Cross rules and non-Abelian lattice equations for the discrete and confluent non-scalar ε-algorithms

2010 J. Phys. A: Math. Theor. 43 205201

(http://iopscience.iop.org/1751-8121/43/20/205201)

IOP PUBLISHING JOURNAL OF PHYSICS A: MATHEMATICAL AND THEORETICAL

J. Phys. A: Math. Theor. **43** (2010) 205201 (11pp) doi:10.1088/1751-8113/43/20/205201

Cross rules and non-Abelian lattice equations for the discrete and confluent non-scalar ε-algorithms

C Brezinski

Laboratoire Paul Painlevé, UMR CNRS 8524, UFR de Mathématiques Pures et Appliquées, Université des Sciences et Technologies de Lille, 59655–Villeneuve d'Ascq cedex, France

E-mail: Claude.Brezinski@univ-lille1.fr

Received 8 February 2010, in final form 24 March 2010
Published 23 April 2010
Online at stacks.iop.org/JPhysA/43/205201

Abstract

In this paper, we give the cross rules of the discrete and confluent vector, topological and matrix ε-algorithms. Then, from the rules of these confluent algorithms, we derive non-Abelian lattice equations, in particular some extensions of the Lotka–Volterra system, in the style of the equation related to the confluent form of the scalar ε-algorithm.

PACS numbers: 02.30.Ik, 02.30.Lt, 02.30.Mv

1. Introduction and motivations

In this paper, we are discussing two cases: the case of a sequence of vectors or matrices (S_n), and the case of a vector or a matrix function $f(t)$. A *sequence transformation* transforms (S_n) into a new sequence (T_n) which, under some assumptions, converges to $S = \lim_{n\to\infty} S_n$ faster than (S_n), that is such that $\lim_{n\to\infty} \|T_n - S\|/\|S_n - S\| = 0$. Similarly, a *function transformation* transforms f into a new function g which, under some assumptions, converges to $S = \lim_{t\to\infty} f(t)$ faster than f, that is such that $\lim_{t\to\infty} \|g(t) - S\|/\|f(t) - S\| = 0$. For many transformations, the new sequence (T_n) and the new function g depend on an index k, and are given as a ratio of determinants of dimension related to k; see [9] for a review. Since determinants cannot be easily computed, a recursive algorithm for implementing the sequence or the function transformation is needed. Such an algorithm is called a *vector* (or *matrix) extrapolation algorithm*; it is a *discrete* vector extrapolation algorithm in the case of a sequence, and a *confluent* one in the case of a function. In these algorithms, a division by zero could sometimes occur and, for some of them, it is possible to derive *singular rules* which allow us to jump over the singularity (or over several of them arising consecutively), and to continue the computation (in the language of discrete systems, the singularity is said to be *confined*). This is the point of view of numerical analysis on this topic.

But there is another point of view, which comes from *discrete systems*. A discrete system is characterized by a (usually nonlinear) difference equation obtained by discretization of a

(usually nonlinear) partial differential equation. It is important that the discretized scheme conserves the quantities that are conserved by the partial differential equation itself. A crucial character is the *integrability* of the equation. Although this term has not yet received a completely satisfactory definition, it can be understood either as the ability to express the solution in a close form, or as the confinement of singularities in finite domains, or both, two features which exist for the extrapolation algorithms mentioned above [3, 12, 18, 19]. Integrability is a rare phenomenon, and dynamical systems are typically non-integrable. This is the reason why extrapolation algorithms are important in the domain of integrable discrete systems; see, for example, [12, 16, 17]. There exist many connections between iterative procedures used in numerical analysis and dynamical systems; see [4], [5, chapter 10], [7] and [10] for a review.

In many extrapolation algorithms, two types of vectors are usually computed: half of them are directly related to the transformation to be implemented, while the other ones are intermediate computations. The *cross rule* of an extrapolation algorithm is a recurrence relation which allows us to implement the transformation by computing only the vectors directly related to it, without computing the intermediate vectors (or vice versa). Many new cross rules of discrete and confluent scalar extrapolation algorithms were derived in [8].

Among extrapolation algorithms, the most popular one is the scalar ε-algorithm devised by Wynn [21]. It allows us to implement a scalar sequence transformation due to Shanks [20]. Wynn extended it to the vector and matrix cases in [23]. Since the vector ε-algorithm was obtained directly from the rules of the scalar one and, thus, was lacking an algebraic support, a vector transformation similar to Shanks' (or, more generally, a transformation for elements of a vector space), and the corresponding ε-algorithm, called *topological*, were derived in [1]. Wynn also proposed a confluent ε-algorithm [22], and a confluent topological ε-algorithm was given in [2]. Other non-scalar generalizations of the ε-algorithm also exist; see [3] for a review.

The aim of this paper is twofold. First, we derive the cross rules of the discrete and confluent vector, topological and matrix ε-algorithms. Such cross rules are useful in numerical analysis for the implementation of sequence and function transformations, and also in the proof of some theoretical results related to them; moreover, as explained above and in [3, 12, 18, 19], they are involved in the integrability of discrete systems. Then, we will show how the rules of these confluent algorithms lead us to coupled non-Abelian lattice equations, in particular to some extensions of the Lotka–Volterra system, in the style of the equation related to the confluent form of the scalar ε-algorithm. Obviously, the exact relations between non-scalar convergence acceleration algorithms and non-commutative integrable systems (described, for example, in [14]) remain to be discovered.

2. The case of dimension 2

Let (V_n) be a sequence of vectors in $\mathbb{R}^2$, and set $V_n = (x_n, y_n)^T$. We convert this sequence into the sequence of the complex numbers (S_n) with $S_n = x_n + \mathrm{i}y_n$, where $\mathrm{i} = \sqrt{-1}$. Since the cross rule of any discrete scalar extrapolation algorithm is valid for sequences of complex numbers, we apply it to (S_n) and, then, we separate the real and imaginary parts of the results thus leading to a cross rule in dimension 2. This idea was already exploited in [8].

The same idea applies to two-dimensional functions of a real variable. Let $v(t) = (x(t), y(t))^T$. We convert it into the complex function $f(t) = x(t) + \mathrm{i}y(t)$, apply any confluent scalar function transformation to it and, then, separate the real and imaginary parts of the results. This idea was already used in [6] for the treatment of the Gibbs phenomenon in Fourier and polynomials series, and in [8] to extend the usual Lotka–Volterra equation to

the case of two-dimensional vectors as follows. The confluent form of the scalar ε-algorithm obeys the rule

$$\varepsilon_{k+1}(t) = \varepsilon_{k-1}(t) + (\varepsilon_k'(t))^{-1}, \qquad k = 0, 1, \ldots$$

with $\varepsilon_{-1}(t) = 0$ and $\varepsilon_0(t) = f(t)$. Setting $M_k(t) = \varepsilon_k'(t)$, differentiating the preceding relation and applying the Miura transformation $N_k(t) = M_k(t)M_{k+1}(t)$ transforms this rule into the Lotka–Volterra equation

$$N_k'(t) = N_k(t)[N_{k-1}(t) - N_{k+1}(t)].$$

Assume now that we want to write such an equation for the two-dimensional vectors $(R_k(t), I_k(t))^T$. Let us write them as $N_k(t) = R_k(t) + \mathrm{i}I_k(t)$. Then, separating the real and imaginary parts in the Lotka–Volterra equation above leads to the coupled system

$$\begin{aligned} R_k'(t) &= R_k(t)[R_{k-1}(t) - R_{k+1}(t)] - I_k(t)[I_{k-1}(t) - I_{k+1}(t)] \\ I_k'(t) &= R_n(t)[I_{k-1}(t) - I_{k+1}(t)] + I_k(t)[R_{k-1}(t) - R_{k+1}(t)]. \end{aligned}$$

This system was given in [15] without explaining its derivation which was presented in [24].

3. Discrete algorithms

3.1. The vector ε-algorithm

The vector ε-algorithm was obtained by Wynn [22]. Its rule is

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

with $\varepsilon_{-1}^{(n)} = 0 \in \mathbb{C}^p$, $\varepsilon_0^{(n)} = S_n \in \mathbb{C}^p$, and where the inverse of a vector $y \in \mathbb{C}^p$ is defined as

$$z^{-1} = \bar{z}/(z, z) \in \mathbb{C}^p,$$

where $(u, v) = u^T\bar{v}$ is the inner product of the vectors u and v in $\mathbb{C}^p$.

Since the rule of the vector ε-algorithm is the same as the rule of the scalar one, and since $(y^{-1})^{-1} = y$, its cross rule is the same. Moreover, for any vector algorithm of the form (called the γ-algorithm)

$$\gamma_{k+1}^{(n)} = \gamma_{k-1}^{(n+1)} + a_k^{(n)}\left(\gamma_k^{(n+1)} - \gamma_k^{(n)}\right)^{-1}, \qquad k, n = 0, 1, \ldots$$

with, for all n, $\gamma_{-1}^{(n)} = 0$ and $\gamma_0^{(n)} = S_n$, and where the $a_k^{(n)}$'s are numbers, the cross rule is [8]

Property 1. *The cross rule of the vector γ-algorithm is*

$$\begin{aligned} &a_{k+1}^{(n-1)}\left(\gamma_{k+2}^{(n-1)} - \gamma_k^{(n)}\right)^{-1} + a_{k-1}^{(n)}\left(\gamma_{k-2}^{(n+1)} - \gamma_k^{(n)}\right)^{-1} \\ &\qquad = a_k^{(n)}\left(\gamma_k^{(n+1)} - \gamma_k^{(n)}\right)^{-1} + a_k^{(n-1)}\left(\gamma_k^{(n-1)} - \gamma_k^{(n)}\right)^{-1}, \end{aligned}$$

with, for all n, $\gamma_{-2}^{(n)} = (\infty, \ldots, \infty)^T \in \mathbb{C}^p$ (so the inverse of any infinity vector is the zero vector), $\gamma_{-1}^{(n)} = 0 \in \mathbb{C}^p$, $\gamma_0^{(n)} = S_n \in \mathbb{C}^p$ and $\gamma_1^{(n)} = a_0^{(n)}(\Delta S_n)^{-1} \in \mathbb{C}^p$.

After inversion, this cross rule allows us to compute $\gamma_{k+2}^{(n-1)}$ from the four other vectors, starting from the initial conditions.

3.2. The topological ε-algorithm

Let E be a vector space on $\mathbb{R}$ or $\mathbb{C}$, and E^* its dual, that is the vector space of linear forms on E. Let (S_n) be a sequence of elements of E. The duality product between E and E^* is denoted by $\langle z, y\rangle$, where $y \in E$ and $z \in E^*$.

The topological *e*–transformation is defined by

$$e_k(S_n) = \frac{\begin{vmatrix} S_n & \cdots & S_{n+k} \\ \langle z, \Delta S_n\rangle & \cdots & \langle z, \Delta S_{n+k}\rangle \\ \vdots & & \vdots \\ \langle z, \Delta S_{n+k-1}\rangle & \cdots & \langle z, \Delta S_{n+2k-1}\rangle \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ \langle z, \Delta S_n\rangle & \cdots & \langle z, \Delta S_{n+k}\rangle \\ \vdots & & \vdots \\ \langle z, \Delta S_{n+k-1}\rangle & \cdots & \langle z, \Delta S_{n+2k-1}\rangle \end{vmatrix}}.$$

These vectors can be recursively computed by the topological ε-algorithm which obeys the rules

$$\varepsilon_{2k+1}^{(n)} = \varepsilon_{2k-1}^{(n+1)} + \frac{z}{\langle z, \varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)}\rangle}$$

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

with $\varepsilon_{-1}^{(n)} = 0 \in E^*$, $\varepsilon_0^{(n)} = S_n \in E$, and where z is any nonzero element of E^* such that the denominators do not vanish. Thus, $\varepsilon_{2k+1}^{(n)} \in E^*$, and $\varepsilon_{2k+2}^{(n)} \in E$, and it holds

$$\varepsilon_{2k}^{(n)} = e_k(S_n), \qquad \varepsilon_{2k+1}^{(n)} = [e_k(\Delta S_n)]^{-1} = \frac{z}{\langle z, e_k(\Delta S_n)\rangle}.$$

Of course, in most applications, E and E^* will be $\mathbb{R}^p$ or $\mathbb{C}^p$ and, in such cases, the duality product reduces to the ordinary scalar or inner product of two vectors. However, it is easier to understand the different role played by the elements with an odd and an even lower index by treating the more general setting.

For obtaining this algorithm, it is necessary to define the inverse $(y^{-1}, z^{-1}) \in E^* \times E$ (in this order) of a couple of elements $(z, y) \in E^* \times E$ (in this order), instead of the inverse of one single element of E or E^*. This definition is

$$y^{-1} = \frac{z}{\langle z, y\rangle} \in E^*, \qquad z^{-1} = \frac{y}{\langle z, y\rangle} \in E.$$

We have

$$(y^{-1})^{-1} = y, \qquad (z^{-1})^{-1} = z, \qquad \langle y^{-1}, y\rangle = 1, \qquad \langle z, z^{-1}\rangle = 1.$$

We will say that y^{-1} (respectively z^{-1}) is the inverse of y (respectively z) with respect to z (respectively y), or in the couple (z, y).

Thus, for writing the topological ε-algorithm as

$$\varepsilon_{k+1}^{(n)} = \varepsilon_{k-1}^{(n+1)} + \left(\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}\right)^{-1}$$

one has to consider that

$$\left(\varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)}\right)^{-1} = \frac{z}{\langle z, \varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)}\rangle}$$

is the inverse of $\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}$ in the couple $\left(z,\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}\right)$, that is with respect to z, while

$$\left(\varepsilon_{2k+1}^{(n+1)}-\varepsilon_{2k+1}^{(n)}\right)^{-1}=\frac{z^{-1}}{\left\langle\varepsilon_{2k+1}^{(n+1)}-\varepsilon_{2k+1}^{(n)},z^{-1}\right\rangle}=\frac{\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}}{\left\langle\varepsilon_{2k+1}^{(n+1)}-\varepsilon_{2k+1}^{(n)},\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}\right\rangle}$$

is the inverse of $\varepsilon_{2k+1}^{(n+1)}-\varepsilon_{2k+1}^{(n)}$ in the couple $\left(\varepsilon_{2k+1}^{(n+1)}-\varepsilon_{2k+1}^{(n)},z^{-1}\right)$, that is with respect to z^{-1}. It is also the inverse of $\varepsilon_{2k+1}^{(n+1)}-\varepsilon_{2k+1}^{(n)}$ in the couple $\left(\varepsilon_{2k+1}^{(n+1)}-\varepsilon_{2k+1}^{(n)},\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}\right)$, that is with respect to $\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}$. Indeed, we also have

$$z^{-1}=\frac{\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}}{\left\langle z,\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}\right\rangle},$$

and the inverse of z^{-1} is always taken in the couple $\left(\left[\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}\right]^{-1},z^{-1}\right)$, that is with respect to $\left[\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}\right]^{-1}$ in order that $(z^{-1})^{-1}=z$.

Since the inverse of the inverse of an element of E or E^* is the element itself, the cross rule given in property 1 is valid for the topological ε-algorithm with $a_k^{(n)}=1$ for all k and n. However, although their form is the same, there is an important difference between these two cross rules because, for the topological ε-algorithm, the couple in which each inverse has to be taken (which is not the same for the cross rule and for the rule of the algorithm) must be clearly pointed out. Thus, we will give the complete proof.

Property 2. *The cross rule of the topological ε-algorithm is*

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

with, for all n, $\varepsilon_{-2}^{(n)}=\infty\in E$, $\varepsilon_{-1}^{(n)}=0\in E^*$, $\varepsilon_0^{(n)}=S_n\in E$ and $\varepsilon_1^{(n)}=(\Delta S_n)^{-1}\in E^*$.

Proof. The first rule of the algorithm writes

$$\left\langle z,\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}\right\rangle\left(\varepsilon_{2k+1}^{(n)}-\varepsilon_{2k-1}^{(n+1)}\right)=z.$$

Inverting z with respect to $\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}$ gives

$$\frac{1}{\left\langle z,\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}\right\rangle}\left(\varepsilon_{2k+1}^{(n)}-\varepsilon_{2k-1}^{(n+1)}\right)^{-1}=z^{-1}=\frac{\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}}{\left\langle z,\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}\right\rangle},$$

that is $\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}=\left(\varepsilon_{2k+1}^{(n)}-\varepsilon_{2k-1}^{(n+1)}\right)^{-1}$. Applying the forward difference operator Δ to the second rule of the algorithm gives

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

which, after replacement using the preceding relation, is the cross rule for the odd indices

$$\left(\varepsilon_{2k+3}^{(n)}-\varepsilon_{2k+1}^{(n+1)}\right)^{-1}+\left(\varepsilon_{2k-1}^{(n+2)}-\varepsilon_{2k+1}^{(n+1)}\right)^{-1}=\left(\varepsilon_{2k+1}^{(n+2)}-\varepsilon_{2k+1}^{(n+1)}\right)^{-1}+\left(\varepsilon_{2k+1}^{(n)}-\varepsilon_{2k+1}^{(n+1)}\right)^{-1}.$$

Similarly, for the second rule, we have

$$\left\langle\varepsilon_{2k+1}^{(n+1)}-\varepsilon_{2k+1}^{(n)},\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}\right\rangle\left(\varepsilon_{2k+2}^{(n)}-\varepsilon_{2k}^{(n+1)}\right)=\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}.$$

Inverting $\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}$ with respect to $\varepsilon_{2k+1}^{(n+1)}-\varepsilon_{2k+1}^{(n)}$ (and not with respect to z as in the first rule of the algorithm) since the inverse of $\varepsilon_{2k+1}^{(n+1)}-\varepsilon_{2k+1}^{(n)}$ is taken with respect to $\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}$, we obtain

$$\frac{1}{\left\langle\varepsilon_{2k+1}^{(n+1)}-\varepsilon_{2k+1}^{(n)},\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}\right\rangle}\left(\varepsilon_{2k+2}^{(n)}-\varepsilon_{2k}^{(n+1)}\right)^{-1}=\left(\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}\right)^{-1}$$

$$=\frac{\varepsilon_{2k+1}^{(n+1)}-\varepsilon_{2k+1}^{(n)}}{\left\langle\varepsilon_{2k+1}^{(n+1)}-\varepsilon_{2k+1}^{(n)},\varepsilon_{2k}^{(n+1)}-\varepsilon_{2k}^{(n)}\right\rangle}$$

that is $\left(\varepsilon_{2k+2}^{(n)} - \varepsilon_{2k}^{(n+1)}\right)^{-1} = \varepsilon_{2k+1}^{(n+1)} - \varepsilon_{2k+1}^{(n)}$. Applying the operator Δ to the first rule of the algorithm gives

$$\Delta\varepsilon_{2k+1}^{(n)} - \Delta\varepsilon_{2k-1}^{(n+1)} = \Delta\left(\left[\Delta\varepsilon_{2k}^{(n)}\right]^{-1}\right).$$

Then, by replacement, we finally get the cross rule for the even indices

$$\left(\varepsilon_{2k+2}^{(n)} - \varepsilon_{2k}^{(n+1)}\right)^{-1} + \left(\varepsilon_{2k-2}^{(n+2)} - \varepsilon_{2k}^{(n+1)}\right)^{-1} = \left(\varepsilon_{2k}^{(n+2)} - \varepsilon_{2k}^{(n+1)}\right)^{-1} + \left(\varepsilon_{2k}^{(n)} - \varepsilon_{2k}^{(n+1)}\right)^{-1}.$$ □

Remark 1. There is an important difference between the cross rules of the vector and the topological ε-algorithm. In the vector case, the cross rule involves only vectors with lower indices of the same parity and, thus, it allows, by inversion, us to compute directly the vector with the highest lower index. In the topological case, although the cross rule seems to involve elements of E or E^* with lower indexes of the same parity, it, in fact, involves elements with a different parity due to the definition of the inverse of a couple of elements in $E^* \times E$. Thus, this cross rule does not allow us to compute the element with the highest lower index that it contains without using elements with indices of the opposite parity.

In fact, since $\left(\varepsilon_{2k+1}^{(n)} - \varepsilon_{2k-1}^{(n+1)}\right)^{-1} = \varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)}$ and $\left(\varepsilon_{2k+2}^{(n)} - \varepsilon_{2k}^{(n+1)}\right)^{-1} = \varepsilon_{2k+1}^{(n+1)} - \varepsilon_{2k+1}^{(n)}$, this cross rule is nothing else than a simple rewriting of the rule of the topological ε-algorithm for the upper indices n and $n+1$, followed by a subtraction between them.

3.3. The matrix ε-algorithm

Let us now assume that (S_n) is a sequence of square matrices. The matrix ε-algorithm is defined by the same rule as the vector one, after replacing the inverse of a vector by the ordinary inverse of a matrix. Thus, the same cross rule also holds.

4. Confluent algorithms

Confluent algorithms are obtained from the discrete ones by setting $\varepsilon_k^{(n)} = \varepsilon_k(t + nh)$, and then letting h tend to zero; for details, see [9].

4.1. The confluent vector ε-algorithm

The confluent vector ε-algorithm is defined by the rule

$$\varepsilon_{k+1}(t) = \varepsilon_{k-1}(t) + [\varepsilon_k'(t)]^{-1},$$

with $\varepsilon_{-1}(t) = 0 \in \mathbb{C}^p$, $\varepsilon_0(t) = f(t) \in \mathbb{C}^p$, and where the inverse of a vector is defined as above, that is $[\varepsilon_k'(t)]^{-1} = \bar{\varepsilon}_k'(t)/(\varepsilon_k'(t), \varepsilon_k'(t))$.

Remark 2. If we discretize this differential equation using the Euler-type scheme given in [13], we get

$$\left(\varepsilon_k^{n+1} - \varepsilon_k^{(n)}\right)^{-1} = \tau\left(\varepsilon_{k+1}^{(n)} - \varepsilon_{k-1}^{(n+1)}\right),$$

where $\varepsilon_k^{(n)}$ is the approximate solution $\varepsilon_k(t)$ at the point $t = n\tau$. Thus, the rule of the confluent vector ε-algorithm is recovered with a unit time step $\tau = 1$. With a different value of τ, the same vectors $\varepsilon_{2k}^{(n)}$ are obtained, while the vectors $\varepsilon_{2k+1}^{(n)}$ are divided by τ.

For this algorithm, the cross rule is similar to the cross rule of the confluent scalar ε-algorithm

Property 3. *The cross rule of the confluent vector ε-algorithm is*

$$(\varepsilon_{k+2}(t) - \varepsilon_k(t))^{-1} + (\varepsilon_{k-2}(t) - \varepsilon_k(t))^{-1} = ([\varepsilon'_k(t)]^{-1})',$$

with, for all t, $\varepsilon_{-2}(t) = \infty \in \mathbb{C}^p$ *(so, the inverse of any infinity vector is the zero vector),* $\varepsilon_{-1}(t) = 0 \in \mathbb{C}^p$, $\varepsilon_0(t) = f(t) \in \mathbb{C}^p$ *and* $\varepsilon_1(t) = (f'(t))^{-1} \in \mathbb{C}^p$.

Proof. Differentiating the rule of the algorithm gives

$$\varepsilon'_{k+1}(t) = \varepsilon'_{k-1}(t) + ([\varepsilon'_k(t)]^{-1})'. \tag{1}$$

But, since $(y^{-1})^{-1} = y$,

$$\varepsilon'_{k+1}(t) = (\varepsilon_{k+2}(t) - \varepsilon_k(t))^{-1},$$

and a similar relation for $\varepsilon'_{k-1}(t)$, which yields the result. □

After inversion, this cross rule allows us to determine $\varepsilon_{k+2}(t)$ from the two other vector functions, starting from the initial conditions.

Let us now discuss the link between the confluent vector ε-algorithm and discrete systems. Setting $M_k = \varepsilon'_k$ (for simplicity, the variable t is suppressed, and the functions are in $\mathbb{R}^p$), relation (1) writes

$$M_{k+1} - M_{k-1} = \left(M_k^{-1}\right)' = \left(\frac{M_k}{(M_k, M_k)}\right)',$$

that is

$$(M_k, M_k)^2[M_{k+1} - M_{k-1}] = (M_k, M_k)M'_k - (M_k, M_k)'M_k = (M_k, M_k)M'_k - 2(M'_k, M_k)M_k.$$

Now, setting $N_k = (M_k, M_{k+1})$, and multiplying scalarly the preceding relation by M_k, gives

$$(M_k, M_k)[N_k - N_{k-1}] = (M'_k, M_k) - (M_k, M_k)' = -(M'_k, M_k) = -\tfrac{1}{2}(M_k, M_k)'.$$

It does not seem possible to eliminate M_k for obtaining a Lotka–Volterra equation. However, if we set $P_k = (M_k, M_k)$, then $P'_k = 2(M'_k, M_k)$, and the preceding relation produces the following kind of Lotka–Volterra equation:

$$P'_k = 2P_k[N_{k-1} - N_k].$$

A differential equation relating $N'_k = (M'_k, M_{k+1}) + (M_k, M'_{k+1})$ to N_k and P_k does not seem to hold since it will need to introduce new scalar products.

4.2. The confluent topological ε-algorithm

The confluent topological *e*-transformation is defined by

$$e_k(f(t)) = \frac{\begin{vmatrix} f(t) & \cdots & f^{(k)}(t) \\ \langle z, f'(t)\rangle & \cdots & \langle z, f^{(k+1)}(t)\rangle \\ \vdots & & \vdots \\ \langle z, f^{(k-1)}(t)\rangle & \cdots & \langle z, f^{(2k-1)}(t)\rangle \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ \langle z, f'(t)\rangle & \cdots & \langle z, f^{(k+1)}(t)\rangle \\ \vdots & & \vdots \\ \langle z, f^{(k-1)}(t)\rangle & \cdots & \langle z, f^{(2k-1)}(t)\rangle \end{vmatrix}}.$$

These vector functions can be recursively computed by the confluent topological ε-algorithm proposed in [2]. It is based on the inverse of a couple of elements of $E^* \times E$ as defined above. It is as follows:

$$\varepsilon_{2k+1}(t) = \varepsilon_{2k-1}(t) + \frac{z}{\langle z, \varepsilon'_{2k}(t)\rangle} \in E^*$$

$$\varepsilon_{2k+2}(t) = \varepsilon_{2k}(t) + \frac{\varepsilon'_{2k}(t)}{\langle \varepsilon'_{2k+1}(t), \varepsilon'_{2k}(t)\rangle} \in E,$$

with $\varepsilon_{-1}(t) = 0 \in E^*$, $\varepsilon_0(t) = f(t) \in E$, and where $z \in E^*$ is any nonzero element of E^* such that the denominators do not vanish. It holds

$$\varepsilon_{2k}(t) = e_k(f(t)), \qquad \varepsilon_{2k+1}(t) = [e_k(f'(t))]^{-1} = \frac{z}{\langle z, e_k(f'(t))\rangle}.$$

For writing this algorithm under the form

$$\varepsilon_{k+1}(t) = \varepsilon_{k-1}(t) + (\varepsilon'_k(t))^{-1}$$

one has to consider that

$$(\varepsilon'_{2k}(t))^{-1} = \frac{z}{\langle z, \varepsilon'_{2k}(t)\rangle}$$

is the inverse of $\varepsilon'_{2k}(t)$ in the couple $(z, \varepsilon'_{2k}(t))$, that is with respect to z, while

$$(\varepsilon'_{2k+1}(t))^{-1} = \frac{\varepsilon'_{2k}(t)}{\langle \varepsilon'_{2k+1}(t), \varepsilon'_{2k}(t)\rangle}$$

is the inverse of $\varepsilon'_{2k+1}(t)$ in the couple $(\varepsilon'_{2k+1}(t), z^{-1})$, that is with respect to z^{-1}. It is also its inverse in the couple $(\varepsilon'_{2k+1}(t), \varepsilon'_{2k}(t))$, that is with respect to $\varepsilon'_{2k}(t)$. Indeed, we also have

$$z^{-1} = \frac{\varepsilon'_{2k}(t)}{\langle z, \varepsilon'_{2k}(t)\rangle},$$

and the inverse of z^{-1} is always taken in the couple $([\varepsilon'_{2k}(t)]^{-1}, z^{-1})$ in order that $(z^{-1})^{-1} = z$.

Contrarily to the topological ε-algorithm where z had to be fixed, the element $z \in E^*$ in the confluent topological ε-algorithm can depend on t, a property which was never noted before.

Similarly to the confluent vector ε-algorithm, the cross rule given in property 3 still holds for the confluent topological ε-algorithm. However, although their form is the same, there is an important difference between these two cross rules because, for the confluent topological ε-algorithm, the couple in which each inverse has to be taken (which is not the same for the cross rule and for the rule of the algorithm) must be clearly understood. Thus, we will give the complete proof.

Property 4. *The cross rule of the confluent topological ε-algorithm is*

$$(\varepsilon_{k+2}(t) - \varepsilon_k(t))^{-1} + (\varepsilon_{k-2}(t) - \varepsilon_k(t))^{-1} = ([\varepsilon'_k(t)]^{-1})',$$

with, for all t, $\varepsilon_{-2}(t) = \infty \in E$, $\varepsilon_{-1}(t) = 0 \in E^*$, $\varepsilon_0(t) = f(t) \in E$ and $\varepsilon_1(t) = (f'(t))^{-1} \in E^*$.

Proof. from the first rule of the algorithm, we have (the variable t has been suppressed for simplicity)

$$\langle z, \varepsilon'_{2k}\rangle(\varepsilon_{2k+1} - \varepsilon_{2k-1}) = z.$$

Inverting z with respect to ε'_{2k} gives

$$\frac{1}{\langle z, \varepsilon'_{2k}\rangle}(\varepsilon_{2k+1} - \varepsilon_{2k-1})^{-1} = z^{-1} = \frac{\varepsilon'_{2k}}{\langle z, \varepsilon'_{2k}\rangle},$$

that is $\varepsilon'_{2k} = (\varepsilon_{2k+1} - \varepsilon_{2k-1})^{-1}$. Differentiating now the second relation of the algorithm gives

$$\varepsilon'_{2k+2} - \varepsilon'_{2k} = ([\varepsilon'_{2k+1}]^{-1})',$$

which proves the cross rule for the odd indices after replacement of the derivatives on the left-hand side:

$$(\varepsilon_{2k+3} - \varepsilon_{2k+1})^{-1} + (\varepsilon_{2k-1} - \varepsilon_{2k+1})^{-1} = ([\varepsilon'_{2k+1}]^{-1})'.$$

Similarly, from the second rule of the algorithm,

$$\langle \varepsilon'_{2k+1}, \varepsilon'_{2k} \rangle (\varepsilon_{2k+2} - \varepsilon_{2k}) = \varepsilon'_{2k}.$$

Inverting ε'_{2k} with respect to ε'_{2k+1} (and not with respect to z as in the first rule of the algorithm since the inverse of ε'_{2k+1} is taken with respect to ε'_{2k} in this second rule of the algorithm) gives

$$\frac{1}{\langle \varepsilon'_{2k+1}, \varepsilon'_{2k} \rangle} (\varepsilon_{2k+2} - \varepsilon_{2k})^{-1} = (\varepsilon'_{2k})^{-1} = \frac{\varepsilon'_{2k+1}}{\langle \varepsilon'_{2k+1}, \varepsilon'_{2k} \rangle},$$

that is $\varepsilon'_{2k+1} = (\varepsilon_{2k+2} - \varepsilon_{2k})^{-1}$. Differentiating the first relation of the algorithm, we obtain

$$\varepsilon'_{2k+1} - \varepsilon'_{2k-1} = ([\varepsilon'_{2k}]^{-1})',$$

which proves the cross rule for the even indices after replacement of the derivatives on the left-hand side:

$$(\varepsilon_{2k+2} - \varepsilon_{2k})^{-1} + (\varepsilon_{2k-2} - \varepsilon_{2k})^{-1} = ([\varepsilon'_{2k}]^{-1})'. \qquad \square$$

Remark 3. The important difference mentioned in Remark 1 is still valid for the cross rules of the confluent vector and topological ε-algorithms.

Let us now assume that $E = E^* = \mathbb{R}^p$. Thus, the duality product $\langle \cdot, \cdot \rangle$ becomes the usual scalar product $(\cdot, \cdot)$. Differentiating the first rule of the algorithm gives

$$(z, \varepsilon'_{2k})^2 [\varepsilon'_{2k-1} - \varepsilon'_{2k+1}] = (z, \varepsilon''_{2k}) z.$$

Multiplying scalarly this relation by ε'_{2k} and setting $M_k = (z, \varepsilon'_k)$ and $P_k = (\varepsilon'_k, \varepsilon'_{k+1})$ yields

$$M'_{2k} = M_{2k}[P_{2k-1} - P_{2k}]. \tag{2}$$

Similarly, differentiating the second rule of the algorithm gives

$$(\varepsilon'_{2k+1}, \varepsilon'_{2k})^2 [\varepsilon'_{2k+2} - \varepsilon'_{2k}] = \varepsilon''_{2k} (\varepsilon'_{2k+1}, \varepsilon'_{2k}) - \varepsilon'_{2k} (\varepsilon'_{2k+1}, \varepsilon'_{2k})',$$

that is

$$P_{2k}^2 [M_{2k+2} - M_{2k}] = M'_{2k} P_{2k} - M_{2k} P'_{2k}.$$

Replacing M'_{2k} by its expression (2) produces

$$M_{2k} P'_{2k} = P_{2k}[M_{2k} P_{2k-1} - M_{2k+2} P_{2k}]. \tag{3}$$

Thus, (2) and (3) form a couple of Lotka–Volterra-type equations.

Remark 4. Let us try to explain why it was not possible to obtain a similar result for the confluent vector ε-algorithm. In this algorithm, the inverse of a vector y was defined as $y^{-1} = \bar{y}/(y, y)$. But, if we consider y as a rectangular matrix, its Moore–Penrose inverse is defined as $y^\dagger = \bar{y}^T/(y, y)$. Thus, $\bar{y}^\dagger$ is a row vector which, when applied to any vector, appears as a linear form on $\mathbb{C}^p$, that is as belonging to the dual space (which, in this case is also $\mathbb{C}^p$), and not as an element of the space itself. This is another argument that the (discrete or confluent) topological ε-algorithm, instead of the (discrete or confluent) vector one, is the natural generalization of the (discrete or confluent) scalar ε-algorithm.

4.3. The confluent matrix ε-algorithm

Similarly, if $f(t)$ is a matrix function of t, the confluent matrix ε-algorithm is defined by the same rule as for the confluent vector one, and the same cross rule is still valid. Integrating the confluent matrix ε-algorithm by the Euler-type scheme of [13] leads to the usual discrete matrix ε-algorithm.

Differentiating the rule of this algorithm gives (1) again, and, setting again $M_k = \varepsilon'_k$, we now have

$$M_{k+1} - M_{k-1} = \left(M_k^{-1}\right)' = -M_k^{-1} M'_k M_k^{-1},$$

by using the usual formula $(A^{-1})' = -A^{-1}A'A^{-1}$ for the derivative of the inverse of matrix A. Thus, we obtain the following matrix discrete equation:

$$\begin{aligned} M'_k &= M_k[M_{k-1} - M_{k+1}]M_k \\ &= P_{k-1}M_k - M_k P_k \\ &= M_k N_{k-1} - N_k M_k, \end{aligned}$$

if we set $N_k = M_k M_{k+1}$ and $P_k = M_{k+1}M_k$.

Deriving N_k and P_k, we have

$$N'_k = M'_k M_{k+1} + M_k M'_{k+1}, \qquad P'_k = M'_{k+1}M_k + M_{k+1}M'_k.$$

Replacing M'_k by its expression above, we obtain, after an easy manipulation, the couple of equations

$$N'_k = P_{k-1}N_k - N_k P_{k+1} \tag{4}$$

$$P'_k = P_k N_{k-1} - N_{k+1}P_k. \tag{5}$$

Defining the shift operator E acting on a sequence (u_k) by $E^n u_k = u_{k+n}$, these equations write

$$\begin{aligned} N'_k &= (E^{-1}P_k)N_k - N_k(EP_k) = P_{k-1}(EN_{k-1}) - N_k(EP_k) \\ P'_k &= P_k(E^{-1}N_k) - (EN_k)P_k = (EP_{k-1})N_{k-1} - (EN_k)P_k. \end{aligned}$$

Let us mention that the shift operator E also plays a role in the discrete zero curvature equations considered in [11] to master symmetries of lattice equations.

Let $[L, B] = LB - BL$ be the matrix commutator of the matrices L and B. It is easy to see that relations (4) and (5) can be written as an extended coupled Lotka–Volterra system

$$\begin{aligned} N'_k &= [P_{k-1}, N_k] + N_k(P_{k-1} - P_{k+1}) \\ &= [P_{k+1}, N_k] + (P_{k-1} - P_{k+1})N_k \\ P'_k &= [P_k, N_{k+1}] + P_k(N_{k-1} - N_{k+1}) \\ &= [P_k, N_{k-1}] + (N_{k-1} - N_{k+1})P_k. \end{aligned}$$

If, for all k, the matrices N_k and P_{k-1} commute, then we obtain the coupled Lotka–Volterra equation

$$\begin{aligned} N'_k &= N_k[P_{k-1} - P_{k+1}] \\ P'_k &= P_k[N_{k-1} - N_{k+1}], \end{aligned}$$

while, if N_k commutes with P_{k+1} for all k, we get

$$\begin{aligned} N'_k &= [P_{k-1} - P_{k+1}]N_k \\ P'_k &= [N_{k-1} - N_{k+1}]P_k. \end{aligned}$$

Acknowledgments

Thanks are due to Basil Grammaticos, Yoshimasa Nakamura and Jonathan Nimmo for providing useful references. I am grateful to Xing-Biao Hu for comments on this paper. I also thank the reviewers for useful remarks which helped to clarify some points in the paper.

References

[1] Brezinski C 1975 Généralisations de la transformation de Shanks, de la table de Padé et de l'ε-algorithme *Calcolo* **12** 317–60
[2] Brezinski C 1975 Forme confluente de l'ε-algorithme topologique *Numer. Math.* **23** 363–70
[3] Brezinski C 2000 Convergence acceleration during the 20th century *J. Comput. Appl. Math.* **122** 1–21
[4] Brezinski C 2001 Dynamical systems and sequence transformations *J. Phys. A: Math. Gen.* **34** 10659–69
[5] Brezinski C 2002 *Computational Aspects of Linear Control* (Dordrecht: Kluwer)
[6] Brezinski C 2004 Extrapolation algorithms for filtering series of functions, and treating the Gibbs phenomenon *Numer. Algorithms* **36** 309–29
[7] Brezinski C 2007 A brief introduction to integrable systems *Int. J. Comput. Sci. Math.* **1** 98–106
[8] Brezinski C, He Y, Hu X-B and Sun J-Q Cross rules of some extrapolation algorithms unpublished
[9] Brezinski C and Redivo–Zaglia M 1991 *Extrapolation Methods. Theory and Practice* (Amsterdam: North–Holland)
[10] Chu M T 2008 Numerical linear algebra algorithms as dynamical systems *Acta Numerica* **17** 1–86
[11] Fuchssteiner B and Ma W-X 1999 An approach to master symmetries of lattice equations *Symmetries and Integrability of Differential Equations* ed P A Clarkson and F W Nijhoff (Cambridge: Cambridge University Press) pp 247–60
[12] Grammaticos B, Halburd R G, Ramani A and Viallet C-M 2009 How to detect the integrability of discrete systems *J. Phys. A: Math. Theor.* **42** 454002
[13] Hirota R, Tsujimoto S and Imai T 1993 Difference scheme of soliton equations *Future Directions of Nonlinear Dynamics in Physical and Biological Systems (Lyngby, 1992) (NATO Adv. Sci. Inst. Ser. B Phys.* vol 312*)* ed P L Christiansen, J C Eilbeck and R D Parmentier (New York: Plenum) pp 7–15
[14] Li C X and Nimmo J J C 2009 A non-commutative semi-discrete Toda equation and its quasi-determinant solutions *Glasgow Math. J.* **51A** 121–7
[15] Lou S Y, Tong B, Jia M and Li J-H 2007 A coupled Volterra system and its exact solutions arXiv:0711.0420v1
[16] Nagai A and Satsuma J 1995 Discrete soliton equations and convergence acceleration algorithms *Phys. Lett.* A **209** 305–12
[17] Nagai A, Tokihiro T and Satsuma J 1998 The Toda molecule equation and the ε–algorithm *Math. Comput.* **67** 1565–75
[18] Papageorgiou V, Grammaticos B and Ramani A 1996 Integrable difference equations and numerical analysis algorithms *Symmetries and Integrability of Difference Equations (CRM Proceedings and Lecture Notes* vol 9*)* ed D Levi *et al* (Providence, RI: AMS) pp 269–79
[19] Papageorgiou V, Grammaticos B and Ramani A 1993 Integrable lattices and convergence acceleration algorithms *Phys. Lett.* A **179** 111–5
[20] Shanks D 1955 Non linear transformations of divergent and slowly convergent sequences *J. Math. Phys.* **34** 1–42
[21] Wynn P 1956 On a device for computing the $e_m(S_n)$ transformation *Math. Tables Aids Comput.* **10** 91–6
[22] Wynn P 1960 Confluent forms of certain nonlinear algorithms *Arch. Math.* **11** 223–34
[23] Wynn P 1962 Acceleration techniques for iterated vector and matrix problems *Math. Comput.* **16** 301–22
[24] Zhao H-Q and Zhu Z-N 2009 Multi-soliton, multi-positon, multi-negaton, and multi-periodic solutions of the coupled Volterra lattice equation arXiv:0911.3458v1