 1 do compute T_k^n(μ^i);
    compute T_k^n(x); compute T_k^n(I); compute S_k^n(x)
  end
  {save all d_k^n, 1 ⩽ n + k ⩽ l}
  {save T_k^n(μ^i), k ⩽ i ⩽ l − 1, T_k^n(x), T_k^n(I) for n + k = l, discarding all others}
end
```

The operation count for Algorithm 2.2 can be obtained as follows: To compute the scalars $T_k^n(\mu^j)$, $k \leqslant j \leqslant K$, $0 \leqslant k \leqslant n \leqslant K$, requires $\frac{1}{3}K^3 + O(K^2)$ multiplications and $\frac{1}{3}K^3 + O(K^2)$ additions. To compute the vectors $T_k^n(x)$ and $S_k^n(x)$, $0 \leqslant k \leqslant n \leqslant K$, requires $K^2 + K$ scalar-vector multiplications and $\frac{1}{2}(K^2 + K)$ vector additions. Finally, to compute remaining scalar quantities d_k^n and $T_k^n(I)$ is $O(K^2)$ multiplications and divisions, thus being negligible.

As for storage, observe that the basic computational flow is along the diagonals $n + k =$ constant in the n-k plane. Therefore, only the vectors $T_k^n(x)$ and the scalars $T_k^n(I)$ and $T_k^n(\mu^i)$, $k \leqslant i \leqslant n + k - 1$, that lie along a given diagonal need to be saved for the next diagonal (and can be overwritten by the new quantities as they are computed). All of the d_k^n, however, must be saved. Thus, with appropriate programming, one needs to save at most $\frac{1}{3}K^3 + O(K^2)$ scalars and $K + 1$ vectors.

We can devise a similar algorithm for computing the $S_k^n(x)$ directly from (2.1) and (2.3). For this algorithm $\frac{2}{3}K^3 + O(K^2)$ scalar multiplications, $\frac{1}{3}K^3 + O(K^2)$ scalar additions, $K^2 + K$ scalar-vector multiplications, and $\frac{1}{2}(K^2 + K)$ vector additions are required.

If B is a vector space of finite dimension M, M being very large, it is the vector operations that dominate the computations. In this case the two algorithms described above have almost the same operation count, namely a total of $\frac{3}{2}(K^2 + K)$ vector operations. If, however, we are not interested in all the $S_k^n(x)$, but only in $S_k^0(x)$, $k = 0, 1, 2, \ldots$, then Algorithm 2.2 (with the obvious modification that $S_k^n(x)$ is now computed only for $n = 0$), is more efficient, its operation count now being $\frac{1}{2}(K^2 + 3K)$ scalar-vector multiplications and $\frac{1}{2}(K^2 + K)$ vector additions, a total of $K^2 + 2K$ vector operations.

We now prove Theorem 2.1 using a different method that will be of considerable advantage in Sections 3 and 4. First we observe that, by construction, $S_k^{n,m}(b)$ and $T_k^{n,m}(b)$ are normalized so that

$$S_k^{n,m}(I) = 1 \tag{2.8}$$

and

$$T_k^{n,m}(b) = b_n + \sum_{j=1}^{k} \sigma_j b_{n+j}; \tag{2.9}$$

that is, for a constant sequence $b_n = C$, $n = 0, 1, 2, \ldots$, we have $S_k^{n,m}(b) = C$, and for an arbitrary sequence b_n, $n = 0, 1, 2, \ldots$, the coefficient of b_n in the expansion of $T_k^{n,m}(b)$ is unity. Another implication of (2.8) is that $S_k^{n,m}(b)$ is a weighted average of b_j, $n \leqslant j \leqslant n + k$, that is, $S_k^{n,m}(b) = \sum_{j=0}^{k} \eta_j b_{n+j}$ with $\sum_{j=0}^{k} \eta_j = 1$, though η_j are not necessarily real and/or positive. Second we observe that a sort of orthogonality property, namely

$$S_k^{n,m}(\mu^j) = T_k^{n,m}(\mu^j) = 0, \quad m \leqslant j \leqslant m + k - 1, \tag{2.10}$$

is satisfied. The proof now proceeds as follows: From Sylvester's determinant identity we infer that

$$S_k^n(b) = \alpha S_{k-1}^n(b) - \beta S_{k-1}^{n+1}(b) \tag{2.11}$$

and

$$T_k^n(b) = \gamma T_{k-1}^n(b) - \delta T_{k-1}^{n+1}(b), \tag{2.12}$$

where α, β, γ, and δ are scalars to be determined. From (2.8) we conclude that $\alpha - \beta = 1$, while from (2.9) and the fact that $T_{k-1}^{n+1}(b)$ does not contain b_n in its expansion we conclude that $\gamma = 1$. Finally, we observe that the orthogonality relations in (2.10) are automatically satisfied by (2.11) and (2.12) for $m \leqslant j \leqslant m + k - 2$, while those for $j = m + k - 1$ lead to $\beta/\alpha = c_k^n$ and $\delta/\gamma = d_k^n$. Details are left to the reader.

The contents of this section are a fuller exposition of ideas sketched out in the recent work of Ford and Sidi [7]. That work treated a generalized extrapolation procedure for scalar sequences, and the vector method was only incidental.

Algorithms related to those described here have previously been proposed by Brezinski, namely the RPA and CRPA [4]. The RPA computes a ratio of determinants (the numerator of which is a vector) having a single index k. The CRPA is a slight variation of the RPA in which the determinants depend on two indices, n and k. For a given sequence x_n, $n = 0, 1, 2, \ldots$, the ratio being computed by CRPA is what we have called $T_k^n(x)$ with

$$\mu_n^p = \langle z_p, x_n \rangle, \tag{2.13}$$

where z_i are members of the dual space B^* of B, and $\langle \cdot, \cdot \rangle$ is the bilinear form of duality between B and B^*. For this choice of μ_n^p,

$$T_k^n(\mu^p) = \langle z_p, T_k^n(x) \rangle, \tag{2.14}$$

and (2.2) and (2.4) (with $m = 1$) reduce to the CRPA recursion relation

$$T_k^n(x) = T_{k-1}^n(x) - \frac{\langle z_k, T_{k-1}^n(x) \rangle}{\langle z_k, T_{k-1}^{n+1}(x) \rangle} T_{k-1}^{n+1}(x). \tag{2.15}$$

When μ_n^p is of the form (2.13) it is more efficient to use (2.15) than (2.2) and (2.4), since the recursion for the coefficients d_k^n is avoided.

3. Recursions for variable m

We now treat the case in which $m = n +$ a fixed integer. Without ambiguity we again denote $S_k^{n,m}(b)$ and $T_k^{n,m}(b)$ by $S_k^n(b)$ and $T_k^n(b)$ respectively. We shall also need the auxiliary quantities $\tilde{S}_k^n(b) \equiv S_k^{n,m-1}(b)$ and $\tilde{T}_k^n(b) = T_k^{n,m-1}(b)$.

Theorem 3.1. *$S_k^n(b)$ and $T_k^n(b)$ can be computed from the recursion relations*

$$S_k^n(b) = \frac{S_{k-1}^n(b) - c_k^n \tilde{S}_{k-1}^{n+1}(b)}{1 - c_k^n},$$
$$\tilde{S}_k^n(b) = \frac{S_{k-1}^n(b) - \tilde{c}_k^n \tilde{S}_{k-1}^{n+1}(b)}{1 - \tilde{c}_k^n} \tag{3.1}$$

and

$$T_k^n(b) = T_{k-1}^n(b) - d_k^n \tilde{T}_{k-1}^{n+1}(b), \qquad \tilde{T}_k^n(b) = T_{k-1}^n(b) - \tilde{d}_k^n \tilde{T}_{k-1}^{n+1}(b), \tag{3.2}$$

where

$$c_k^n = S_{k-1}^n(\mu^{m+k-1})/\tilde{S}_{k-1}^{n+1}(\mu^{m+k-1}), \qquad \tilde{c}_k^n = S_{k-1}^n(\mu^{m-1})/\tilde{S}_{k-1}^{n+1}(\mu^{m-1}) \tag{3.3}$$

and

$$d_k^n = T_{k-1}^n(\mu^{m+k-1})/\tilde{T}_{k-1}^{n+1}(\mu^{m+k-1}), \qquad \tilde{d}_k^n = T_{k-1}^n(\mu^{m-1})/\tilde{T}_{k-1}^{n+1}(\mu^{m-1}). \tag{3.4}$$

Furthermore, $S_k^n(b)$ and $T_k^n(b)$ also satisfy the four-term (lozenge) recursion relations

$$S_{k+1}^n(b) = \frac{\begin{vmatrix} S_{k-1}^n(b) & S_{k-1}^{n+1}(b) & S_k^{n+1}(b) \\ S_{k-1}^n(\mu^m) & S_{k-1}^{n+1}(\mu^m) & S_k^{n+1}(\mu^m) \\ S_{k-1}^n(\mu^{m+k}) & S_{k-1}^{n+1}(\mu^{m+k}) & S_k^{n+1}(\mu^{m+k}) \end{vmatrix}}{\begin{vmatrix} 1 & 1 & 1 \\ S_{k-1}^n(\mu^m) & S_{k-1}^{n+1}(\mu^m) & S_k^{n+1}(\mu^m) \\ S_{k-1}^n(\mu^{m+k}) & S_{k-1}^{n+1}(\mu^{m+k}) & S_k^{n+1}(\mu^{m+k}) \end{vmatrix}} \tag{3.5}$$

and

$$T_{k+1}^n(b) = \frac{\begin{vmatrix} T_{k-1}^n(b) & T_{k-1}^{n+1}(b) & T_k^{n+1}(b) \\ T_{k-1}^n(\mu^m) & T_{k-1}^{n+1}(\mu^m) & T_k^{n+1}(\mu^m) \\ T_{k-1}^n(\mu^{m+k}) & T_{k-1}^{n+1}(\mu^{m+k}) & T_k^{n+1}(\mu^{m+k}) \end{vmatrix}}{\begin{vmatrix} T_{k-1}^{n+1}(\mu^m) & T_k^{n+1}(\mu^m) \\ T_{k-1}^{n+1}(\mu^{m+k}) & T_k^{n+1}(\mu^{m+k}) \end{vmatrix}}. \tag{3.6}$$

Proof. Applying Sylvester's determinant identity to $f_k^{n,m}(b)$ with respect to the first and last rows and first and last columns, we obtain

$$f_k^{n,m}(b) = \rho f_{k-1}^{n,m}(b) - \sigma f_{k-1}^{n+1,m}(b), \tag{3.7}$$

where ρ and σ are scalars. Similarly, applying Sylvester's identity to $f_k^{n,m-1}(b)$ with respect to the first two rows and first and last columns, we obtain

$$f_k^{n,m-1}(b) = \tau f_{k-1}^{n,m}(b) - \omega f_{k-1}^{n+1,m}(b), \tag{3.8}$$

where τ and ω are scalars. It follows directly that

$$S_k^n(b) = \alpha S_{k-1}^n(b) - \beta \tilde{S}_{k-1}^{n+1}(b), \qquad \tilde{S}_k^n(b) = \tilde{\alpha} S_{k-1}^n(b) - \tilde{\beta} \tilde{S}_{k-1}^{n+1}(b). \tag{3.9}$$

The proof of (3.1) and (3.3) can now be completed by using the normalization and orthogonality relations for $S_k^{n,m}(b)$ given in (2.8) and (2.10) just as in the previous section.

As for (3.5), we first observe that the normalization condition (2.8) holds by construction, and that the orthogonality conditions for $S_{k+1}^n(b)$, namely $S_{k+1}^n(\mu^j) = 0$, $m \leqslant j \leqslant m+k$, are satisfied by the quotient on the right-hand side. Hence what remains to be shown is that $S_{k+1}^n(b)$ can be expressed as a linear combination of $S_k^{n+1}(b)$, $S_k^n(b)$, and $S_{k-1}^{n+1}(b)$, and this can be accomplished by manipulating the two equations in (3.9). Details are left to the reader.

The proof of (3.2), (3.4), and (3.6) can be carried out in a similar manner. □

Using Theorem 3.1 the S_k^n and T_k^n can now be computed by Algorithm 3.2. Without loss of generality we set $m = n$. Again we assume that x_n, $0 \leqslant n \leqslant K$, are introduced one by one along with the appropriate μ_n^p, and that automatically $T_0^n(x) = x_n$, $\tilde{T}_0^n(x) = x_n$, $T_0^n(\mu^p) = \mu_n^p$, and $\tilde{T}_0^n(\mu^p) = \mu_n^p$.

Algorithm 3.2 (for $m = n$).

```
{read x_0};
for l := 1 to K do
begin
  {read x_l, μ_i^{l-1}, μ_l^i, 0 ≤ i ≤ l-1};
  for p := 1 to l-1 do for k := 1 to p do
    begin compute T_k^{p-k}(μ^{l-1}); if p > k then compute T̃_k^{p-k}(μ^{l-1}) end;
  for k := 1 to l do
  begin
    n := l-k; compute d_k^n; if n > 0 then compute d̃_k^n;
    compute T_k^n(x); compute T_k^n(I); compute S_k^n(x);
    if n > 0 then begin compute T̃_k^n(x); compute T̃_k^n(I) end
  end;
  for i := 0 to l-1 do for k := 1 to l do
    begin
      n := l-k; if i < n then compute T_k^n(μ^i);
      if (i < n-1 or i = l-1) and n > 0 then compute T̃_m^n(μ^i)
    end
  {save all d_k^n and d̃_k^n, 1 ≤ n+k ≤ l}
  {save T_k^n(μ^i), k ≤ i ≤ l-1, T_k^n(x), and T_k^n(I) on the diagonal n+k = l, discarding
   all others}
end
```

(The **if** statements near the end of Algorithm 3.2 are used to avoid calculating $T_k^n(\mu^i)$ or $\tilde{T}_k^n(\mu^i)$ when they are known to vanish.)

The operation count for Algorithm 3.2 is as follows: Computing all needed values of $T_k^n(\mu^i)$, $\tilde{T}_k^n(\mu^i)$, $T_k^n(I)$, d_k^n, and $\tilde{d}_k^n$, a total of $\frac{2}{3}K^3 + O(K^2)$ scalar quantities, requires $\frac{2}{3}K^3 + O(K^2)$ multiplications and $\frac{2}{3}K^3 + O(K^2)$ additions. Computing all needed values of $T_k^n(x)$, $\tilde{T}_k^n(x)$, and $S_k^n(x)$, a total of $\frac{3}{2}(K^2 + K)$ vector quantities requires $\frac{3}{2}(K^2 + K)$ scalar-vector multiplications and $K^2 + K$ vector additions.

As for storage, since the basic computational flow is along the diagonals $n + k =$ constant in the n-k plane, none of the quantities $\tilde{T}_k^n(\mu^i)$, $\tilde{T}_k^n(I)$, or $\tilde{T}_k^n(x)$ must be saved. All of the d_k^n and $\tilde{d}_k^n$ need to be saved, however, and the storage requirements for $T_k^n(\mu^i)$, $T_k^n(I)$ and $T_k^n(x)$ are as in Algorithm 2.2.

4. Recursions for fixed n and varying m

Let us now treat the case in which n is fixed and m is varying. Without ambiguity we denote $S_k^{n,m}(b)$ and $T_k^{n,m}(b)$ by $\hat{S}_k^m(b)$ and $\hat{T}_k^m(b)$ respectively.

Theorem 4.1. *$\hat{S}_k^m(b)$ and $\hat{T}_k^m(b)$ can be computed from the recursion relations*

$$\hat{S}_k^m(b) = \frac{\hat{S}_{k-1}^m(b) - \hat{c}_k^m \hat{S}_k^{m-1}(b)}{1 - \hat{c}_k^m} \tag{4.1}$$

and

$$\hat{T}_k^m(b) = \frac{\hat{T}_{k-1}^m(b) - \hat{d}_k^m \hat{T}_k^{m-1}(b)}{1 - \hat{d}_k^m}, \tag{4.2}$$

where

$$\hat{c}_k^m = \hat{S}_{k-1}^m(\mu^{m+k-1}) / \hat{S}_k^{m-1}(\mu^{m+k-1}) \tag{4.3}$$

and

$$\hat{d}_k^m = \hat{T}_{k-1}^m(\mu^{m+k-1}) / \hat{T}_k^{m-1}(\mu^{m+k-1}). \tag{4.4}$$

Proof. Eliminating $f_{k-1}^{n+1,m}(b)$ from (3.7) and (3.8), we obtain a relation among $f_k^{n,m}(b)$, $f_{k-1}^{n,m}(b)$, and $f_k^{n,m-1}(b)$, which can also be expressed as

$$\hat{S}_k^m(b) = \alpha \hat{S}_{k-1}^m(b) - \beta \hat{S}_k^{m-1}(b), \tag{4.5}$$

where α and β are scalars to be determined. The proof of (4.1) and (4.3) can now completed by using the normalization and orthogonality relations for $S_k^{n,m}(b)$ given in (2.8) and (2.10) just as in the previous sections.

The proof of (4.2) and (4.4) can be accomplished in a similar manner. □

We note that (4.1) and (4.2) can also be expressed as

$$\hat{S}_k^m(b) = \frac{\begin{vmatrix} \hat{S}_{k-1}^m(b) & \hat{S}_k^{m-1}(b) \\ \hat{S}_{k-1}^m(\mu^{m+k-1}) & \hat{S}_k^{m-1}(\mu^{m+k-1}) \end{vmatrix}}{\begin{vmatrix} 1 & 1 \\ \hat{S}_{k-1}^m(\mu^{m+k-1}) & \hat{S}_k^{m-1}(\mu^{m+k-1}) \end{vmatrix}} \tag{4.6}$$

and

$$\hat{T}_k^m(b) = \frac{\begin{vmatrix} \hat{T}_{k-1}^m(b) & \hat{T}_k^{m-1}(b) \\ \hat{T}_{k-1}^m(\mu^{m+k-1}) & \hat{T}_k^{m-1}(\mu^{m+k-1}) \end{vmatrix}}{\begin{vmatrix} 1 & 1 \\ \hat{T}_{k-1}^m(\mu^{m+k-1}) & \hat{T}_k^{m-1}(\mu^{m+k-1}) \end{vmatrix}}. \tag{4.7}$$

Using Theorem 4.1 the $\hat{S}_k^m$ and $\hat{T}_k^m$ can be computed by Algorithm 4.2. We note that this algorithm works provided both $\hat{T}_0^m$, $m = 0, 1, \ldots$, *and* $\hat{T}_k^0$, $k = 0, 1, \ldots$, are given. Although $\hat{T}_0^m(b) = T_0^{n,m}(b) = b_n$ are immediately available for all m, $\hat{T}_k^0(b) = T_k^{n,0}(b)$ have to be computed for increasing k and this can be done, for example, by using Algorithm 2.2. (This should be compared with Algorithms 2.2 and 3.2, for both of which $T_0^n(b) = b_n$ are the only quantities that need to be given.) Without loss of generality, we set $n = 0$. Consequently we assume that $\hat{T}_0^m(x) = x_0$ and $\hat{T}_0^m(\mu^p) = \mu_0^p$ automatically. Finally, $\hat{d}_k^m$, $\hat{T}_k^m(b)$ and $\hat{S}_k^m(b)$ are to be computed using (4.4), (4.2), and (1.5) respectively.

Algorithm 4.2 (for $n = 0$).

```
{read x_0}
for l := 1 to L do
begin
  {read x_l, μ_i^{l-1}, μ_l^i, 0 ≤ i ≤ l − 1, compute T̂_l^0(x), T̂_k^0(μ^{l-1}), 1 ≤ k ≤ l − 1};
  for p := 1 to l − 1 do for k := 1 to p − 1 do compute T̂_k^{p-k}(μ^{l-1});
  for k := 1 to l − 1 do
  begin
    m := l − k; compute d̂_k^m;
    compute T̂_k^m(x); compute T̂_k^m(I); compute Ŝ_k^m(x)
  end
  {save all d̂_k^m, 2 ≤ m + k ≤ l}
  {save T̂_k^m(μ^l), T̂_k^m(x), T̂_k^m(I), m + k = l, discarding all others}
end
```

5. Special applications

The algorithms developed in Sections 2 and 3 are most effective when used to generate all quantities $S_k^n(x)$ along the diagonals $n + k = l$ in the n-k plane for increasing l. In some applications not all $S_k^n(x)$ may be needed, and direct use of Algorithms 2.2 and 3.2 for these applications may be relatively expensive as far as the number of vector operations is concerned. The applications that are of special interest are those in which

(1) $S_k^N(x)$, $k = 0, 1, 2, \ldots$ (N fixed) are needed,
(2) $S_K^n(x)$, $n = 0, 1, 2, \ldots$ (K fixed) are needed.

5.1. Computing $S_k^N(x)$, $k = 0, 1, 2, \ldots$ (N fixed)

As mentioned earlier, invoking (1.5) in Algorithm 2.2 for computing $S_k^n(x)$ only when $n = N$, the computation of $S_k^N(x)$ requires $2k + 1$ vector operations for each k.

Similarly, invoking (1.5) in Algorithm 3.2 for computing $S_k^n(x)$ only when $n = N$, the computation of $S_k^N(x)$ requires $4k + 1$ vector operations. But if we recall that $S_k^N(x) \equiv S_k^{N,N}(x)$ and $T_k^N(x) = T_k^{N,N}(x)$ in Algorithm 3.2, we see that the sequence $S_k^N(x)$, $k \geq 0$, is a subsequence of $S_k^{n,N}(x)$, $n + k \geq 0$, in which $m = N$ is fixed. Thus Algorithm 2.2 can be used to generate $S_k^N(x)$, at the cost of $2k + 1$ vector operations for each k.

In view of what has been said in the previous paragraph, we now give a precise vector operation count for the minimal polynomial extrapolation (MPE) and the reduced rank extrapolation (RRE) for a process that was termed "cycling" in [13]. At each "cycle" the vectors $x_0, x_2, \ldots, x_{K+1}$ are generated and the vector $S_K^{0,0}(x)$ is computed and used as the vector x_0 for the next cycle. Thus, given the x_i, we would like to know the overhead for each cycle. We recall that $\mu_q^p = (u_p, u_q)$ for MPE, and $\mu_q^p = (w_p, u_q)$ for RRE, that $u_i = x_{i+1} - x_i$, and $w_i = u_{i+1} - u_i$, and that $\overline{(y, z)} = (z, y)$. To avoid saving the u_i as well as the x_i we will compute μ_q^p from inner products (x_i, x_j). Then one cycle (using Algorithm 2.2 and computing $S_k^{0,0}(x)$, $1 \leq k \leq K$) requires $\frac{1}{2}(K^2 + 3K)$ scalar-vector multiplications and $\frac{1}{2}(K^2 + K)$ vector additions for both MPE and RRE, and $L = \frac{1}{2}(K^2 + 5K + 4)$ inner products for MPE and $L + 1$ inner products for RRE. Thus the total number of vector operations for one cycle of MPE or RRE requires $\frac{3}{2}(K^2 + 3K) + O(1)$ vector operations.

If the modified minimal polynomial extrapolation method is used with cycling, then by proper choice of the functionals $Q_1, Q_2, \ldots$, the need for computing inner products can be eliminated altogether (see [12]), the rest of the operation count being as for MPE or RRE.

It is instructive to compare the overhead for cycling MPE and RRE with that for the vector epsilon algorithm (VEA). VEA is defined and implemented through the recursion relation

$$\varepsilon_{k+1}^{(n)} = \varepsilon_{k-1}^{(n+1)} + \frac{\overline{\Delta\varepsilon_k^{(n)}}}{\left(\Delta\varepsilon_k^{(n)}, \Delta\varepsilon_k^{(n)}\right)}, \qquad k \geq 0, \quad n \geq 0, \tag{5.1}$$

where $\Delta\varepsilon_k^{(n)} = \varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}$, with the initial conditions

$$\varepsilon_{-1}^{(n)} = 0, \quad \varepsilon_0^{(n)} = x_n, \qquad n \geq 0. \tag{5.2}$$

Thus, the computation of $\varepsilon_k^{(n)}$ for $k \geq 2$ requires one scalar-vector multiplication, two vector additions, and one inner product. For $\varepsilon_1^{(n)}$ only one vector addition is required. Now as is suggested by experience and as can be justified heuristically, for given K, $S_K^{0,0}(x)$ for MPE or RRE and $\varepsilon_{2K}^{(0)}$ would have comparable performance. The overhead for $\varepsilon_{2K}^{(0)}$ turns out to be $2K^2 + K$ scalar-vector multiplications, $4K^2$ vector additions, and $2K^2 + K$ inner products, a total of $8K^2 + O(K)$ vector operations. Thus, it is seen that VEA, in addition to requiring $2K + 1$ vectors x_i (as opposed to $K + 2$ for MPE or RRE), requires a much larger overhead in vector operations. Storage requirements are similar, namely, about $2K$ vectors for VEA and about K vectors for MPE or RRE.

5.2. Computing $S_K^n(x)$, $n = 0, 1, 2, \ldots$ (K fixed)

Invoking (1.5) for computing $S_k^{n,m}(x)$ (m fixed) only for $k = K$, the computation of $S_k^{n,m}(x)$ requires $2K + 1$ vector operations for Algorithm 2.2.

The computation of $S_K^{n,n}(x)$ by Algorithm 3.2, however, requires $4K+1$ vector operations. Employing the relation

$$S_k^{n,m}(x) = \sum_{i=0}^{k} \eta_{k,i}^{n,m} x_{n+i} \qquad \left(\sum_{i=0}^{k} \eta_{k,i}^{n,m} = 1 \right), \tag{5.3}$$

for computing $S_k^{n,m}(x)$ requires $2k+1$ vector operations for each k, thus it can be used to evaluate $S_K^{n,n}(x)$, reducing the number of vector operations to $2K+1$. The coefficients $\eta_{k,i}^n \equiv \eta_{k,i}^{n,n}$, along with the auxiliary $\tilde{\eta}_{k,i}^n \equiv \eta_{k,i}^{n,n-1}$, can be conveniently computed by Theorem 3.1 using the recursion relations

$$\begin{aligned} \eta_{k,i}^n &= \frac{\eta_{k-1,i}^n - c_k^n \tilde{\eta}_{k-1,i-1}^{n+1}}{1 - c_k^n}, \\ \tilde{\eta}_{k,i}^n &= \frac{\eta_{k-1,i}^n - \tilde{c}_k^n \tilde{\eta}_{k-1,i-1}^{n+1}}{1 - \tilde{c}_k^n}, \end{aligned} \qquad 0 \leqslant i \leqslant k, \tag{5.4}$$

with

$$\begin{aligned} &\eta_{0,0}^n = \tilde{\eta}_{0,0}^n = 1 \quad \text{all } n, \\ &\eta_{k,i}^n = \tilde{\eta}_{k,i}^n = 0 \quad \text{for } i < 0 \text{ or } i > k. \end{aligned} \tag{5.5}$$

We leave the proof of (5.4) and (5.5) to the reader.

Finally, we consider two related applications involving the vector E-algorithm. This algorithm, whose details will not be given here, computes recursively the quantities

$$E_k(x_n) = \frac{\begin{vmatrix} x_n & g_1(n) & \dots & g_k(n) \\ (y, \Delta x_n) & (y, \Delta g_1(n)) & \dots & (y, \Delta g_k(n)) \\ \vdots & \vdots & & \vdots \\ (y, \Delta x_{n+k-1}) & (y, \Delta g_1(n+k-1)) & \dots & (y, \Delta g_k(n+k-1)) \end{vmatrix}}{\begin{vmatrix} (y, \Delta g_1(n)) & \dots & (y, \Delta g_k(n)) \\ \vdots & & \vdots \\ (y, \Delta g_1(n+k-1)) & \dots & (y, \Delta g_k(n+k-1)) \end{vmatrix}}, \tag{5.6}$$

where

$$\Delta x_n = x_{n+1} - x_n, \qquad \Delta g_i(n) = g_i(n+1) - g_i(n),$$

and y is an arbitrary fixed vector. We first note that $x_n, g_1(n), \dots, g_k(n)$, the entries of the first row of the numerator determinant, are the nth members of $k+1$ different vector sequences; therefore the top row of $E_k(x_{n+1})$ need not have any vector in common with the top row of $E_k(x_n)$, which is quite different than the cases treated in this work so far.

Nevertheless, the sequence $E_k(x_N)$, $k = 0, 1, \dots$, with N fixed, can be efficiently computed using Algorithm 2.2. For convenience, we define

$$g_0(l) = x_l, \qquad h_p^q(l) = g_p(l+q+1) - g_p(l+q). \tag{5.7}$$

Then, setting

$$b_p = g_p(N), \qquad p = 0, 1, 2, \ldots,$$
$$\mu_p^q = (y, h_p^q(N)), \quad p, q = 0, 1, 2, \ldots, \tag{5.8}$$

and substituting into (5.6), we find that

$$E_k(x_N) = T_k^0(b), \tag{5.9}$$

which may be obtained using Algorithm 2.2 at the cost of $2k$ vector operations for each k.

Computing $E_K(x_m)$, $m = 0, 1, 2, \ldots$, with K fixed, by Algorithm 2.2 as explained above, requires $K^2 + K$ vector operations for each m. This number can be reduced to $2K$ by using the fact that

$$E_k(x_m) = x_m + \sum_{i=1}^{k} \hat{\sigma}_{k,i}^m g_i(m), \tag{5.10}$$

for some scalars $\hat{\sigma}_{k,i}^m$, $1 \leqslant i \leqslant k$, which follows from (5.6). Note that the $\hat{\sigma}_{k,i}^m$ do not depend on the first row of the numerator determinant in $E_k(x_m)$. Letting now

$$\begin{aligned} &(y, \Delta x_{m+i}) = \mu_0^{m+i}, && i \geqslant 0, \\ &(y, \Delta g_j(m+i)) = \mu_j^{m+i}, && i \geqslant 0, \quad j \geqslant 1, \end{aligned} \tag{5.11}$$

we see that for an arbitrary sequence $b_0, b_1, b_2, \ldots$,

$$T_k^{0,m}(b) = b_0 + \sum_{i=1}^{k} \hat{\sigma}_{k,i}^m b_i. \tag{5.12}$$

Thus Theorem 4.1 can be conveniently used to generate the $\hat{\sigma}_{k,i}^m$ recursively, the appropriate recursions being

$$\hat{\sigma}_{k,i}^m = \frac{\hat{\sigma}_{k-1,i}^m - \hat{c}_k^m \hat{\sigma}_{k,i}^{m-1}}{1 - \hat{c}_k^m}, \quad 0 \leqslant i \leqslant k, \tag{5.13}$$

where

$$\begin{aligned} &\hat{\sigma}_{k,0}^m = 1 \quad \text{all } m \text{ and } k \\ &\hat{\sigma}_{k,i}^m = 0, \quad \text{for } i < 0 \text{ or } i > k, \quad \text{all } m \text{ and } k, \end{aligned} \tag{5.14}$$

and $\hat{\sigma}_{k,i}^0$, $0 \leqslant i \leqslant k$, have to be given. Actually $\hat{\sigma}_{k,i}^0$, which are related to $T_k^{0,0}$, can also be computed recursively by using Theorem 2.1. We leave the details to the reader.

6. Conclusions

In this work we have presented recursive means for evaluating certain vector quantities $S_k^{n,m}(x)$ and $T_k^{n,m}(x)$ that arise in a number of vector extrapolation methods. Our methods are divided into two major categories, one in which the $S_k^{n,m}(x)$, for fixed m, are computed by a three-term recursion relation, and another in which the $S_k^{n,m}(x)$, for $m = n +$ a fixed integer, are computed essentially by a four-term (lozenge) recursion relation. The methods in the former

category can be used to implement the modified minimal polynomial extrapolation technique and the topological epsilon algorithm, and are related to, but not identical to, others that have been proposed in recent literature. Those in the latter category, however, are, to the best of our knowledge, new, and permit recursive evaluation of the extrapolants that arise in, for example, minimal polynomial and reduced rank extrapolation techniques. We have also devised recursion relations for the case in which n is being held fixed while m is increasing, and have used them in the implementation of the vector E-algorithm.

References

[1] G.A. Baker, Jr. and P.R. Graves-Morris, *Padé Approximants, Part I: Basic Theory*, Encyclopedia of Mathematics and Its Applications 13 (Addison-Wesley, London, 1981).

[2] C. Brezinski, *Accélération de la Convergence en Analyse Numérique*, Lecture Notes in Mathematics 584 (Springer, Berlin, 1977).

[3] C. Brezinski, A general extrapolation algorithm, *Numer. Math. 35* (1980) 175–187.

[4] C. Brezinski, Recursive interpolation, extrapolation and projection, *J. Comput. Appl. Math. 9* (1983) 369–376.

[5] S. Cabay and L.W. Jackson, A polynomial extrapolation method for finding limits and antilimits of vector sequences, *SIAM J. Numer. Anal. 13* (1976) 734–752.

[6] R.P. Eddy, Extrapolating to the limit of a vector sequence, in: P.C.C. Wang, ed., *Information Linkage between Applied Mathematics and Industry* (Academic Press, New York, 1979) 387–396.

[7] W.F. Ford and A. Sidi, An algorithm for a generalization of the Richardson extrapolation process, *SIAM J. Numer. Anal. 24* (1987) 1212–1232.

[8] M. Mešina, Convergence acceleration for the iterative solution of the equations $X = AX + f$, *Comput. Methods Appl. Mech. Engrg. 10* (1977) 165–173.

[9] A. Sidi, Convergence and stability properties of minimal polynomial and reduced rank extrapolation algorithms, *SIAM J. Numer. Anal. 23* (1986) 197–209.

[10] A. Sidi, Extrapolation vs. projection methods for linear systems of equations, *J. Comput. Appl. Math. 22* (1988) 71–88.

[11] A. Sidi and J. Bridger, Convergence and stability analyses for some vector extrapolation methods in the presence of defective iteration matrices, *J. Comput. Appl. Math. 22* (1988) 35–61.

[12] A. Sidi, W.F. Ford and D.A. Smith, Acceleration of convergence of vector sequences, *SIAM J. Numer. Anal. 23* (1986) 178–196.

[13] D.A. Smith, W.F. Ford and A. Sidi, Extrapolation methods for vector sequences, *SIAM Rev. 29* (1987) 199–233; see also: Correction to "Extrapolation methods for vector sequences", *SIAM Rev.* (to appear).

[14] P. Wynn, On a device for computing the $e_m(S_n)$ transformation, *MTAC 10* (1956) 91–96.

[15] P. Wynn, Acceleration techniques for iterated vector and matrix problems, *Math. Comp. 16* (1962) 301–322.