ZASTOSOWANIA MATEMATYKI
APPLICATIONES MATHEMATICAE
20, 3 (1990), pp. 435 443

C. BREZINSKI (Lille)

BORDERING METHODS AND PROGRESSIVE FORMS FOR SEQUENCE TRANSFORMATIONS

Abstract. Most of the scalar and vector transformations can be unified in the same framework. An economical computational scheme, based on the bordering method for solving recursively a system of linear equations, is presented for these transformations. Such a scheme is fundamental in the progressive forms of the algorithms used for implementing these sequence transformations. Usually these progressive forms are numerically more stable than the normal forms as shown by an example.

1. Introduction. Sequence transformations are used to accelerate the convergence. In most of them the members of the transformed sequence are given as a ratio of two determinants. This is due to the fact that the solution of a system of linear equations is implicitly involved in the computation as shown in [3] for a particular case. As every sequence, the transformed sequence depends on an index n but, in our case, it also depends on the size k of the determinants appearing in the ratio that is the dimension of the underlying linear system. Thus sequence transformations transform the sequence to accelerate into a set of sequences whose members depend on two indices: n (numbering the successive members of the transformed sequence) and k (numbering the various transformed sequences obtained).

It is well known that a numerical analyst is unable to compute the value of a determinant since it needs too many arithmetical operations and because of the rounding errors due to the computer's arithmetic. Thus there exist recursive algorithms for computing the members of the transformed sequences for all k and all n without computing the determinants involved in their mathematical expressions (see [9] for a review). In some applications all these quantities are not needed but one wants to fix the index n (most of the time to zero) and to compute the members of the transformed sequences for $k = 0, 1, 2, \ldots$ It is of course possible to use the general algorithm but it is

uneconomical since many unneeded quantities are also obtained and the amount of work is much larger than necessary. It is possible to reduce it by using a bordering method since the dimension of the underlying system of linear equations increases with k at each step. Such an idea was already used for Shanks' transformation [2]. It gave rise to economical methods for computing diagonal sequences of Padé approximants and continued fractions. It was also extended to a generalisation of Shanks' transformation for sequences of vectors [6]. Since this time quite a number of new transformations were obtained. Some of them deal with scalar sequences, some others with vector sequences or, more generally, with sequences of elements of a vector space.

It is the purpose of this paper to extend the bordering method to these new transformations and then to use it for their implementation by the so-called progressive forms.

2. Sequence transformations. Let E be a vector space on K ($\boldsymbol{R}$ or $\boldsymbol{C}$), e_i elements of E, and b_i and $a_j^{(i)}$ elements of K. We consider $R_k \in E$ defined by

$$R_k = \alpha_0 e_0 + \ldots + \alpha_k e_k, \tag{1}$$

where the α_i's (which depend on k) are the solution of the system of linear equations

$$\begin{aligned} &\alpha_0 b_0 + \ldots + \alpha_k b_k = 1, \\ &\alpha_0 a_1^{(0)} + \ldots + \alpha_k a_1^{(k)} = 0, \\ &\ldots\ldots\ldots\ldots \\ &\alpha_0 a_k^{(0)} + \ldots + \alpha_k a_k^{(k)} = 0. \end{aligned} \tag{2}$$

R_k is given as a ratio of two determinants

$$R_k = \frac{\begin{vmatrix} e_0 & \ldots & e_k \\ a_1^{(0)} & \ldots & a_1^{(k)} \\ \ldots & \ldots & \ldots \\ a_k^{(0)} & \ldots & a_k^{(k)} \end{vmatrix}}{\begin{vmatrix} b_0 & \ldots & b_k \\ a_1^{(0)} & \ldots & a_1^{(k)} \\ \ldots & \ldots & \ldots \\ a_k^{(0)} & \ldots & a_k^{(k)} \end{vmatrix}}, \quad k = 0, 1, \ldots, \tag{3}$$

where the determinant in the numerator denotes the element of E obtained by expanding it with respect to its first row by using the classical rules for the expansion of a determinant.

Most of the sequence transformations actually used are of this type. Let us review them and begin by the case $E = \boldsymbol{C}$. As explained in the introduction, the index n is always kept fixed.

E-algorithm [7]: with

$$e_i = S_{n+i}, \quad a_j^{(i)} = g_j(n+i) \quad \text{and} \quad b_i = 1$$

we obtain

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

Many sequence transformations are particular cases of the E-algorithm: Shanks' transformation (that is the ε-algorithm), polynomial (Richardson) and rational (Thiele) extrapolation, Padé approximation, G-transformation, Germain-Bonne transformation, etc...

Composite sequence transformations [10]: let t_i: $(S_n) \to (t_i^{(n)})$ be various sequence transformations. For a fixed index l, if we set

$$e_i = t_{l+i}^{(n)}, \quad a_j^{(i)} = \Delta t_{l+i}^{(n+j-1)} \quad \text{and} \quad b_i = 1,$$

then

$$R_k = {}_l T_k^{(n)}$$

as defined in [10].

Formal orthogonal polynomials [6]: with

$$e_i = x^i, \quad a_j^{(i)} = c_{i+j-1} \quad \text{and} \quad b_i = 1$$

we obtain

$$R_k = P_k(x)$$

with the normalisation $P_k(1) = 1$.

w-algorithm [23]: with

$$e_i = a_{n+i}, \quad a_j^{(i)} = a_{n+i+j}, \quad b_0 = 1 \quad \text{and} \quad b_i = 0 \quad \text{for } i \geqslant 1$$

we have

$$R_k = w_{2k}^{(n)}.$$

If $a_n = \Delta^n S_i$, then we recover Shanks' transformation again. Let f be a function assumed to be differentiable as much as necessary. For the choices $a_n = f^{(n)}(t)$ and $a_n = f^{(n)}(t)/n!$ we obtain, respectively, the confluent forms of the ε-algorithm ($w_{2k}^{(0)} = \varepsilon_{2k}(t)$) and of the ϱ-algorithm ($w_{2k}^{(0)} = \varrho_{2k}(t)$). See [3] for a review.

The confluent form of the E-algorithm [7] can also be included into this framework.

Let us now consider the case of an arbitrary topological vector space E. We shall denote by $\langle \cdot, \cdot \rangle$ the bilinear form of the duality. In practical applications $E = \boldsymbol{C}^p$.

Topological E-algorithm [7]: with

$$e_0 = S_n, \quad e_i = g_i(n) \text{ for } i \geqslant 1, \quad a_j^{(0)} = \langle y, \Delta S_{n+j-1} \rangle,$$

where y is an arbitrary element of the dual space E',

$$a_j^{(i)} = \langle y, \Delta g_i(n+j-1) \rangle \text{ for } i \geqslant 1, \quad b_0 = 1 \quad \text{and} \quad b_i = 0 \text{ for } i \geqslant 1,$$

we obtain

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

The Wimp generalisation of the topological E-algorithm [20], which uses a sequence (y_n) instead of a fixed y, can also be included into this framework. The same is true for a transformation due to Germain-Bonne [14] and for some other transformations described in [5].

Topological ε-algorithm [1]: it is a generalisation of Wynn's scalar ε-algorithm which, contrarily to the vector ε-algorithm, can be expressed in the above determinantal form. For

$$e_i = S_{n+i}, \quad a_j^{(i)} = \langle y, \Delta S_{n+i+j-1} \rangle \quad \text{and} \quad b_i = 1$$

we obtain

$$R_k = \varepsilon_{2k}^{(n)}.$$

H-algorithm [12]: with

$$e_i = S_{n+i}, \quad a_j^{(i)} = g_j(n+i) \quad \text{and} \quad b_i = 1$$

we have

$$R_k = H_k^{(n)}.$$

The g_j's are now scalars while, in the topological E-algorithm, they were elements of E.

Recursive projection algorithm [8]: for

$$e_0 = y, \quad e_i = x_i \text{ for } i \geqslant 1,$$
$$a_j^{(0)} = \langle z_j, y \rangle, \quad a_j^{(i)} = \langle z_j, x_i \rangle \text{ for } i \geqslant 1,$$
$$b_0 = 1 \quad \text{and} \quad b_i = 0 \text{ for } i \geqslant 1$$

we obtain

$$R_k = E_k.$$

The *compact recursive projection algorithm* (CRPA) is recovered with

$$e_i = x_{n+i}, \quad a_j^{(i)} = \langle z_j, x_{n+i} \rangle, \quad b_0 = 1 \quad \text{and} \quad b_i = 0 \text{ for } i \geqslant 1.$$

In that case $R_k = e_k^{(n)}$. With the same choice for the e_i's and the $a_j^{(i)}$'s but with $b_i = 1$ for $i \geqslant 0$ we obtain a variant of the CRPA, namely

$$R_k = \tilde{e}_k^{(n)}.$$

Vector composite sequence transformations and the related fixed point methods [17] can also be expressed as above.

3. The bordering method. Let us first recall the *bordering method* as described in [13]. Let A_k be a square regular matrix of dimension k, a_k a scalar, u_k and v_k, respectively, a column and a row vector of dimension k. We consider the bordered matrix of dimension $k+1$:

$$A_{k+1} = \begin{bmatrix} A_k & u_k \\ v_k & a_k \end{bmatrix}.$$

We have

$$A_{k+1}^{-1} = \begin{pmatrix} A_k^{-1} + A_k^{-1} u_k v_k A_k^{-1}/\beta_k & -A_k^{-1} u_k/\beta_k \\ -v_k A_k^{-1}/\beta_k & 1/\beta_k \end{pmatrix}$$

with $\beta_k = a_k - v_k A_k^{-1} u_k$.

Let z_k be the solution of $A_k z_k = d_k$. Then, using the expression of A_{k+1}^{-1}, we can obtain the solution of the bordered system

$$A_{k+1} z_{k+1} = d_{k+1} = \begin{pmatrix} d_k \\ f_k \end{pmatrix},$$

where f_k is a scalar, in the form

$$z_{k+1} = \begin{pmatrix} z_k \\ 0 \end{pmatrix} + \frac{f_k - v_k z_k}{\beta_k} \begin{pmatrix} -A_k^{-1} u_k \\ 1 \end{pmatrix}. \tag{4}$$

The use of this formula needs the computation and the storage of A_k^{-1}. This drawback can be avoided by computing $q_k = -A_k^{-1} u_k$ also by a bordering method. Let k be fixed and let us denote by $q_k^{(i)}$ the solution of $A_i q_k^{(i)} = -u_k^{(i)}$, where $u_k^{(i)}$ is the vector formed by the first i components of u_k. Thus $u_i^{(i)} = u_i$ and $q_i^{(i)} = q_i$ for all i. The bordering method for this system leads to the following scheme:

$$q_k^{(1)} = -u_k^{(1)}/A_1,$$

$$q_k^{(i+1)} = \begin{pmatrix} q_k^{(i)} \\ 0 \end{pmatrix} - \frac{u_{k,i+1} + v_i q_k^{(i)}}{a_i + v_i q_i^{(i)}} \begin{pmatrix} q_i^{(i)} \\ 1 \end{pmatrix}, \quad i = 1, \ldots, k-1, \tag{5}$$

where $u_{k,i+1}$ is the $(i+1)$-st component of u_k. We have

$$q_k^{(k)} = q_k = -A_k^{-1} u_k$$

in (4).

The bordering method (4), (5) can be applied directly to the system (2). Since $f_k = 0$ for $k \geqslant 1$ and $d_1 = 1$, (4) reduces to a simpler expression. The vector z_{k+1} thus obtained has components $\alpha_0, \ldots, \alpha_k$, and then R_k is computed by (1), which gives a recursive method for the ratios (3) for $k = 0, 1, \ldots$

For some particular algorithms such as the scalar and topological ε-algorithms, orthogonal polynomials and Padé approximants the general bordering method given above can be simplified due to the special structure of the linear system (2) and some improvements of the bordering method ([2], [6], [19]). Ratios of the form (3) and the bordering method are connected with determinantal identities used in deriving recursive algorithms for their implementation and with the Schur complement of a matrix [12].

4. Progressive forms of the algorithms. In Section 2, we saw that most of the ratios R_k also depend on the second index n. In these cases let us denote

them by $R_k^{(n)}$ and display them in a two-dimensional array:

$$\begin{array}{llll} R_0^{(0)} = S_0 & & & \\ R_0^{(1)} = S_1 & R_1^{(0)} & & \\ R_0^{(2)} = S_2 & R_1^{(1)} & R_2^{(0)} & \\ R_0^{(3)} = S_3 & R_1^{(2)} & R_2^{(1)} & R_3^{(0)} \\ \vdots & \vdots & \vdots & \vdots \end{array}$$

For all these cases there exist recursive algorithms which, starting from the initial column, compute recursively all the $R_k^{(n)}$ from the left to the right and from the top to the bottom. However, most of them suffer from numerical instability. A method for possibly avoiding this instability is to start from the first diagonal $R_0^{(0)}, R_1^{(0)}, R_2^{(0)}, \ldots$ instead of the first column. This trick was first used for the *qd-algorithm*, an algorithm very much similar to those described here, and it was called its *progressive form* [16] (see also [15]). The same trick can be used for all the algorithms of the form (3): we first compute the diagonal $R_0^{(0)}, R_1^{(0)}, R_2^{(0)}, \ldots$ by the bordering method (4), (5), and then the other $R_k^{(n)}$ are obtained by the progressive form of each algorithm. Of course, in using such a bordering method it is of main importance to check its numerical stability. However, in some applications the first diagonal $R_0^{(0)}, R_1^{(0)}, R_2^{(0)}, \ldots$ can be obtained in the closed form, thus avoiding this problem.

Let us now show an example of the advantage of using the progressive form of an algorithm instead of its normal form. We shall study the case of Wynn's scalar ε-algorithm [21] which fits in our framework after a slight modification. This algorithm is a recursive procedure for implementing Shanks' transformation [18]. We consider the ratios of determinants

$$\varepsilon_{2k}^{(n)} = \frac{\begin{vmatrix} S_n & \ldots & S_{n+k} \\ \Delta S_n & \ldots & \Delta S_{n+k} \\ \ldots & \ldots & \ldots \\ \Delta S_{n+k-1} & \ldots & \Delta S_{n+2k-1} \end{vmatrix}}{\begin{vmatrix} 1 & \ldots & 1 \\ \Delta S_n & \ldots & \Delta S_{n+k} \\ \ldots & \ldots & \ldots \\ \Delta S_{n+k-1} & \ldots & \Delta S_{n+2k-1} \end{vmatrix}} = e_k(S_n)$$

and $\varepsilon_{2k+1}^{(n)} = 1/e_k(\Delta S_n)$. These quantities are related by the normal form of the ε-algorithm:

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

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

As above these quantities are displayed in a two-dimensional array as follows:

$$
\begin{array}{lllll}
\varepsilon_{-1}^{(0)}=0 & & & & \\
 & \varepsilon_0^{(0)}=S_0 & & & \\
\varepsilon_{-1}^{(1)}=0 & & \varepsilon_1^{(0)} & & \\
 & \varepsilon_0^{(1)}=S_1 & & \varepsilon_2^{(0)} & \\
\varepsilon_{-1}^{(2)}=0 & & \varepsilon_1^{(1)} & & \\
 & \varepsilon_0^{(2)}=S_2 & & \varepsilon_2^{(1)} & \\
\varepsilon_{-1}^{(3)}=0 & & \varepsilon_1^{(2)} & & \\
\vdots & \varepsilon_0^{(3)}=S_3 & & \varepsilon_2^{(2)} & \\
 & \vdots & \varepsilon_1^{(3)} & & \\
 & & \vdots & \varepsilon_2^{(3)} & \\
 & & & \vdots &
\end{array}
$$

The normal form of the ε-algorithm allows us to compute all the $\varepsilon_k^{(n)}$ from the two initial columns. If the first diagonal is known, all these quantities can also be obtained by the progressive form of the algorithm:

$$\varepsilon_{k+1}^{(n+1)}=\varepsilon_{k+1}^{(n)}+[\varepsilon_{k+2}^{(n)}-\varepsilon_k^{(n+1)}]^{-1}, \quad k, n=0, 1, \ldots$$

For the particular case $S_n=(n+1)^{-1}$ it can be proved that, for all k and n,

$$\varepsilon_{2k}^{(n)}=1/(k+1)(n+k+1),$$

$$\varepsilon_{2k+1}^{(n)}=-(k+1)(k+2)(n+k+1)(n+k+2)/2,$$

thus providing a good example for checking the numerical stability of the algorithm.

If the normal form of the ε-algorithm is applied to this sequence, then the following conclusions hold:

In the computation of $\varepsilon_{2k+1}^{(n)}$,
$\varepsilon_{2k}^{(n)}$ and $\varepsilon_{2k}^{(n+1)}$ have $\log_{10}(n+k+2)$ common decimal digits;
$\varepsilon_{2k-1}^{(n+1)}$ and $[\Delta\varepsilon_{2k}^{(n)}]^{-1}$ have $-\log_{10}(1-2/k)$ common digits.

In the computation of $\varepsilon_{2k+2}^{(n)}$,
$\varepsilon_{2k+1}^{(n)}$ and $\varepsilon_{2k+1}^{(n+1)}$ have $-\log_{10}(2/(n+k+3))$ common decimal digits;
$\varepsilon_{2k}^{(n+1)}$ and $[\Delta\varepsilon_{2k+1}^{(n)}]^{-1}$ have opposite signs.

For the progressive form the conclusions are:

In the computation of $\varepsilon_{2k+1}^{(n)}$,
$\varepsilon_{2k}^{(n)}$ and $\varepsilon_{2k+2}^{(n-1)}$ have $\log_{10}(k+2)$ common digits;
$\varepsilon_{2k+1}^{(n-1)}$ and $[\varepsilon_{2k+2}^{(n-1)}-\varepsilon_{2k}^{(n)}]^{-1}$ have $-\log_{10}(1-2/(n+k))$ common digits.

In the computation of $\varepsilon_{2k+2}^{(n)}$,
$\varepsilon_{2k+1}^{(n)}$ and $\varepsilon_{2k+3}^{(n-1)}$ have $-\log_{10}(2/(k+3))$ common digits;
$\varepsilon_{2k+2}^{(n+1)}$ and $[\varepsilon_{2k+3}^{(n-1)}-\varepsilon_{2k+1}^{(n)}]^{-1}$ have opposite signs.

Thus all the computations conducted with the progressive form are more stable than the corresponding ones with the normal form. The programming of the progressive form can be realized by descending anti-diagonals as follows: let us assume that one such diagonal is known (for example: $\varepsilon_4^{(0)}$, $\varepsilon_3^{(1)}$, $\varepsilon_2^{(2)}$, $\varepsilon_1^{(3)}$ and $\varepsilon_0^{(4)}$); we add the next term of the first diagonal ($\varepsilon_5^{(0)}$); we compute the next descending anti-diagonal ($\varepsilon_4^{(1)}$, $\varepsilon_3^{(2)}$, $\varepsilon_2^{(3)}$, $\varepsilon_1^{(4)}$ and $\varepsilon_0^{(5)}$) replacing successively the terms of the old anti-diagonal by the new ones as soon as they are computed. The same computation scheme can be applied to the other algorithms described above.

In the ε-algorithm the quantities $\varepsilon_{2k+1}^{(n)}$ are intermediate computations which are not very useful. They can be eliminated, thus leading to the so-called *cross rule* for the ε-algorithm [22]:

$$E = C+[(N-C)^{-1}-(W-C)^{-1}+(S-C)^{-1}]^{-1}$$

with $C = \varepsilon_{2k}^{(n+1)}$, $N = \varepsilon_{2k}^{(n)}$, $S = \varepsilon_{2k}^{(n+2)}$, $W = \varepsilon_{2k-2}^{(n+2)}$ and $E = \varepsilon_{2k+2}^{(n)}$.

This normal cross rule can be replaced by its progressive form:

$$S = C-[(N-C)^{-1}-(W-C)^{-1}-(E-C)^{-1}]^{-1}$$

which is more stable than the normal one when applied to $S_n = (n+1)^{-1}$.

Let us mention that when two neighbouring quantities in the table are equal (or almost equal), it is possible to jump over the singularity (or the almost singularity which induces instability) by using some particular rules which can be obtained from Schur's formula [11] (see [3] for a review of such rules and [4] for the corresponding subroutines).

References

[1] C. Brezinski, *Généralisations de la transformation de Shanks, de la table de Padé et de l'ε-algorithme*, Calcolo 12 (1975), pp. 317–360.

[2] — *Computation of Padé approximants and continued fractions*, J. Comput. Appl. Math. 2 (1976), pp. 113–123.

[3] — *Accélération de la convergence en analyse numérique*, Lecture Notes in Math. 584, Springer, Heidelberg 1977.

[4] — *Algorithmes d'accélération de la convergence, Étude numérique*, Technip, Paris 1978.

[5] — *Sur le calcul de certains rapports de déterminants*, in: L. Wuytack (Ed.), *Padé Approximation and its Applications*, Lecture Notes in Math. 765, Springer, Heidelberg 1979.

[6] — *Padé type approximation and general orthogonal polynomials*, ISNM Vol. 50, Birkhäuser, Basel 1980.

[7] — *A general extrapolation algorithm*, Numer. Math. 35 (1980), pp. 175–187.

[8] — *Recursive interpolation, extrapolation and projection*, J. Comput. Appl. Math. 9 (1983), pp. 369–376.

[9] — *Convergence acceleration methods: the past decade*, ibidem 12–13 (1985), pp. 19–36.

[10] C. Brezinski, *Composite sequence transformations*, Numer. Math. 46 (1985), pp. 311–321.
[11] – *Other manifestations of the Schur complement*, Linear Algebra Appl. 111 (1988), pp. 231–247.
[12] – and H. Sadok, *Vector sequence transformations and fixed point methods*, in: C. Taylor et al. (Eds.), *Numerical Methods in Laminar and Turbulent Flow*, Pineridge Press, Swansea 1987.
[13] V. N. Faddeeva, *Computational Methods of Linear Algebra*, Dover, New York 1959.
[14] B. Germain-Bonne, *Estimation de la limite de suites et formalisation de procédés d'accélération de convergence*, Thesis, Université de Lille 1, 1978.
[15] P. Henrici, *Applied and Computational Complex Analysis*, Wiley, Chichester 1974.
[16] H. Rutishauser, *Stabile Sonderfälle des Quotienten-Differenzen-Algorithmus*, Numer. Math. 5 (1963), pp. 95–112.
[17] H. Sadok, *Accélération de la convergence de suites vectorielles et méthodes de point fixe*, Thesis, Université de Lille 1, 1988.
[18] D. Shanks, *Non-linear transformations of divergent and slowly convergent sequences*, J. Math. Phys. 34 (1955), pp. 1–42.
[19] W. F. Trench, *An algorithm for the inversion of finite Hankel matrices*, SIAM J. Appl. Math. 13 (1965), pp. 1102–1107.
[20] J. Wimp, *Sequence Transformations and their Applications*, Academic Press, New York 1981.
[21] P. Wynn, *On a device for computing the $e_m(S_n)$ transformation*, MTAC 10 (1956), pp. 91–96.
[22] – *Upon systems of recursions which obtain among others the quotients of the Padé table*, Numer. Math. 8 (1966), pp. 264–269.
[23] – *Upon some continuous prediction algorithms, I*, Calcolo 9 (1972), pp. 177–234.

CLAUDE BREZINSKI
LABORATOIRE D'ANALYSE NUMÉRIQUE ET D'OPTIMISATION
UFR IEEA – M3
UNIVERSITÉ DE LILLE FLANDRES-ARTOIS
59655 VILLENEUVE D'ASCQ – CEDEX, FRANCE

Received on 1987.10.01