Shanks sequence transformations and the ε–algorithms. Theory and applications

C. Brezinski [1]   **M. Redivo–Zaglia** [2]

[1]Université de Lille - Sciences et Technologies, France

[2]Università degli Studi di Padova, Italy.

Presentation

Introduction

Let (S_n) be a sequence converging to a limit S.

If it converges **slowly**, it is possible to transform it, by a **sequence transformation**, into a sequence (or a set of sequences) which converges, under some assumptions, **faster** to the same limit.

Introduction

Let $(\mathbf{S_n})$ be a sequence converging to a limit $\mathbf{S}$.

If it converges **slowly**, it is possible to transform it, by a **sequence transformation**, into a sequence (or a set of sequences) which converges, under some assumptions, **faster** to the same limit.

Any sequence transformation is built on the idea that the sequence to be accelerated behaves like a **model sequence** defined by some given properties.

The scalar Shanks transformation

Shanks transformation (1955) assumes that the **scalar sequence** (S_n) satisfies the **linear difference equation of order k**

$$a_0(S_n - S) + a_1(S_{n+1} - S) + \cdots + a_k(S_{n+k} - S) = 0, \quad n = 0, 1, \ldots$$

where the **unknown scalars** a_i are such that $a_0 a_k \neq 0$ and $a_0 + \cdots + a_k \neq 0$.

The scalar Shanks transformation

Shanks transformation (1955) assumes that the **scalar sequence** (S_n) satisfies the **linear difference equation of order k**

$$a_0(S_n - S) + a_1(S_{n+1} - S) + \cdots + a_k(S_{n+k} - S) = 0, \quad n = 0, 1, \ldots$$

where the **unknown scalars** a_i are such that $a_0 a_k \neq 0$ and $a_0 + \cdots + a_k \neq 0$.

Then, the idea is to compute S.

The scalar Shanks transformation

Shanks transformation (1955) assumes that the **scalar sequence** $(\mathbf{S_n})$ satisfies the **linear difference equation of order k**

$$\mathbf{a_0(S_n - S) + a_1(S_{n+1} - S) + \cdots + a_k(S_{n+k} - S) = 0, \quad n = 0, 1, \ldots}$$

where the **unknown scalars** $\mathbf{a_i}$ are such that $\mathbf{a_0 a_k \neq 0}$ and $\mathbf{a_0 + \cdots + a_k \neq 0}$.

Then, the idea is to compute $\mathbf{S}$.

Writing the preceding relation for the indexes

$$\mathbf{n, n+1, \ldots, n+k}$$

and solving the linear system for the unknown $\mathbf{S}$ gives the following formula

$$S = \frac{\begin{vmatrix} S_n & \cdots & S_{n+k} \\ \Delta S_n & \cdots & \Delta S_{n+k} \\ \vdots & & \vdots \\ \Delta S_{n+k-1} & \cdots & \Delta S_{n+2k-1} \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ \Delta S_n & \cdots & \Delta S_{n+k} \\ \vdots & & \vdots \\ \Delta S_{n+k-1} & \cdots & \Delta S_{n+2k-1} \end{vmatrix}}.$$

If the sequence (S_n) does not satisfy the **linear difference equation** given above, then the preceding ratio of determinant will give only an **approximation** of the limit S of the sequence (S_n).

If the sequence (S_n) does not satisfy the **linear difference equation** given above, then the preceding ratio of determinant will give only an **approximation** of the limit S of the sequence (S_n).

This approximation will depend on k and n, and will be denoted by

$$e_k(S_n)$$

If the sequence (S_n) does not satisfy the **linear difference equation** given above, then the preceding ratio of determinant will give only an **approximation** of the limit S of the sequence (S_n).

This approximation will depend on k and n, and will be denoted by

$$e_k(S_n)$$

Thus the sequence (S_n) has been transformed into the set of sequences $(e_k(S_n))$.

If the sequence (S_n) does not satisfy the **linear difference equation** given above, then the preceding ratio of determinant will give only an **approximation** of the limit S of the sequence (S_n).

This approximation will depend on k and n, and will be denoted by

$$e_k(S_n)$$

Thus the sequence (S_n) has been transformed into the set of sequences $(e_k(S_n))$.

This is **Shanks transformation**.

The scalar ε–algorithm

It is well–known that a numerical analyst is unable to compute determinants.

The scalar ε–algorithm

It is well–known that a numerical analyst is unable to compute determinants.

The ε-algorithm due to Wynn (1956) is a recursive algorithm for implementing Shanks transformation. Its rules are

$$\left.\begin{array}{rcl} \varepsilon_{-1}^{(n)} & = & 0, \qquad n = 0, 1, \ldots, \\ \varepsilon_{0}^{(n)} & = & S_n, \qquad n = 0, 1, \ldots, \\ \varepsilon_{k+1}^{(n)} & = & \varepsilon_{k-1}^{(n+1)} + (\varepsilon_{k}^{(n+1)} - \varepsilon_{k}^{(n)})^{-1}, \qquad k, n = 0, 1, \ldots, \end{array}\right\}$$

The scalar ε–algorithm

It is well–known that a numerical analyst is unable to compute determinants.

The ε-algorithm due to Wynn (1956) is a recursive algorithm for implementing Shanks transformation. Its rules are

$$\left.\begin{array}{lcll}
\varepsilon_{-1}^{(n)} & = & 0, & n = 0, 1, \ldots, \\
\varepsilon_{0}^{(n)} & = & S_n, & n = 0, 1, \ldots, \\
\varepsilon_{k+1}^{(n)} & = & \varepsilon_{k-1}^{(n+1)} + (\varepsilon_{k}^{(n+1)} - \varepsilon_{k}^{(n)})^{-1}, & k, n = 0, 1, \ldots,
\end{array}\right\}$$

and it holds

$$\varepsilon_{2k}^{(n)} = e_k(S_n), \qquad \varepsilon_{2k+1}^{(n)} = 1/e_k(\Delta S_n), \qquad k, n = 0, 1, \ldots$$

The scalar ε–algorithm

It is well–known that a numerical analyst is unable to compute determinants.

The ε-algorithm due to Wynn (1956) is a recursive algorithm for implementing Shanks transformation. Its rules are

$$\left.\begin{array}{lcl}
\varepsilon_{-1}^{(n)} & = & 0, \qquad n = 0, 1, \ldots, \\
\varepsilon_{0}^{(n)} & = & S_n, \qquad n = 0, 1, \ldots, \\
\varepsilon_{k+1}^{(n)} & = & \varepsilon_{k-1}^{(n+1)} + (\varepsilon_{k}^{(n+1)} - \varepsilon_{k}^{(n)})^{-1}, \qquad k, n = 0, 1, \ldots,
\end{array}\right\}$$

and it holds

$$\varepsilon_{2k}^{(n)} = e_k(S_n), \qquad \varepsilon_{2k+1}^{(n)} = 1/e_k(\Delta S_n), \qquad k, n = 0, 1, \ldots$$

Remark: The quantities with an odd lower index are intermediate computations.

The ε's are put in a table, called the **ε-array (red quantities are computed from blue ones)**

$$
\begin{array}{llllll}
\varepsilon_{-1}^{(0)} = 0 & & & & & \\
& \varepsilon_{0}^{(0)} = S_0 & & & & \\
\varepsilon_{-1}^{(1)} = 0 & & \varepsilon_{1}^{(0)} & & & \\
& \varepsilon_{0}^{(1)} = S_1 & & \varepsilon_{2}^{(0)} & & \\
\varepsilon_{-1}^{(2)} = 0 & & \varepsilon_{1}^{(1)} & & \varepsilon_{3}^{(0)} & \\
& \varepsilon_{0}^{(2)} = S_2 & & \boldsymbol{\varepsilon_{2}^{(1)}} & & \varepsilon_{4}^{(0)} \\
\varepsilon_{-1}^{(3)} = 0 & & \boldsymbol{\varepsilon_{1}^{(2)}} & & \boldsymbol{\varepsilon_{3}^{(1)}} & \\
& \varepsilon_{0}^{(3)} = S_3 & & \boldsymbol{\varepsilon_{2}^{(2)}} & & \\
\varepsilon_{-1}^{(3)} = 0 & & \varepsilon_{1}^{(3)} & & & \\
& \varepsilon_{0}^{(4)} = S_4 & & & & \\
\varepsilon_{-1}^{(4)} = 0 & & & & &
\end{array}
$$

It is the so called **rhombus scheme**, where the quantity in **red** is computed from the three ones in **blue**.

$$\begin{tikzcd}
 & \varepsilon_{\mathbf{k}}^{(\mathbf{n})} \arrow[dr] & \\
\varepsilon_{\mathbf{k-1}}^{(\mathbf{n+1})} \arrow[ur] \arrow[dr] & & \varepsilon_{\mathbf{k+1}}^{(\mathbf{n})} \\
 & \varepsilon_{\mathbf{k}}^{(\mathbf{n+1})} \arrow[ur] &
\end{tikzcd}$$

Shanks transformation and the ε–algorithm are related to **Padé approximants**.

Indeed, let

$$\mathbf{f(t) = c_0 + c_1 t + c_2 t^2} + \cdots$$

If Shanks transformation is applied to the sequence of the **partial sums of f**, then

$$\mathbf{e_k(S_n)} = \varepsilon_{\mathbf{2k}}^{\mathbf{(n)}} = \mathbf{[n + k/k]_f(t)}.$$

The vector ε–algorithm

If $(\mathbf{S_n})$ is a **sequence of vectors**, the ε–algorithm can still be used by defining the **inverse of a vector y** by

$$\mathbf{y}^{-1} = \mathbf{y}/(\mathbf{y}, \mathbf{y})$$

One obtains the **vector ε–algorithm** also due to Wynn (1962).

The vector ε–algorithm

If $(\mathbf{S_n})$ is a **sequence of vectors**, the ε–algorithm can still be used by defining the **inverse of a vector y** by

$$\mathbf{y}^{-1} = \mathbf{y}/(\mathbf{y}, \mathbf{y})$$

One obtains the **vector ε–algorithm** also due to Wynn (1962).

But, the vectors $\varepsilon_{\mathbf{2k}}^{(\mathbf{n})}$ it produces are no longer represented by a ratio of determinants (but by a ratio of much larger determinants) or by designants, which makes its algebraic theory much more arduous.

The topological Shanks transformation

For building a **transformation similar to Shanks'**, it was proposed (C.B. (1975)) to consider a sequence $(\mathbf{S_n})$ of **elements of a vector space E**, and to start again from the difference equation

$$\mathbf{a_0}(\mathbf{S_n} - \mathbf{S}) + \cdots + \mathbf{a_k}(\mathbf{S_{n+k}} - \mathbf{S}) = \mathbf{0}, \quad \mathbf{n} = \mathbf{0}, \mathbf{1}, \ldots,$$

with $a_0 a_k \neq 0$ and $\mathbf{a_0} + \cdots + \mathbf{a_k} = \mathbf{1}$ which does not restrict the generality.

The topological Shanks transformation

For building a **transformation similar to Shanks'**, it was proposed (C.B. (1975)) to consider a sequence $(\mathbf{S_n})$ of **elements of a vector space E**, and to start again from the difference equation

$$\mathbf{a_0(S_n - S) + \cdots + a_k(S_{n+k} - S) = 0}, \quad \mathbf{n = 0, 1, \ldots},$$

with $a_0 a_k \neq 0$ and $\mathbf{a_0 + \cdots + a_k = 1}$ which does not restrict the generality.

For computing the unknown coefficients $\mathbf{a_i}$, we have to obtain $\mathbf{k+1}$ **equations in** $\mathbb{R}$ from the preceding difference equation in **E**.

Let $\mathbf{y}$ be an arbitrary nonzero element of $\mathbf{E}^*$, the **dual space** of $\mathbf{E}$ (which means that it is a linear functional).

We denote by $\langle \mathbf{y}, \mathbf{v} \rangle \in \mathbb{R}$ the duality product between $\mathbf{y} \in \mathbf{E}^*$ and $\mathbf{v} \in \mathbf{E}$.

Let **y** be an arbitrary nonzero element of $\mathbf{E}^*$, the **dual space** of **E** (which means that it is a <u>linear functional</u>).

We denote by $\langle \mathbf{y}, \mathbf{v} \rangle \in \mathbb{R}$ the duality product between $\mathbf{y} \in \mathbf{E}^*$ and $\mathbf{v} \in \mathbf{E}$.

Writing the difference equation for two successive indexes, subtracting them, and using the duality product with **y**, the unknown coefficients $\mathbf{a_i}$ are computed as the solution of the system

$$\left.\begin{array}{ccccccc}
\mathbf{a_0} & + & \cdots & + & \mathbf{a_k} & = & 1 \\
\mathbf{a_0} < \mathbf{y}, \mathbf{\Delta S}_n > & + & \cdots & + & \mathbf{a_k} < \mathbf{y}, \mathbf{\Delta S}_{n+k} > & = & 0 \\
\vdots & & & & \vdots & & \\
\mathbf{a_0} < \mathbf{y}, \mathbf{\Delta S}_{n+k-1} > & + & \cdots & + & \mathbf{a_k} < \mathbf{y}, \mathbf{\Delta S}_{n+2k_1} > & = & 0.
\end{array}\right\}$$

Then, the transformation is defined by

$$e_k(S_n) = a_0 S_n + \cdots + a_k S_{n+k}.$$

It holds

$$e_k(S_n) = \frac{\begin{vmatrix} S_n & \cdots & S_{n+k} \\ < y, \Delta S_n > & \cdots & < y, \Delta S_{n+k} > \\ \vdots & & \vdots \\ < y, \Delta S_{n+k-1} > & \cdots & < y, \Delta S_{n+2k-1} > \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ < y, \Delta S_n > & \cdots & < y, \Delta S_{n+k} > \\ \vdots & & \vdots \\ < y, \Delta S_{n+k-1} > & \cdots & < y, \Delta S_{n+2k-1} > \end{vmatrix}}, \quad k, n = 0, 1, \ldots$$

The topological ε-algorithm

These elements of **E** can be recursively computed by the **first topological ε-algorithm (TEA1)**.
For $n = 0, 1, \ldots,$

$$
\begin{aligned}
\varepsilon_{-1}^{(n)} &= \mathbf{0} \in \mathbf{E}^*, \quad \varepsilon_{0}^{(n)} = \mathbf{S_n} \in \mathbf{E}, \\
\varepsilon_{2k+1}^{(n)} &= \varepsilon_{2k-1}^{(n+1)} + \frac{\mathbf{y}}{< \mathbf{y}, \varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)} >} \in \mathbf{E}^*, \\
\varepsilon_{2k+2}^{(n)} &= \varepsilon_{2k}^{(n+1)} + \frac{\varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)}}{< \varepsilon_{2k+1}^{(n+1)} - \varepsilon_{2k+1}^{(n)}, \varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)} >} \in \mathbf{E}.
\end{aligned}
$$

The topological ε-algorithm

These elements of **E** can be recursively computed by the **first topological ε-algorithm (TEA1)**.
For $n = 0, 1, \ldots,$

$$\begin{aligned}
\varepsilon_{-1}^{(n)} &= \mathbf{0} \in \mathbf{E}^*, \quad \varepsilon_0^{(n)} = \mathbf{S_n} \in \mathbf{E}, \\
\varepsilon_{2k+1}^{(n)} &= \varepsilon_{2k-1}^{(n+1)} + \frac{\mathbf{y}}{< \mathbf{y}, \varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)} >} \in \mathbf{E}^*, \\
\varepsilon_{2k+2}^{(n)} &= \varepsilon_{2k}^{(n+1)} + \frac{\varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)}}{< \varepsilon_{2k+1}^{(n+1)} - \varepsilon_{2k+1}^{(n)}, \varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)} >} \in \mathbf{E}.
\end{aligned}$$

and it holds

$$\varepsilon_{2k}^{(n)} = \mathbf{e_k}(\mathbf{S_n}), \qquad \varepsilon_{2k+1}^{(n)} = \mathbf{y} / < \mathbf{y}, \mathbf{e_k}(\Delta \mathbf{S_n}) >, \quad \mathbf{k}, \mathbf{n} = 0, 1, \ldots$$

The topological ε-algorithm

These elements of **E** can be recursively computed by the **first topological ε-algorithm (TEA1)**.

For $n = 0, 1, \ldots,$

$$\begin{aligned}
\varepsilon_{-1}^{(n)} &= 0 \in E^*, \quad \varepsilon_0^{(n)} = S_n \in E, \\
\varepsilon_{2k+1}^{(n)} &= \varepsilon_{2k-1}^{(n+1)} + \frac{y}{< y, \varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)} >} \in E^*, \\
\varepsilon_{2k+2}^{(n)} &= \varepsilon_{2k}^{(n+1)} + \frac{\varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)}}{< \varepsilon_{2k+1}^{(n+1)} - \varepsilon_{2k+1}^{(n)}, \varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)} >} \in E.
\end{aligned}$$

and it holds

$$\varepsilon_{2k}^{(n)} = e_k(S_n), \qquad \varepsilon_{2k+1}^{(n)} = y / < y, e_k(\Delta S_n) >, \quad k, n = 0, 1, \ldots$$

Remark: Again, the elements with an odd lower index are intermediate computations (they are elements of E^*).

The ε's are put in a table as above.

The ε's are put in a table as above.
The quantities in **red** are computed from the quantities in **blue** but **the rule is different** for those with an **odd or an even lower index**.

The ε's are put in a table as above.
The quantities in **red** are computed from the quantities in **blue** but **the rule is different** for those with an **odd or an even lower index**.

odd columns

$$\begin{tikzcd}
 & \varepsilon_{2k}^{(n)} \arrow[dr] & \\
\varepsilon_{2k-1}^{(n+1)} \arrow[ur] \arrow[dr] & & \varepsilon_{2k+1}^{(n)} \\
 & \varepsilon_{2k}^{(n+1)} \arrow[ur] &
\end{tikzcd}$$

even columns

$$\begin{tikzcd}
\varepsilon_{2k}^{(n)} \arrow[dr] & & \\
 & \varepsilon_{2k+1}^{(n)} \arrow[dr] & \\
\varepsilon_{2k}^{(n+1)} \arrow[ur] \arrow[dr] & & \varepsilon_{2k+2}^{(n)} \\
 & \varepsilon_{2k+1}^{(n+1)} \arrow[ur] &
\end{tikzcd}$$

The $\varepsilon_{\mathbf{2k}}^{(\mathbf{n})}$ are elements of **E**, while the $\varepsilon_{\mathbf{2k+1}}^{(\mathbf{n})}$ are elements of $\mathbf{E}^*$, that is **linear functionals** on E.

The $\varepsilon_{2k}^{(n)}$ are elements of **E**, while the $\varepsilon_{2k+1}^{(n)}$ are elements of $\mathbf{E}^*$, that is **linear functionals** on E.

For the **implementation**, each time a new element of the sequence $(\mathbf{S_n})$ is added in the first column, a new ascending diagonal in the table is computed.

The $\varepsilon_{2k}^{(n)}$ are elements of **E**, while the $\varepsilon_{2k+1}^{(n)}$ are elements of $\mathbf{E}^*$, that is **linear functionals** on E.

For the **implementation**, each time a new element of the sequence $(\mathbf{S_n})$ is added in the first column, a new ascending diagonal in the table is computed.

For computing the elements in **red**, it is necessary to store the elements in **blue** and those in **green**.

The $\varepsilon_{2k}^{(n)}$ are elements of **E**, while the $\varepsilon_{2k+1}^{(n)}$ are elements of $\mathbf{E}^*$, that is **linear functionals** on E.

For the **implementation**, each time a new element of the sequence $(\mathbf{S_n})$ is added in the first column, a new ascending diagonal in the table is computed.

For computing the elements in **red**, it is necessary to store the elements in **blue** and those in **green**.

Thus, in total,

one and a half diagonal

of elements (of E and E^*) has to be stored.

$$\begin{array}{llllll}
\varepsilon_{-1}^{(0)} = \mathbf{0} & & & & & \\
& \varepsilon_{0}^{(0)} = \mathbf{S}_0 & & & & \\
\varepsilon_{-1}^{(1)} = \mathbf{0} & & \varepsilon_{1}^{(0)} & & & \\
& \varepsilon_{0}^{(1)} = \mathbf{S}_1 & & \varepsilon_{2}^{(0)} & & \\
\varepsilon_{-1}^{(2)} = \mathbf{0} & & \varepsilon_{1}^{(1)} & & \varepsilon_{3}^{(0)} & \\
& \varepsilon_{0}^{(2)} = \mathbf{S}_2 & & \varepsilon_{2}^{(1)} & & \varepsilon_{4}^{(0)} \\
\varepsilon_{-1}^{(3)} = \mathbf{0} & & \varepsilon_{1}^{(2)} & & \varepsilon_{3}^{(1)} & \\
& \varepsilon_{0}^{(3)} = \mathbf{S}_3 & & \varepsilon_{2}^{(2)} & & \\
\varepsilon_{-1}^{(3)} = \mathbf{0} & & \varepsilon_{1}^{(3)} & & & \\
& \varepsilon_{0}^{(4)} = \mathbf{S}_4 & & & & \\
\varepsilon_{-1}^{(4)} = \mathbf{0} & & & & &
\end{array}$$

The simplified topological ε-algorithm

We will now show how to **simplify the rules** of this algorithm in order to have **to store less elements**.

The simplified topological ε-algorithm

We will now show how to **simplify the rules** of this algorithm in order to have **to store less elements**.

Moreover, we will be able to **store and compute** only those with an **even lower index**, which are elements of **E** (the interesting ones).

The simplified topological ε-algorithm

We will now show how to **simplify the rules** of this algorithm in order to have **to store less elements**.

Moreover, we will be able to **store and compute** only those with an **even lower index**, which are elements of **E** (the interesting ones).

It holds

$$< \mathbf{y}, \varepsilon_{2k}^{(n)} > = e_k(< \mathbf{y}, \mathbf{S_n} >) \quad \text{and} \quad \varepsilon_{2k+1}^{(n)} = \mathbf{y}/e_k(< \mathbf{y}, \Delta\mathbf{S_n} >).$$

Then, applying the **scalar ε-algorithm** to the sequence $(S_n = < \mathbf{y}, \mathbf{S_n} >)$, we have $\varepsilon_{2k}^{(n)} = e_k(< \mathbf{y}, \mathbf{S_n} >)$.

Remark: Letters in **blue** are **scalars**, while those in **red** denote elements of **E** and **E***.

We finally obtain the **first simplified topological ε-algorithm (STEA1)**

$$\boldsymbol{\varepsilon}_{2k+2}^{(n)} = \boldsymbol{\varepsilon}_{2k}^{(n+1)} + \frac{\varepsilon_{2k+2}^{(n)} - \varepsilon_{2k}^{(n+1)}}{\varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)}} (\boldsymbol{\varepsilon}_{2k}^{(n+1)} - \boldsymbol{\varepsilon}_{2k}^{(n)}),$$

with $\boldsymbol{\varepsilon}_0^{(n)} = \mathbf{S_n}$ and $\varepsilon_0^{(n)} =< \mathbf{y}, \mathbf{S_n} >$.

The ε (in **red** and **bold**) are **elements of** E.

The ε's **in blue** are **scalars** computed by the **scalar ε-algorithm** applied to the sequence $(< \mathbf{y}, \mathbf{S_n} >)$.

We finally obtain the **first simplified topological ε-algorithm (STEA1)**

$$\boldsymbol{\varepsilon}_{2k+2}^{(n)} = \boldsymbol{\varepsilon}_{2k}^{(n+1)} + \frac{\varepsilon_{2k+2}^{(n)} - \varepsilon_{2k}^{(n+1)}}{\varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)}} (\boldsymbol{\varepsilon}_{2k}^{(n+1)} - \boldsymbol{\varepsilon}_{2k}^{(n)}),$$

with $\boldsymbol{\varepsilon}_0^{(n)} = \mathbf{S}_n$ and $\varepsilon_0^{(n)} = < \mathbf{y}, \mathbf{S}_n >$.

The ε (in **red** and **bold**) are **elements of** E.

The ε's **in blue** are **scalars** computed by the **scalar ε-algorithm** applied to the sequence $(< \mathbf{y}, \mathbf{S}_n >)$.

Remark: In all these formulas, only elements of E (even lower indexes) now appear in the rule of the algorithm.
Elements of $\mathbf{E}^*$ (odd lower indexes) are **no longer required**. They have been replaced by the quantities obtained by the **scalar** ε–algorithm.

In the **ε-array**, one element in **red** is computed from those in **blue** (now a simple **triangular scheme** involving only even lower terms).

$$\begin{tikzcd}
\varepsilon_{2k}^{(n)} \arrow[dr] & \\
\varepsilon_{2k}^{(n+1)} \arrow[r] & \varepsilon_{2k+2}^{(n)}
\end{tikzcd}$$

In the **ε-array**, one element in **red** is computed from those in **blue** (now a simple **triangular scheme** involving only even lower terms).

$$\begin{tikzcd}
\varepsilon_{2k}^{(n)} \arrow[dr] & \\
\varepsilon_{2k}^{(n+1)} \arrow[r] & \varepsilon_{2k+2}^{(n)}
\end{tikzcd}$$

Now, in the implementation, only

two half diagonals = one diagonal

are needed.

In the ε-**array**, one element in **red** is computed from those in **blue** (now a simple **triangular scheme** involving only even lower terms).

$$\begin{tikzcd}
\varepsilon_{2k}^{(n)} \arrow[dr] & \\
\varepsilon_{2k}^{(n+1)} \arrow[r] & \varepsilon_{2k+2}^{(n)}
\end{tikzcd}$$

Now, in the implementation, only

two half diagonals = one diagonal

are needed.

Obviously also the **scalars** $\varepsilon_{2k}^{(n)}$ have to be stored, but it is not costly.

$$
\begin{array}{llllll}
\varepsilon_{-1}^{(0)} = 0 & & & & & \\
 & \boldsymbol{\varepsilon_0^{(0)}} = \mathbf{S_0} & & & & \\
\varepsilon_{-1}^{(1)} = 0 & & \varepsilon_1^{(0)} & & & \\
 & \boldsymbol{\varepsilon_0^{(1)}} = \mathbf{S_1} & & \boldsymbol{\varepsilon_2^{(0)}} & & \\
\varepsilon_{-1}^{(2)} = 0 & & \varepsilon_1^{(1)} & & \varepsilon_3^{(0)} & \\
 & \boldsymbol{\varepsilon_0^{(2)}} = \mathbf{S_2} & & \boldsymbol{\varepsilon_2^{(1)}} & & \boldsymbol{\varepsilon_4^{(0)}} \\
\varepsilon_{-1}^{(3)} = 0 & & \varepsilon_1^{(2)} & & \varepsilon_3^{(1)} & \\
 & \boldsymbol{\varepsilon_0^{(3)}} = \mathbf{S_3} & & \boldsymbol{\varepsilon_2^{(2)}} & & \\
\varepsilon_{-1}^{(3)} = 0 & & \varepsilon_1^{(3)} & & & \\
 & \boldsymbol{\varepsilon_0^{(4)}} = \mathbf{S_4} & & & & \\
\varepsilon_{-1}^{(4)} = 0 & & & & &
\end{array}
$$

There is **second topological Shanks transformation** which is defined by

$$\mathbf{e_k(S_n) = a_0 S_{n+k} + \cdots + a_k S_{n+2k}},$$

where the a_i's are the same as for the first one.

There is **second topological Shanks transformation** which is defined by

$$\mathbf{e_k(S_n) = a_0 S_{n+k} + \cdots + a_k S_{n+2k}},$$

where the a_i's are the same as for the first one.

It can be implemented by a **second topological ε-algorithm** which needs **one ascending diagonal** for its implementation.

There is **second topological Shanks transformation** which is defined by

$$e_k(S_n) = a_0 S_{n+k} + \cdots + a_k S_{n+2k},$$

where the a_i's are the same as for the first one.

It can be implemented by a **second topological ε-algorithm** which needs **one ascending diagonal** for its implementation.

There is a **second simplified topological ε-algorithm (STEA2)** which only requires the storage of

half of an ascending diagonal

instead of one.

$$\begin{tikzcd}
\varepsilon_{2k}^{(n+1)} \arrow[r] & \varepsilon_{2k+2}^{(n)} \\
{\varepsilon_{2k}}^{(n+2)} \arrow[ur] &
\end{tikzcd}$$

Convergence and acceleration

Thank to the form of the **simplified topological ε-algorithms**, and the fact that the vector **y** only intervenes in the initializations of the **scalar ε -algorithm**, and that the elements belonging to $\mathbf{E}^*$ are no longer needed, we are able **to prove some new convergence and acceleration results**.

We will not discuss them here.

Summary

- **The topological ε-algorithms: TEA1, TEA2**
  - **2 rules**
  - storage of the intermediate linear functionals $\varepsilon_{2k+1}^{(n)} \in \mathbf{E}^*$
  - computations involving the duality product with the $\varepsilon_{2k+1}^{(n)}$'s and with $\mathbf{y}$ inside the algorithm
  - convergence and acceleration **results difficult to obtain** because the rules of the algorithm are complicated
  - **numerical instability** can be present
- **The simplified topological ε -algorithms: STEA1, STEA2**
  - **only 1 rule**
  - **much less storage:** only the elements $\varepsilon_{2k}^{(n)} \in \mathbf{E}$
  - application of the linear functional $\mathbf{y}$ only to $\mathbf{S_n} \in \mathbf{E}$
  - no use of the duality product inside the algorithms
  - clearer role played by $\mathbf{y}$, but difficult to choose
  - **possibility to prove convergence and acceleration results**
  - **possibility to avoid some numerical instability**

Implementation and numerical examples

For computing **an element** $\varepsilon_{2k}^{(n)}$ **an element of the column 2k**, one needs the following **storage**

	# elements ε-array	in spaces	# working elements
TEA1	3k	E, E^*	4
STEA1	2k	E	2
TEA2	2k	E, E^*	4
STEA2	k	E	2

Implementation and numerical examples

For computing **an element** $\varepsilon_{2k}^{(n)}$ **an element of the column 2k**, one needs the following **storage**

	# elements ε-array	in spaces	# working elements
TEA1	3k	E, E^*	4
STEA1	2k	E	2
TEA2	2k	E, E^*	4
STEA2	k	E	2

In our numerical examples, we took for vectors $\mathbf{y} = (\mathbf{1}, \ldots, \mathbf{1})^T$ and, in the matrix case $< \mathbf{y}, \cdot >= \mathbf{tr}(\cdot)$.

Solution of equations

Let $\mathbf{F} : \mathbb{R}^{\mathbf{m}\times\mathbf{s}} \longmapsto \mathbb{R}^{\mathbf{m}\times\mathbf{s}}$. We consider the system of nonlinear equations

$$\mathbf{x} = \mathbf{F}(\mathbf{x}) \quad \text{or} \quad \mathbf{f}(\mathbf{x}) = \mathbf{F}(\mathbf{x}) - \mathbf{x} = \mathbf{0} \in \mathbb{R}^{\mathbf{m}\times\mathbf{s}}.$$

Solution of equations

Let $\mathbf{F} : \mathbb{R}^{\mathbf{m \times s}} \longmapsto \mathbb{R}^{\mathbf{m \times s}}$. We consider the system of nonlinear equations

$$\mathbf{x} = \mathbf{F}(\mathbf{x}) \quad \text{or} \quad \mathbf{f}(\mathbf{x}) = \mathbf{F}(\mathbf{x}) - \mathbf{x} = \mathbf{0} \in \mathbb{R}^{\mathbf{m \times s}}.$$

The Acceleration Method (AM)

For computing x, we can generate the **fixed point iterates** $\mathbf{x_{n+1}} = \mathbf{F}(\mathbf{x_n})$ starting for $\mathbf{x_0}$, and **accelerate** this sequence by applying the simplified topological ε-algorithm by considering the sequence $(\varepsilon_{\mathbf{2k}}^{(\mathbf{n})})_{\mathbf{n}}$ for a fixed value of k, or the sequence $(\varepsilon_{\mathbf{2k}}^{(\mathbf{0})})_{\mathbf{k}}$.

The Restarted and the Generalized Steffensen Methods (RM and GSM)

Another possibility is to use a generalization of the well-known Steffensen method

Restarted method (RM)

$$\begin{cases} \mathbf{u_0} = \mathbf{x_n} \\ \mathbf{u_i} = \mathbf{F}(\mathbf{u_{i-1}}), \quad \mathbf{i} = \mathbf{1}, \ldots, \mathbf{2k} \ \ (\textit{basic iterations}) \\ \text{Apply the simplified topological } \varepsilon\text{-algorithm to } \mathbf{u_0}, \ldots, \mathbf{u_{2k}} \\ \text{Set } \mathbf{x_{n+1}} = \varepsilon_{\mathbf{2k}}^{(\mathbf{0})} \end{cases}$$

The Restarted and the Generalized Steffensen Methods (RM and GSM)

Another possibility is to use a generalization of the well-known Steffensen method

Restarted method (RM)

$$\begin{cases} \mathbf{u_0} = \mathbf{x_n} \\ \mathbf{u_i} = \mathbf{F}(\mathbf{u_{i-1}}), \quad \mathbf{i} = \mathbf{1}, \ldots, \mathbf{2k} \ \ (\textit{basic iterations}) \\ \text{Apply the simplified topological } \varepsilon\text{-algorithm to } \mathbf{u_0}, \ldots, \mathbf{u_{2k}} \\ \text{Set } \mathbf{x_{n+1}} = \varepsilon_{\mathbf{2k}}^{(\mathbf{0})} \end{cases}$$

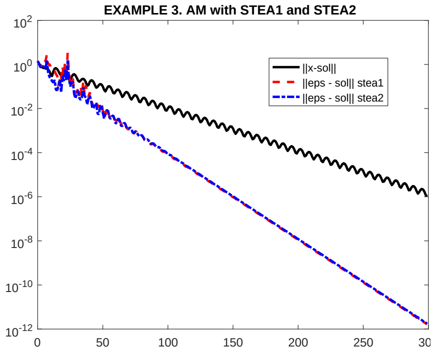
When $\mathbf{k} = \mathbf{m}$, the **dimension** of the system, the method is called the **Generalized Steffensen Method (GSM)**.
Under some assumptions, **this method converges quadratically** when $\mathbf{F} : \mathbb{R}^{\mathbf{m}} \longmapsto \mathbb{R}^{\mathbf{m}}$ (no proof yet when $F : \mathbb{R}^{m \times s} \longmapsto \mathbb{R}^{m \times s}$) .

Nonlinear algebraic system

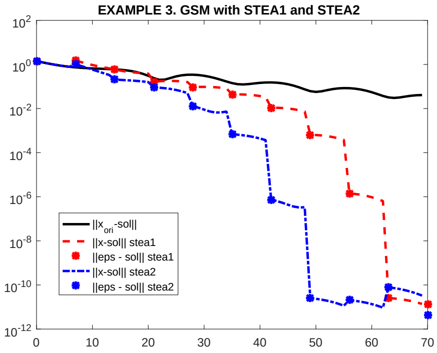
We consider the nonlinear system

$$\left\{\begin{array}{lcl} \mathbf{x_1} & = & \mathbf{x_1 x_2^3/2 - 1/2} + \sin \mathbf{x_3} \\ \mathbf{x_2} & = & (\exp(\mathbf{1 + x_1 x_2}) + \mathbf{1})/\mathbf{2} \\ \mathbf{x_3} & = & \mathbf{1} - \cos \mathbf{x_3} + \mathbf{x_1^4 - x_2}, \end{array}\right.$$

whose solution is $(-1, 1, 0)^T$.
Starting from $\mathbf{x_0} = \mathbf{0}$, we obtain the results of the left Figure for the AM. For the GSM, the STEA2 gives better results than the STEA1 as shown on the right Figure.



Acceleration Method



Generalized Steffensen Method

Maximal solution of matrix equation

We are looking for the **maximal Hermitian positive definite solution $\mathbf{X}_+$** of the matrix equation

$$\mathbf{f(X) = X + A^*X^{-1}A - Q = 0},$$

where $A, Q \in \mathbb{C}^{m\times m}$ with Q Hermitian positive definite.

Maximal solution of matrix equation

We are looking for the **maximal Hermitian positive definite solution $\mathbf{X}_+$** of the matrix equation

$$\mathbf{f(X) = X + A^*X^{-1}A - Q = 0},$$

where $A, Q \in \mathbb{C}^{m\times m}$ with Q Hermitian positive definite.

The matrix $\mathbf{X}_+$ is called a **maximal solution** if $X_+ - X$ is positive semidefinite for any Hermitian solution X of f.

Maximal solution of matrix equation

We are looking for the **maximal Hermitian positive definite solution $\mathbf{X}_+$** of the matrix equation

$$\mathbf{f}(\mathbf{X}) = \mathbf{X} + \mathbf{A}^*\mathbf{X}^{-1}\mathbf{A} - \mathbf{Q} = \mathbf{0},$$

where $A, Q \in \mathbb{C}^{m\times m}$ with Q Hermitian positive definite.

The matrix $\mathbf{X}_+$ is called a **maximal solution** if $X_+ - X$ is positive semidefinite for any Hermitian solution X of f.

For $Q = I + A^*A$, $X_+ = I$ if and only if $\rho(A) < 1$, a result which provides an easy way of constructing numerical examples by taking $A = S/r$ with $r > \rho(S)$ and S any matrix.

We use the following iterative method proposed by C.-H. Guo (1999)

$$\begin{aligned} \mathbf{X_0} &= \mathbf{Q}, \\ \mathbf{X_{n+1}} &= \mathbf{Q} - \mathbf{A}^*\mathbf{X_n}^{-1}\mathbf{A}, \quad \mathbf{n} = \mathbf{0}, \mathbf{1}, \ldots \end{aligned}$$

We use the following iterative method proposed by C.-H. Guo (1999)

$$\begin{aligned} \mathbf{X_0} &= \mathbf{Q}, \\ \mathbf{X_{n+1}} &= \mathbf{Q} - \mathbf{A}^*\mathbf{X_n^{-1}}\mathbf{A}, \quad \mathbf{n} = \mathbf{0}, \mathbf{1}, \ldots \end{aligned}$$

Converges slowly if the spectral radius of A is close to 1.

We use the following iterative method proposed by C.-H. Guo (1999)

$$\begin{aligned} \mathbf{X_0} &= \mathbf{Q}, \\ \mathbf{X_{n+1}} &= \mathbf{Q} - \mathbf{A}^*\mathbf{X_n}^{-1}\mathbf{A}, \quad \mathbf{n} = \mathbf{0}, \mathbf{1}, \ldots \end{aligned}$$

Converges slowly if the spectral radius of A is close to 1.

If we accelerate the convergence of the sequence (X_n) by the simplified topological ε–algorithm, we obtain the results of next Figure for the `prolate` matrix of the matrix toolbox.

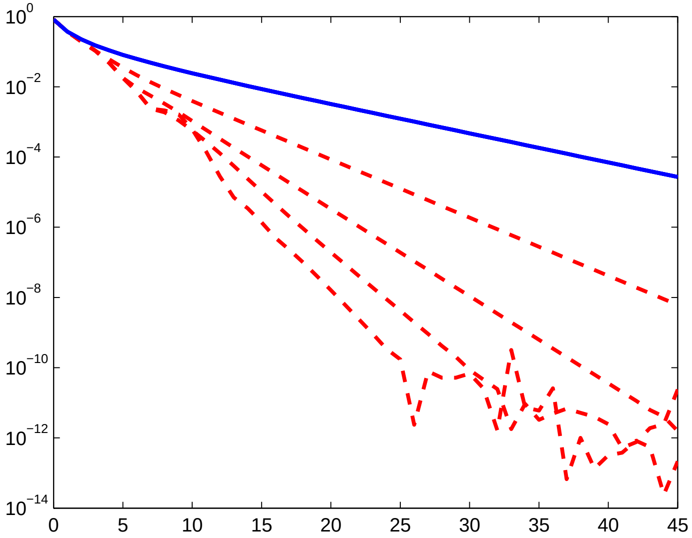
We use the following iterative method proposed by C.-H. Guo (1999)

$$\begin{aligned} \mathbf{X_0} &= \mathbf{Q}, \\ \mathbf{X_{n+1}} &= \mathbf{Q} - \mathbf{A}^*\mathbf{X_n^{-1}}\mathbf{A}, \quad \mathbf{n} = \mathbf{0}, \mathbf{1}, \ldots \end{aligned}$$

Converges slowly if the spectral radius of A is close to 1.

If we accelerate the convergence of the sequence (X_n) by the simplified topological ε–algorithm, we obtain the results of next Figure for the `prolate` matrix of the matrix toolbox.

The solid line corresponds to the error of the iterates $\mathbf{X_n}$, and the dashed ones to the sequences $(\varepsilon_{\mathbf{2}}^{(\mathbf{n})}), (\varepsilon_{\mathbf{4}}^{(\mathbf{n})}), (\varepsilon_{\mathbf{6}}^{(\mathbf{n})}), (\varepsilon_{\mathbf{8}}^{(\mathbf{n})})$.



10^0
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
0
5
10
15
20
25
30
35
40
45

We can also use this method as the basic iterations for the **Generalized Steffensen Method** described above. With the `clement` matrix of dimension 10 of the matrix toolbox (divided by 10 in order to have $\rho(A) = 0.9$), this method produces the following results for the norms.

We can also use this method as the basic iterations for the **Generalized Steffensen Method** described above. With the `clement` matrix of dimension 10 of the matrix toolbox (divided by 10 in order to have $\rho(A) = 0.9$), this method produces the following results for the norms.

	iter. 0	iter. 1	iter. 2	iter. 3
error	$9.77e-001$	$4.24e-004$	$2.72e-008$	$3.58e-011$
$\|f\|$	$5.17e-001$	$2.93e-004$	$2.88e-008$	$3.68e-011$

We can also use this method as the basic iterations for the **Generalized Steffensen Method** described above. With the `clement` matrix of dimension 10 of the matrix toolbox (divided by 10 in order to have $\rho(A) = 0.9$), this method produces the following results for the norms.

	iter. 0	iter. 1	iter. 2	iter. 3
error	$9.77e-001$	$4.24e-004$	$2.72e-008$	$3.58e-011$
$\|f\|$	$5.17e-001$	$2.93e-004$	$2.88e-008$	$3.68e-011$

The quadratic convergence is clearly seen.

Discrete-time Lyapunov equation

We consider the symmetric Stein matrix equation, also called the **discrete-time Lyapunov equation**,

$$\mathbf{S} - \mathbf{A}\mathbf{S}\mathbf{A}^{\mathbf{T}} = \mathbf{F}\mathbf{F}^{\mathbf{T}},$$

$\mathbf{F} \in \mathbb{R}^{\mathbf{m}\times\mathbf{s}}$, $s \ll m$, with the eigenvalues of $\mathbf{A}$ inside the unit disk.

Discrete-time Lyapunov equation

We consider the symmetric Stein matrix equation, also called the **discrete-time Lyapunov equation**,

$$\mathbf{S} - \mathbf{ASA}^{\mathbf{T}} = \mathbf{FF}^{\mathbf{T}},$$

$\mathbf{F} \in \mathbb{R}^{\mathbf{m}\times\mathbf{s}}$, $s \ll m$, with the eigenvalues of $\mathbf{A}$ inside the unit disk.

This equation can be solved by the iterative method

$$\mathbf{S}_{\mathbf{n+1}} = \mathbf{FF}^{\mathbf{T}} + \mathbf{AS}_{\mathbf{n}}\mathbf{A}^{\mathbf{T}}, \mathbf{n} = \mathbf{0}, \mathbf{1}, \ldots, \quad \mathbf{S}_{\mathbf{0}} = \mathbf{0}.$$

The results of the next Figure correspond to the `moler` matrix of dimension 500 for $\mathbf{A}$.

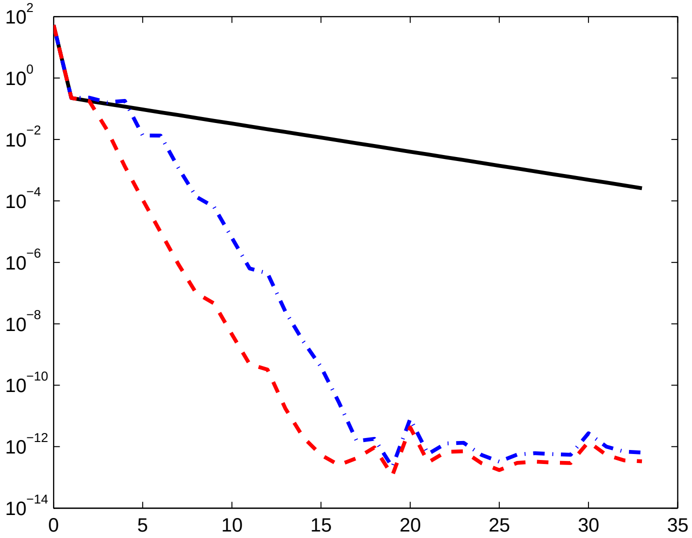
It is a symmetric positive definite matrix with one small eigenvalue. Its elements are $a_{ij} = \min(i, j) - 2$ and $a_{ii} = i$.

The results of the next Figure correspond to the `moler` matrix of dimension 500 for $\mathbf{A}$.

It is a symmetric positive definite matrix with one small eigenvalue. Its elements are $a_{ij} = \min(i,j) - 2$ and $a_{ii} = i$.

The `moler` matrix $\mathbf{A}$ is then divided by an adequate factor so that its spectral radius be equal to 0.9.

The matrix $\mathbf{F}$ is the `parter` matrix of dimension 500×30, and we took $k = 3$.


10^2
10^0
10^-2
10^-4
10^-6
10^-8
10^-10
10^-12
10^-14
0
5
10
15
20
25
30
35

Matrix function: square root

The binomial iteration for computing the **square root of $\mathbf{I}-\mathbf{C}$**, where $\rho(\mathbf{C})<\mathbf{1}$, consists in the iterations

$$\mathbf{X}_{n+1}=\frac{1}{2}(\mathbf{C}+\mathbf{X}_n^2),\quad \mathbf{k}=\mathbf{0},\mathbf{1},\ldots,\qquad \mathbf{X}_0=\mathbf{0}.$$

The sequence (X_n) converges linearly to $X=I-(I-C)^{1/2}$ and X_n reproduces the series

$$(I-C)^{1/2}=\sum_{i=0}^{\infty}\binom{1/2}{i}(-C)^i=I-\sum_{i=1}^{\infty}\alpha_i C_i,\quad \alpha_i>0,$$

up to and including the term C^n.

Matrix function: square root

The binomial iteration for computing the **square root of $\mathbf{I}-\mathbf{C}$**, where $\rho(\mathbf{C}) < \mathbf{1}$, consists in the iterations

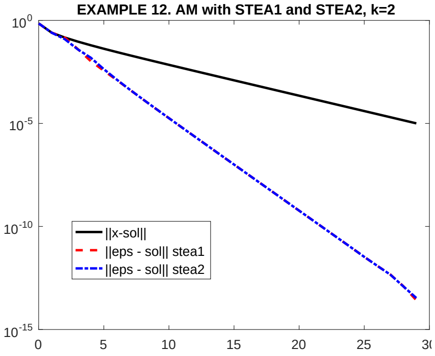
$$\mathbf{X_{n+1}} = \frac{\mathbf{1}}{\mathbf{2}}(\mathbf{C} + \mathbf{X_n^2}), \quad \mathbf{k} = \mathbf{0}, \mathbf{1}, \ldots, \qquad \mathbf{X_0} = \mathbf{0}.$$

The sequence (X_n) converges linearly to $X = I - (I - C)^{1/2}$ and X_n reproduces the series

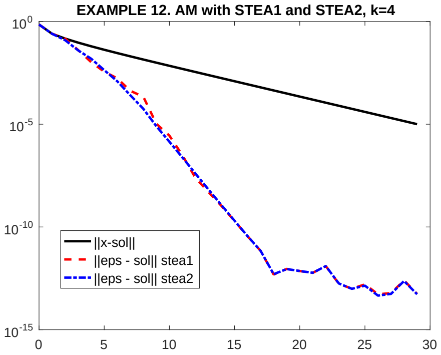
$$(I - C)^{1/2} = \sum_{i=0}^{\infty} \binom{1/2}{i} (-C)^i = I - \sum_{i=1}^{\infty} \alpha_i C_i, \quad \alpha_i > 0,$$

up to and including the term C^n.

For C, we took the matrix `moler` of dimension 500 divided by 1.1×10^5 so that $\rho(C) = 0.9855$. The **AM** gives the results of the following Figure with $k = 2$ on the left and $k = 4$ on the right, for the acceleration of the sequence (X_n).


EXAMPLE 12. AM with STEA1 and STEA2, k=2
10^{0}
10^{-5}
10^{-10}
10^{-15}
0
5
10
15
20
25
30
||x-sol||
||eps - soll|| stea1
||eps - soll|| stea2

AM, $k = 2$


EXAMPLE 12. AM with STEA1 and STEA2, k=4
10^{0}
10^{-5}
10^{-10}
10^{-15}
0
5
10
15
20
25
30
||x-sol||
||eps - soll|| stea1
||eps - soll|| stea2

AM, $k = 4$

Matrix function: logarithm

We now want to compute an approximation of

$$\log(\mathbf{I}+\mathbf{A}) = \mathbf{A} - \frac{\mathbf{A}^2}{2} + \frac{\mathbf{A}^3}{3} - \frac{\mathbf{A}^4}{4} + \cdots + (-1)^{n+1}\frac{\mathbf{A}^n}{n} + \cdots$$

where $\mathbf{A}$ is a real matrix.

Matrix function: logarithm

We now want to compute an approximation of

$$\log(\mathbf{I} + \mathbf{A}) = \mathbf{A} - \frac{\mathbf{A}^2}{2} + \frac{\mathbf{A}^3}{3} - \frac{\mathbf{A}^4}{4} + \cdots + (-1)^{\mathbf{n}+1}\frac{\mathbf{A}^{\mathbf{n}}}{\mathbf{n}} + \cdots$$

where $\mathbf{A}$ is a real matrix.
We know that the series is convergent if $\rho(\mathbf{A}) < 1$.

Matrix function: logarithm

We now want to compute an approximation of

$$\log(\mathbf{I}+\mathbf{A}) = \mathbf{A} - \frac{\mathbf{A}^2}{2} + \frac{\mathbf{A}^3}{3} - \frac{\mathbf{A}^4}{4} + \cdots + (-1)^{n+1}\frac{\mathbf{A}^n}{n} + \cdots$$

where $\mathbf{A}$ is a real matrix.
We know that the series is convergent if $\rho(\mathbf{A}) < 1$.
We will show that the topological ε-algorithm is able to accelerate the convergence of the **partial sums of this series** and also ... to **converge** to an approximation of the value, when the **series is divergent!**

Matrix function: logarithm

We now want to compute an approximation of

$$\log(\mathbf{I} + \mathbf{A}) = \mathbf{A} - \frac{\mathbf{A}^2}{2} + \frac{\mathbf{A}^3}{3} - \frac{\mathbf{A}^4}{4} + \cdots + (-1)^{\mathbf{n}+1}\frac{\mathbf{A}^{\mathbf{n}}}{\mathbf{n}} + \cdots$$

where $\mathbf{A}$ is a real matrix.
We know that the series is convergent if $\rho(\mathbf{A}) < 1$.
We will show that the topological ε-algorithm is able to accelerate the convergence of the **partial sums of this series** and also ... to **converge** to an approximation of the value, when the **series is divergent!**
We consider a random matrix B of dimension $N = 50$, we choose a value r, and we define $A = r \times B/\rho(B)$.

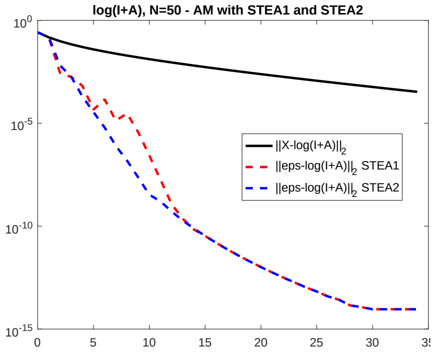
Matrix function: logarithm

We now want to compute an approximation of

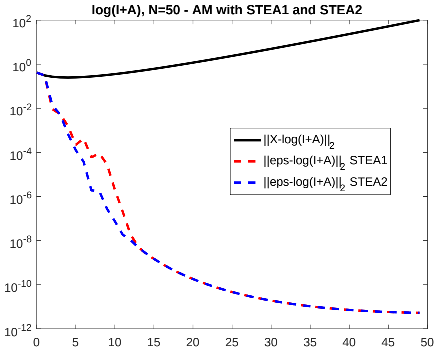
$$\log(\mathbf{I}+\mathbf{A}) = \mathbf{A} - \frac{\mathbf{A}^2}{\mathbf{2}} + \frac{\mathbf{A}^3}{\mathbf{3}} - \frac{\mathbf{A}^4}{\mathbf{4}} + \cdots + (-\mathbf{1})^{\mathbf{n}+\mathbf{1}}\frac{\mathbf{A}^\mathbf{n}}{\mathbf{n}} + \cdots$$

where $\mathbf{A}$ is a real matrix.
We know that the series is convergent if $\rho(\mathbf{A}) < \mathbf{1}$.
We will show that the topological ε-algorithm is able to accelerate the convergence of the **partial sums of this series** and also . . . to **converge** to an approximation of the value, when the **series is divergent!**
We consider a random matrix B of dimension $N = 50$, we choose a value r, and we define $A = r \times B/\rho(B)$.

We obtain for the **AM** the results of the next Figure (on the left $r = 0.9, k = 4$, and on the right $r = 1.2, k = 4$). The linear functional $\mathbf{y}$ induces the **trace of a matrix**.



AM, $r = 0.9, k = 4$



AM, $r = 1.2, k = 4$

Fredholm integral equation

We consider the following nonlinear **Fredholm integral equation of the second kind** with a given kernel K

$$\mathbf{u}(\mathbf{t}) = \int_{\mathbf{a}}^{\mathbf{b}} \mathbf{K}(\mathbf{t}, \mathbf{x}, \mathbf{u}(\mathbf{x}))\, \mathbf{dx} + \mathbf{f}(\mathbf{t}), \quad \mathbf{t} \in [\mathbf{a}, \mathbf{b}].$$

Fredholm integral equation

We consider the following nonlinear **Fredholm integral equation of the second kind** with a given kernel K

$$\mathbf{u}(\mathbf{t}) = \int_{\mathbf{a}}^{\mathbf{b}} \mathbf{K}(\mathbf{t}, \mathbf{x}, \mathbf{u}(\mathbf{x}))\, \mathbf{dx} + \mathbf{f}(\mathbf{t}), \quad \mathbf{t} \in [\mathbf{a}, \mathbf{b}].$$

The integral is approximated by a **quadrature formula** at points x_i. Then, u is computed at the points $t_i = x_i$ which leads to a system of **nonlinear equations**.

Fredholm integral equation

We consider the following nonlinear **Fredholm integral equation of the second kind** with a given kernel K

$$\mathbf{u}(\mathbf{t}) = \int_{\mathbf{a}}^{\mathbf{b}} \mathbf{K}(\mathbf{t}, \mathbf{x}, \mathbf{u}(\mathbf{x}))\, \mathbf{dx} + \mathbf{f}(\mathbf{t}), \quad \mathbf{t} \in [\mathbf{a}, \mathbf{b}].$$

The integral is approximated by a **quadrature formula** at points x_i. Then, u is computed at the points $t_i = x_i$ which leads to a system of **nonlinear equations**.
This system is solved by the GSM or by accelerating Picard iterations by the AM.

Fredholm integral equation

We consider the following nonlinear **Fredholm integral equation of the second kind** with a given kernel K

$$\mathbf{u}(\mathbf{t}) = \int_{\mathbf{a}}^{\mathbf{b}} \mathbf{K}(\mathbf{t}, \mathbf{x}, \mathbf{u}(\mathbf{x}))\, \mathbf{dx} + \mathbf{f}(\mathbf{t}), \quad \mathbf{t} \in [\mathbf{a}, \mathbf{b}].$$

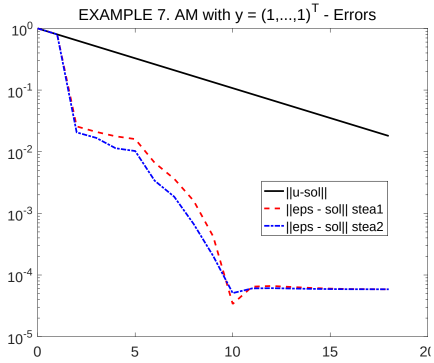
The integral is approximated by a **quadrature formula** at points x_i. Then, u is computed at the points $t_i = x_i$ which leads to a system of **nonlinear equations**.

This system is solved by the GSM or by accelerating Picard iterations by the AM.

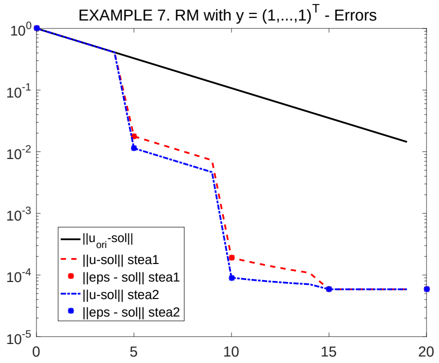
Consider the following example

$$\mathbf{u}(\mathbf{t}) = \mathbf{t}^2 \int_0^1 \frac{\mathbf{x}^2}{1 + \mathbf{u}^2(\mathbf{x})}\, \mathbf{dx} + (1/2 - \ln 2)\mathbf{t}^2 + \sqrt{\mathbf{t}},$$

whose solution is $\mathbf{u}(\mathbf{x}) = \sqrt{\mathbf{x}}$.


EXAMPLE 7. AM with y = (1,...,1)^T - Errors
||u-sol||
||eps - sol|| stea1
||eps - sol|| stea2
10^0
10^-1
10^-2
10^-3
10^-4
10^-5
0
5
10
15
20

AM


EXAMPLE 7. RM with y = (1,...,1)^T - Errors
||u_ori-sol||
||u-sol|| stea1
||eps - sol|| stea1
||u-sol|| stea2
||eps - sol|| stea2
10^0
10^-1
10^-2
10^-3
10^-4
10^-5
0
5
10
15
20

RM

Lotka-Volterra equations

The ε-algorithm is related to the **Lotka-Volterra equations**.

Lotka-Volterra equations

The ε-algorithm is related to the **Lotka-Volterra equations**.

Instead of transforming a **sequence $\mathbf{S_n}$** when **$\mathbf{n}$** goes to infinity, one can transform a **function $\mathbf{f(t)}$** when **$\mathbf{t}$** goes to infinity.

Lotka-Volterra equations

The ε-algorithm is related to the **Lotka-Volterra equations**.

Instead of transforming a **sequence $\mathbf{S_n}$** when **$\mathbf{n}$** goes to infinity, one can transform a **function $\mathbf{f(t)}$** when **$\mathbf{t}$** goes to infinity.

The corresponding Shanks transformation and the **confluent forms of the ε -algorithms** are also related to the **Lotka-Volterra equations**.

References

- Scalar Shanks transformation

  D. Shanks, An analogy between transient and mathematical sequences and some nonlinear sequence-to-sequence transforms suggested by it. Part I, Memorandum 9994, Naval Ordnance Laboratory, White Oak, July 1949.

  D. Shanks, Non linear transformations of divergent and slowly convergent sequences, J. Math. and Phys., 34 (1955) 1–42.
- The scalar ε-algorithm

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

References cont'd

- The topological Shanks transformation and its ε-algorithm
  C. Brezinski, Généralisation de la transformation de Shanks, de la table de Padé et de l'ε–algorithme, Calcolo, 12 (1975) 317–360.
- Derivation and analysis of the STEAs
  C. Brezinski, M. Redivo–Zaglia, The simplified topological ε–algorithms for accelerating sequences in a vector space, SIAM J. Sci. Comput., 36 (2014) A2227–A2247.
- Software and applications (Matlab in NUMERALGO of NETLIB: na44)
  C. Brezinski, M. Redivo–Zaglia, The simplified topological ε–algorithms: software and applications, Numer. Algorithms, 74 (2017) 1237–1260.
- Solution of integral equations
  C. Brezinski, M. Redivo–Zaglia, Extrapolation methods for the numerical solution of nonlinear Fredholm integral equations, submitted.

References cont'd

- Lotka-Volterra equations

  C. Brezinski, Forme confluente de l'ε-algorithme topologique. Numer. Math., 23 (1975) 363-370.

  C. Brezinski, Cross rules and non-Abelian lattice equations for the discrete and confluent non-scalar epsilon-algorithms. J. Phys. A: Math. Theor., 43 (2010) 205201.

  C. Brezinski, Y. He, X.-B. Hu, J.-Q. Sun, H.-W. Tam, Confluent form of the multistep epsilon-algorithm, and the relevant integrable system. Stud. Appl. Math., 127 (2011) 191-209.

  C. Brezinski, Y. He, X.-B. Hu, M. Redivo Zaglia, J.-Q. Sun, Multistep epsilon-algorithm, Shanks' transformation, and the Lotka-Volterra system by Hirota's method. Math. Comput., 81 (2012) 1527-1549.

  C.Brezinski, M. Redivo Zaglia, Shanks function transformations in a vector space. Appl. Numer. Math., 116 (2017) 57–63.

Thank you!